2007

Role-Based Access Control for the Open Grid Services Architecture - Data Access and Integration (OGSA-DAI)

Anil L. Pereira
Wright State University

ROLE-BASED ACCESS CONTROL FOR THE OPEN GRID SERVICES ARCHITECTURE - DATA ACCESS AND INTEGRATION (OGSA-DAI)

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

By

ANIL L. PEREIRA
M.S., Wright State University, 2002
B.E., Bombay University, 1999

2007
Wright State University

WRIGHT STATE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

April, 4, 2007

I HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER MY SUPERVISION BY Anil L. Pereira ENTITLED Role-based Access Control for the Open Grid Services Architecture - Data Access and Integration (OGSA-DAI) BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Doctor of Philosophy.

Soon M. Chung, Ph.D.
Dissertation Director

Thomas Sudkamp, Ph.D.
Ph.D. Program Director of
Computer Science and Engineering

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

Committee on
Final Examination

Soon M. Chung, Ph.D.

Krishnaprasad Thirunarayan, Ph.D.

Natsuhiko Futamura, Ph.D.

Henry Chen, Ph.D.

Chansu Yu, Ph.D.

ABSTRACT

Pereira, Anil L. Ph.D., Department of Computer Science and Engineering, Wright State University, 2007. Role-based Access Control for Grid Data Resources in the Open Grid Services Architecture - Data Access and Integration (OGSA-DAI).

Grid has emerged recently as an integration infrastructure for the 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. In this dissertation, role-based access control (RBAC) systems for heterogeneous data resources in Data Grid systems are proposed. The Open Grid Services Architecture - Data Access and Integration (OGSA-DAI) is a widely used framework for the integration of heterogeneous data resources in Grid systems.

However, in the OGSA-DAI system, access control causes substantial administration overhead for resource providers in VOs because each of them has to manage the authorization information for individual Grid users. Its identity-based access control mechanisms are severely inefficient and too complicated to manage because the direct mapping between users and privileges is transitory. To solve this problem, (1) the Community Authorization Service (CAS), provided by the Globus toolkit, and (2) the Shibboleth, an attribute authorization service, are used to support RBAC in the OGSA-DAI system. The Globus Toolkit is widely used software for building Grid systems.

Access control policies need to be specified and managed across multiple VOs. For this purpose, the Core and Hierarchical RBAC profile of the eXtensible Access Control

Markup Language (XACML) is used; and for distributed administration of those policies, the Object, Metadata and Artifacts Registry (OMAR) is used. OMAR is based on the e-business eXtensible Markup Language (ebXML) registry specifications developed to achieve interoperable registries and repositories.

The RBAC systems allow quick and easy deployments, privacy protection, and the centralized and distributed management of privileges. They support scalable, interoperable and fine-grain access control services; dynamic delegation of rights; and user-role assignments. They also reduce the administration overheads for resource providers because they need to maintain only the mapping information from VO roles to local database roles. Resource providers maintain the ultimate authority over their resources. Moreover, unnecessary mapping and connections can be avoided by denying invalid requests at the VO level. Performance analysis shows that our RBAC systems add only a small overhead to the existing security infrastructure of OGSA-DAI.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1
Introduction

1.1 Data Grid

Grid has emerged recently as an integration infrastructure for the 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. It provides a distributed system middleware that allows different communities to access and share data, networks, and other resources in a controlled and secure manner [1]. Data Grids facilitate the management of distributed heterogeneous data. The burden of managing the operations is removed from the user. The collective operations required are all managed by the system via a single sign-on and uniform querying mechanism for the user. The motivation behind such a system is to address the following considerations [2]: (1) Large data set size, geographic distribution of users and resources, and computationally intensive analysis results in complex and stringent performance demands that are not satisfied by any existing data management infrastructure; (2) No integrating architecture exists that allows us to identify requirements and components common to different systems and hence apply different

technologies in a coordinated fashion to a range of data-intensive large-scale application domains. Current technology cannot easily handle these scenarios which require the coordinated sharing of data and resources across multiple organizations. It either does not accommodate the range of resource types or does not provide the flexibility and control on sharing relationships [3].

Scientific and Business communities are increasingly collaborating, and this is giving rise to the need for more sophisticated technologies for data and resource sharing. Data Grids reduce hardware and software costs by enabling the secure exchange of programs and data between collaborating organizations. Without a Data Grid, a separate set of resources are purchased (and managed) in a demilitarized zone (DMZ) behind a completely separate firewall. By using Data Grid technology, there is no need to build a separate DMZ [4].

1.2 Motivation

User authorization is one of the most challenging issues in Data Grids. Current authorization mechanisms cannot address all the issues that arise in dynamic Grid environments which often encompass multiple organizations, each with its own security policy [5]. Traditional means of security administration that involves manual editing of policy databases or issuance of credentials cannot meet the demands of these dynamic scenarios [6]. There will be a profound impact on the security of distributed systems by using a Data Grid system. In traditional systems, the focus of security mechanisms has

been to protect the system from its users and, in turn, to protect data maintained by the system on behalf of each user. While such protection remains important for Data Grid applications, Data Grids introduce the extra requirements of protecting applications and user data from the systems on which parts of an application will execute [7]. Also, traditional network security research has focused primarily on two-party client-server interactions with relatively low performance requirements. Data Grid applications frequently involve many more entities, impose stringent performance requirements and involve more complex activities, such as collective operations and the downloading of code [8].

The typical identity-based authorization used in Grids today is not scalable because authorization information should be maintained for each user. In role-based access control (RBAC) [9, 10] permissions are associated with roles, and users are assigned appropriate roles, thereby acquiring the roles' permissions [11]. Hence, RBAC is quite scalable since authorization information is associated with roles, not with individual users. RBAC shows clear advantages over traditional access control models in Grid environments, because it allows a uniform representation of diverse security policies and ensures that no security violations occur during inter-domain accesses [5]. None of the current access control systems in Grids provide comprehensive support for RBAC.

The Data Access and Integration Services Working Group (DAIS-WG) of the Global Grid Forum (GGF) established standards for Grid interface to data resources [12]. The Open Grid Services Architecture - Data Access and Integration (OGSA-DAI) [13]

provides the first implementation for these standards. OGSA-DAI is a middleware infrastructure for accessing and controlling data sources and sinks. Though OGSA-DAI is widely used, its access control mechanisms are not scalable and cause substantial overhead for resource providers in VOs because each of them has to manage a role-map file containing authorization information for individual Grid users.

The Community Authorization Service (CAS) [14] and the Shibboleth [15] are authorization services that have several advantages over other authorization services used in Grids, and they both use the Security Assertion Markup Language (SAML) [16] standard. None of the other authorization services in Grids use a standard format to express authorization assertions. CAS records user groups and their permissions on resources, and it targets access control for computational and file-based storage resources. CAS is part of the Globus Toolkit [8]. The Globus Toolkit provides a set of basic services to establish a Grid system and it has a wide support base in the Grid community. Shibboleth is designed to provide user attributes to requesting resources and it targets access control for internet based resources. Shibboleth has a wide support base in the Internet2 community. SAML is used to express authentication and authorization assertions between different security domains.

1.3 Contribution

In this dissertation two RBAC systems for heterogeneous data resources in Data Grids are proposed. The first system uses CAS as the main building block. This system is

scalable in terms of the number of users and VOs; it can be quickly and easily deployed; and it provides centralized privilege management and delegation via roles. The second system uses Shibboleth as its main building block. This system is scalable in terms of the number of access requests in addition to the number of users and VOs; it is robust as there is no single point of failure; it supports the distributed management of privileges and fine-grain attribute release policy; and it provides privacy protection for users, in addition to dynamic delegation via roles. We also use the Core and Hierarchical RBAC profile of the eXtensible Access Control Markup Language (XACML) [17] to specify access control policies for multiple VOs. XACML is a standard of the Organization for the Advancement of Structured Information Standards (OASIS) for describing access control policies uniformly across different security domains [18]. The Core and Hierarchical RBAC profile of XACML defines how the ANSI core and hierarchical RBAC standard [19] can be specified in XACML. For the distributed storage and administration of XACML policies, we propose the use of the Object, Metadata and Artifacts Registry (OMAR) which provides an implementation of the OASIS e-business eXtensible Markup Language (ebXML) registry specifications. The ebXML registry specifications are developed to achieve interoperable registries and repositories with an interface that enables submission, query and retrieval. The main contributions of this dissertation are outlined as follows:

- The RBAC systems will support a wide range of security policies using role-privileges, role hierarchies, delegations, and constraints. It is shown how the CAS policy statements and SAML assertions can be used to support RBAC. The RBAC profile of XACML has several drawbacks for the access control in Grids. It does not

address dynamic delegation of rights and dynamic user-role assignments. They are supported in our system by using OMAR.

- With the proposed RBAC systems resource providers will have to maintain only the mapping information from VO roles to local roles and the local policies, thus the administration overhead is reduced. Furthermore, the resource providers can grant or refuse the access requests of specific users by maintaining their authorization information separately. This enables the resource providers to have the ultimate authority over their resources. Also, unnecessary authentication, mapping and connections can be avoided by denying invalid requests at the VO level. The 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 integration of the systems with OGSA-DAI will bring several advantages (as noted above) into its authorization infrastructure. Enhancements have been proposed to the role-map files so that they can also contain mapping information from VO roles to local database roles, and local policy. This dramatically reduces the number of entries to be managed in the role-map files and updates to them need to be made far less frequently. The implementation on the client side has been extended to request and delegate policy assertions. The server-side has been extended to parse the policy assertions to obtain the VO roles. The server also verifies the capabilities associated with a VO role against the local policies of the resource provider and maps it to a local database role. The performance evaluation shows that not much extra time is required to set up the security contexts between clients and servers.

1.4 Outline of Dissertation

The rest of the dissertation is structured as follows: Chapter 2 provides detailed background in terms of the major components used such as the Globus Toolkit, OGSA-DAI, CAS and Shibboleth. We discuss the security issues related to these components and Data Grids in general. We explain the need for RBAC, and provide the required concepts and foundation for our research. Chapter 3 reviews our work of the RBAC system for Grid Databases using CAS and includes performance analysis of the system. Chapter 4 reviews our work of the RBAC system for Grid Databases using Shibboleth and also includes performance analysis of the system. Chapter 5 has some conclusions.

Chapter 2
Background

2.1 Existing Middleware for Data Grids

Distributed data resources can be diverse in their formats, schema, quality, access mechanisms, ownership, access policies, capabilities, and authentication and authorization mechanisms. To efficiently manage these, a Data Grid needs technical solutions and standards for data discovery and access, data exploration and analysis, resource management, and security [1, 20, 21]. The Globus Toolkit [8] provides a set of basic services to establish a Grid system. The Open Grid Services Architecture – Data Access and Integration (OGSA-DAI) [13] is an existing middleware implementation that is widely used for the integration of heterogeneous data resources in Grid systems.

2.1.1 Globus Toolkit

The basic Grid middleware components provided by the Globus Toolkit are: (1) Grid Security Infrastructure (GSI): authentication and related security services; (2) Globus Resource Allocation Manager (GRAM): resource allocation and process management; (3) Meta Computing Directory Service (MDS): distributed access to the

structure and state information of a system; (4) Globus Executable Management (GEM): construction, caching and location of executables; (5) Global Access to Secondary Storage (GASS): remote access to data via sequential and parallel interfaces; (6) Nexus: unicast and multicast communication services; (7) Heart Beat Monitor (HBM): monitoring of the health and status of system components; and (8) General Purpose architecture for Reservation and Allocation (GARA): reservation of resources and monitoring of reservations. In addition to these components, the Globus Toolkit implements the Open Grid Service Architecture (OGSA). OGSA integrates Grid and Web services technologies and defines standard interfaces and behaviors for distributed system integration and management [1]. The toolkit has recently been aligned with the Web Services Resource Framework (WSRF) [22]. WSRF defines conventions for managing the state in distributed systems based on Web services. For each of these components, a C and/or Java Application Programming Interface (API) is available for developers [23].

2.1.2 Open Grid Services Architecture – Data Access and Integration (OGSA-DAI)

Grid integrates several communities of resource providers and resource consumers. This integration can be technically challenging because of the need to achieve various qualities of service when running on top of different native platforms. The Open Grid Services Architecture (OGSA) addresses these challenges and defines uniform exposed service semantics, the Grid Service [24]. Version 3 of the Globus Toolkit and its

accompanying Grid Security Infrastructure (GSI) provide the first implementation of OGSA mechanisms and cast security functions as OGSA services. This version of the Globus Toolkit also publishes service security policies and specifies standards for interoperability [7].

Current research in the area of Grid databases is undertaken by *Project Spitfire* associated with the European Data Grid [25] and the *Open Grid Services Architecture - Data Access and Integration* (OGSA-DAI) [13]. Project Spitfire provides access control based on authorization tags specified within XML-based query files. These tags are mapped by a database resource to local roles via a role-database that it maintains. A drawback of their approach is that the role-database contains the mapping to local database roles for all Grid users that have access to that database resource. Multiple entries in multiple role-databases may need to be updated if new Grid users are allowed to access multiple data resources or if the access privileges of current users change.

The Database Access and Integration Services Working Group (DAIS-WG) of the Global Grid Forum (GGF) is currently establishing the standards for Grid interface to data resources [12]. OGSA-DAI is a widely used middleware infrastructure, aligned with the GGF's OGSA vision, to facilitate uniform access to data resources using a service-oriented architecture (SOA) [13]. OGSA-DAI provides activities to access relational, XML databases, and indexed files, etc. It also provides data translation and third-party delivery activities [13]. OGSA-DAI enables client applications to submit request documents in order to perform a set of tasks on a remote data resource.

OGSA-DAI provides a set of core activities that implement the basic functionality needed to interact with a data resource, and it is easy for users to add new activities that operate within the OGSA-DAI framework [26]. OGSA-DAI users can extend OGSA-DAI web services to expose their own data resources and to support application-specific functionality. OGSA-DAI also provides a consistent transactional framework and facilities to allow developers to add transactional behavior to their activities.

OGSA-DAI has over 1100 registered users and projects which require continuously available data access and integration services [13]. OGSA-DAI is used by a number of large projects both within the US and UK to satisfy their data access and integration requirements. In addition to this, the OGSA-DAI project is working in close collaboration with other major Grid middleware providers, such as Globus, IBM and the UK Open Middleware Infrastructure Institute (OMII), to ensure that OGSA-DAI integrates seamlessly with their products.

2.2 Issues for Access Control in Grids

The overall direction for access control architectures in Grid computing is toward the need for leveraging IT infrastructure as it emerges. Integration with Web services and hosting environment technologies introduces opportunities to leverage emerging security standards and technologies such as the Security Assertion Markup Language (SAML) [16] and Web Services Security (WSS) [27]. Participating organizations within a Grid often have significant investment in existing security mechanisms and infrastructure, and

Grid services could be built on sophisticated container-based hosting environments such as J2EE or .Net. Grid security mechanisms should interoperate with, rather than replace, those mechanisms [6]. Most security functionality should be placed in the hosting environments, so that application development will be simplified and security functionality can be upgraded independently of applications [6]. The WSS specifications address these issues. WSS is a standard mechanism for interoperability and enables the interaction between different platforms and security models. WSS standard can be used to transport credentials from a client to a server, such as the ones represented by SAML attribute assertions [28]. Users need globally defined names that are recognized at all sites they access. A user's identity needs to be passed securely and transparently between sites as jobs progress [29]. Users must be able to access resources dynamically without any administrator intervention. These resources must be coordinated properly and must interact securely with other services. Thus, resources must have global identities, and they should be accessed without violating their local policies.

Significant challenges remain for cross-domain auditing and privacy management [30]. An audit mechanism can be used to determine whether or not the access control policies have been administered properly. The audit mechanism is responsible for producing records which track security related events [31]. And, for this purpose, it is essential to keep a log of the access requests and the enforced security policies. In traditional systems, the audit mechanism is local to each server; however, on the Grid, either the audit mechanism should be distributed or the audit records should be transmitted to a location where a higher level view of the system can be constructed [32].

Standards are required to facilitate the audit and to reconcile different audit trails that are distributed among different organizations. It is extremely difficult to browse the audit logs if they are in different formats and in different administrative domains. Also, the access control mechanism should be able to match the audit entries in different audit logs and administrative domains.

Auditing also depends on authentication because audit records usually associate individuals with the actions they have taken, and the identity of the user must be determined if these entries are to be trusted [32]. The user identity can be used to identify the user who initiated the request. The request can be logged at the resource along with the mapping information and the subsequent actions performed. This information can be used to find patterns that fit the profile of a system intrusion or the activities that do not fit the profiles of legitimate users [32]. However this information could affect the privacy of users. For example, by examining the information logged at various sites with respect to users belonging to a particular research group, it is possible to infer their data access patterns and thus obtain information about their work. To solve this problem, a user could be issued a set of pseudonym identities [33] and he/she could access each site with a different identity in the set. The information that binds the set of pseudo identities to the user identity should be maintained securely and can be used when security violations occur.

2.3 Need for Role-Based Access Control for Grid Data Resources

In Grids, both users and resources are dynamic. Furthermore, those users and 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. Trust should be established not only among users and resources, but also among the resources themselves so that they can be coordinated. These trust domains can span across multiple organizations and must adapt dynamically as participants join or leave and resources are accessed or released [6]. Resource providers must understand and support the mechanisms and policies that are not strictly under their control.

It is desirable to group users and resources that need to be coordinated towards a common goal into virtual organizations (VOs). The key requirement is to design access control mechanisms for these VOs, which can interoperate with existing local security infrastructures and allow resource providers to have the ultimate control over their resources. A VO spanning across multiple sites can use a single security mechanism, but usually it needs to accommodate multiple security mechanisms [29]. While acknowledging and respecting the site autonomy, there are a number of requirements to be met for Grid security, in order to achieve the goals of the VOs.

Supporting role-based access control (RBAC) [9, 10] is desirable in 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 and ensures that no security violations occur during inter-domain accesses [5]. In RBAC, permissions are associated with roles, and users are assigned appropriate roles, thereby acquiring the roles' permissions [11]. 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, while the cost of associating users directly with permissions is proportional to $U \times P$, where U is the number of individuals in a role and P is the number of permissions required by the role [34, 35].

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 [30], which requires that a user be given no more privileges than necessary to perform a job [9]. It can be easily enforced by first identifying the roles in an organization correctly and then assigning only those privileges to each role that allow the role members to perform their tasks. 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 [10].

2.4 Drawbacks of the Access Control Mechanisms in OGSA-DAI

To date, most work on data storage, access and transfer on the Grid has focused on files [36], but the Grid can also be used to integrate various distributed heterogeneous databases and supports query/transaction processing on them through a uniform interface [36, 37]. The use of databases in Data Grids presents different security needs and access policies compared with the use of computational resources. For example, certain applications may be authorized to access only certain parts of the information in a database during a specific time interval. OGSA-DAI uses Access Control Lists (ACLs) for user authorization.

OGSA-DAI supports access control via an ACL held in a role-map file that maps individual Grid users to local database usernames and passwords. In this case, each resource provider has to maintain a role-map file to authorize access to its resources. This access control method is not suitable for VOs, especially in terms of scalability, because both users and resources are dynamic in VOs. Multiple entries in multiple role-map files may need to be updated if new users are allowed to access multiple data resources or if the access privileges of current users change, which is not unusual in Data Grids. This puts an unnecessary burden on the resource providers in managing the role-map files, especially when both the users and resource providers belong to multiple VOs. Furthermore, there are unnecessary overheads on the server side whenever users make

invalid requests. This is because users are authenticated, mapped and connected without first verifying their requests against their access privileges.

2.5 Incorporation of Existing Authorization Services in Data Grids

None of the current access control systems in Grids provide comprehensive support for RBAC. We will use the Community Authorization Service (CAS) and Shibboleth (along with the GridShib interface) to support RBAC in OGSA-DAI, as described later in Chapters 3 and 4. CAS and Shibboleth services have certain advantages over other authorization services for Grids, such as the Virtual Organization Management Service (VOMS) [38] and Akenti [39]. VOMS authorization assertions 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 [38]. Akenti does not provide support for dynamic delegation [40]. 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 [3].

PERMIS [40] 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 [28]. 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 [28].

SAML can be used to express authorization queries, and Extensible Access Control Markup Language (XACML) [18] can be used to express authorization policy statements. Except CAS and Shibboleth, which use the SAML standard, none of the other authorization services use a standard format to express authorization assertions. SAML is used to uniformly express the authentication and authorization assertions between different security domains. These assertions could contain the following three types of statements: (1) Authentication statements which assert that the user has been authenticated by the authorization service; (2) Attribute statements which can express the attributes of the user such as institutional affiliation, group membership, and so on; (3) Authorization decision statements which can assert how a user is allowed to access a resource.

2.5.1 Community Authorization Service (CAS)

The Community Authorization service (CAS) provides a scalable mechanism for specifying and enforcing complex and dynamic policies that govern resource usage within Grids. It allows resource providers to delegate some of the authority for maintaining fine-grain access control policies to communities, while still maintaining the ultimate authority over their resources [14].

A community runs a CAS server to keep track of its membership and fine-grain access control policies. A user accessing community resources contacts the CAS server, which delegates rights to the user based on the request and the user's responsibilities within the community. These rights are in the form of capabilities, which users can present at a resource to gain access on behalf of the community. The user effectively obtains the intersection of the set of rights granted to the community by the resource provider and the set of rights defined by the capabilities granted to the user by the community. The CAS server uses a backend database to store the capabilities of the users. The CAS architecture builds on the public key authentication and delegation mechanisms provided by the Globus Toolkit's Grid Security Infrastructure (GSI) [14].

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

CAS provides scalability in terms of the number of users and VOs. The CAS structure reduces the number of necessary trust relationships from $C \times P$ to $C+P$, when there are C consumers and P providers. Each consumer needs to be known and trusted by the CAS server, but not by each provider. Similarly, each provider needs to be known and trusted by the CAS server, but not by each consumer [14]. However, in terms of the actual number of access requests on resources, using a single CAS server may not be quite scalable. A single CAS server can be a bottleneck if a large number of users attempt to access it at the same time, and it can be a single point of failure. A possible solution for these problems depends on how frequently the community policies change. If the community policies do not change frequently, a single master server can be maintained to accept the changes and then routinely replicate the policies to one or more read-only slave servers. If the community policies change frequently, multiple peer servers can be used. All the servers update the policies, so that the failure of any one server will not lead to a loss of functionality [14].

However, when policies are changing dynamically, it is believed that complete centralization of policies (which can be realized by using CAS) can achieve better consistency. Also, in the case that a user credential is compromised, revocation is easier when a single CAS server is used because the user needs to be removed only from that server [14]. Even though CAS was designed primarily for fine-grain policies, it has been also shown to be capable of asserting coarse-grain group memberships [41, 42]. CAS comes packaged within the Globus Toolkit and is easily deployable.

2.5.2 Shibboleth and GridShib

In the GridShib project [33], they leverage the local security infrastructures of different organizations so that users can be authenticated to Grid resources by using the methods already supported at their home organizations. The goal of GridShib is to create a distributed authorization framework that supports anonymous interactions between users and, hence, protects their privacy. The rights of the users can be expressed using attributes such as institutional affiliation, group membership, or their role in collaboration [33]. Resource providers can make informed authorization decisions using these attributes unlike in 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 certain database records.

GridShib incorporates the Shibboleth [15], which is an Attribute Authority service, developed by the Internet2 community for cross-organization identity federation [33]. Shibboleth creates a distributed authorization infrastructure for web resources, and simplifies access control policies and makes them more scalable [43]. It enables anonymous interaction between users, thus protecting individual privacy while still providing basic security for resource providers [33]. 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 [29].

A target 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 Grid Service is authorized to see, from the Shibboleth service. These attributes are presented using SAML attribute statements. To determine the address of the Shibboleth attribute service, the Grid service obtains a pointer that will be placed in the user's proxy certificate [33]. The attributes obtained will then be used by the Grid service in making authorization decisions. These attributes will be passed securely through a trust relationship to the Grid service. To provide anonymity in the 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 doesn't need to be released.

2.6 Concepts and Foundation for our Research Topics

In order to provide scalable and fine-grain access control in Data Grids, we will enhance the access control mechanisms in OGSA-DAI to allow users to be assigned memberships on VO roles and to support role hierarchies and constraints. We will show that the SAML assertions of CAS and Shibboleth can provide the user's privileges directly in addition to the VO roles. These assertions could be obtained by the resource using either a push model or a pull model. In the push model, the user can directly obtain the permissions from the authorization server and pass them to the target resources at the time of making a request. The resource will verify the authenticity of the user and then authorize the user based on the permissions obtained, provided the authority that issued

them is trustworthy. The advantage of the push model is that the user can explicitly select a role. Also, in the case that the user and the authorization service belong to the same organization and are protected by a firewall, the push model should be deployed because the resources may not be able to contact the authorization service directly. Some authorization services, like Akenti, support the pull model, where the user is authenticated by a target resource. The target resource contacts the authorization server to obtain the user's permissions. An advantage of the pull model is that it can be deployed easily because users do not need to interact with the authorization service [33].

Specification of VO policies by CAS and Shibboleth will allow for authorization decisions to be made easily based on the user's request and VO policies. In case the user does not possess the required privileges, the access can be denied by CAS and Shibboleth without involving the resource providers. For OGSA-DAI, this eliminates authentication, mapping and connection overheads on the resource providers in case the user's request is not valid. The resource providers then need to maintain only the mapping information from the VO roles to local database roles and the local policies, thus the number of entries to be managed in the role-map files will be reduced dramatically. When users join/leave collaborations, the resource providers do not have to bother about updating their information in the role-map files, because the authorization service can just grant/revoke their memberships on the VO roles. Different VOs may have different role structures. Furthermore, the resource providers can grant or refuse the access requests of specific users by maintaining their authorization information separately in the role-map

files. This will enable the resource providers to maintain ultimate authority over their resources.

2.6.1 Enforcement of VO Policies

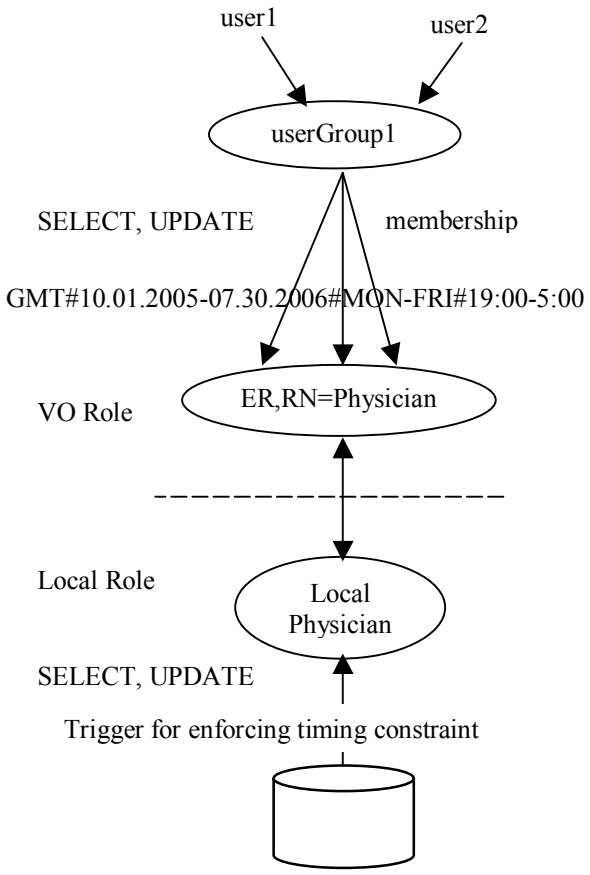


Figure 2.1: Mapping VO Roles to Local Roles

In our systems, the decision to map a VO role to a local role lies in the hands of the resource provider. The assignment of privileges to the local role and specifying constraints on it will also be the responsibility of the resource provider. For example, as

shown in Figure 2.1, a resource provider could decide to map the ER,RoleName=Physician role, where ER could be an Emergency Team that forms a VO across several hospitals, to a local role that allows the SELECT and UPDATE operations to be performed between 19:00 and 5:00 GMT, Monday through Friday during 10.01.2006–07.30.2006 (say, only for those patients affected by a natural disaster). This timing constraint could be enforced by a database trigger, which executes an action automatically on the occurrence of a predefined event. The privileges and constraints associated with the local role can be negotiated between the VO and the resource provider. Alternatively, if local privileges and constraints have been fixed already, they can be made known to the VO.

The VO can restrict the policies further by specifying a subset of the privileges associated with the local role and/or specifying tighter constraints. For example, applications invoked by users in a Junior Physician VO role may be allowed to perform SELECT and UPDATE operations only between 21:00 and 5:00 GMT, instead of between 19:00 and 5:00 GMT. This scheme allows the VO to change privileges and constraints without involving the resource provider. However, these changes have to be enforced at the VO level. For example, the user's query and the current time can be examined in order to check the conformance with those changes. In this way, the resource provider does not have to create new local roles in addition to existing ones because both the original and restricted VO roles can be mapped to the same local role. Furthermore, the resource provider can enforce more restrictions in addition to those imposed by the VO policy; for example, restricting the access privilege of particular users based on their

institutional affiliation. For this purpose, the resource provider can maintain a separate list of users and deny their access by checking the Grid identity in the SAML assertion. This enables the resource provider to have the ultimate authority over its resources.

2.6.2 Distributing VO Policies among Resource Providers

If roles and privileges do not change often and VOs have a long lifespan, then it is feasible to distribute the VO policies among the various resource providers. The fine-grain privileges and constraints associated with the local role can be negotiated between the VO and the resource provider. But, the resource provider will have control over the actual assignment of fine-grain privileges to the local role and the specification of constraints on it. For example, a resource provider could grant permission to perform basic database operations (e.g., SELECT) on a particular database table. The resource provider could also grant permissions for more complex operations such as executing stored database procedures.

With our systems, a user can delegate a subset of his/her authorized VO roles to certain applications and services. In this case, the privileges associated with the delegated VO roles are the privileges associated with the corresponding local roles. As shown in Figure 2.2, after a negotiation with a VO, a resource provider could decide to map the Alpha,RN=Supervisor role (where Alpha is the name of the VO) to a local Supervisor role that allows the function updateInventory( ) to be performed between 9:00 and 5:00 GMT from Monday to Friday during 01.20.2006–07.30.2006. Another resource provider

could decide to map the same VO role to a local Employee role that allows the function viewInventory() to be performed between 19:00 and 5:00 GMT from Monday to Friday during 05.20.2006–07.30.2006.

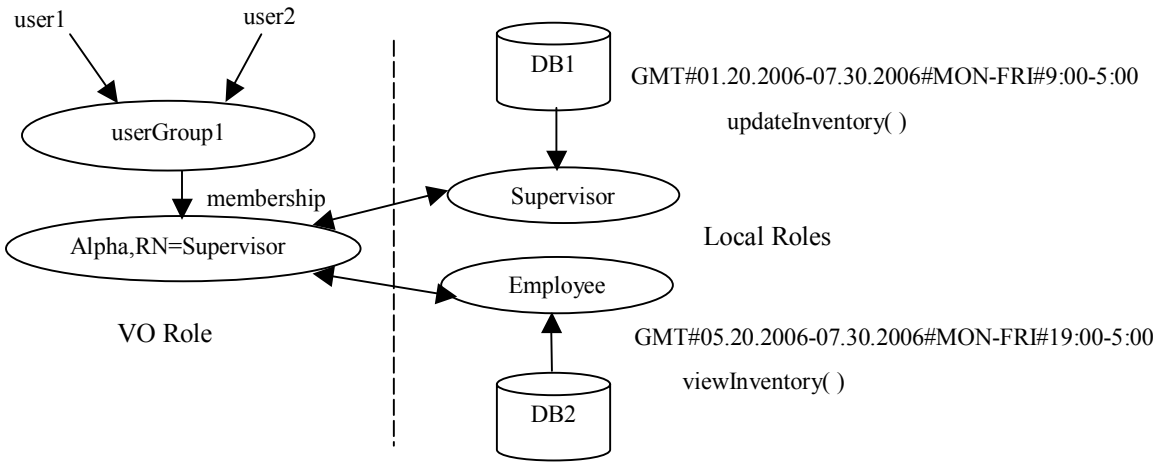


Figure 2.2: Distributing VO Policy among the Resource Providers

Chapter 3

RBAC with CAS in the OGSA-DAI System

The goal of this research topic was to design and implement a Role-based Access Control (RBAC) system with CAS, which can be quickly and easily deployed at various sites using the Open Grid Services Architecture – Data Access and Integration (OGSA-DAI). We demonstrate that our system can support RBAC for multiple virtual organizations (VOs) to access Grid databases within the OGSA-DAI framework. Our system extends the access control mechanism supported by OGSA-DAI to allow users to be assigned memberships on VO roles, to assign privileges and specify constraints on those roles, and to allow role hierarchies.

In our system, CAS maintains the security policies of VOs, grants users' memberships on VO roles, and then authorizes them in those roles. The resource providers need to maintain only the mapping information from VO roles to local database roles and the local policies, thus the number of entries in the role-map file is reduced 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 itself. This eliminates unnecessary authentication, mapping and connection overheads on the resource providers. When users join/leave a VO, the resource providers do not have to bother about individually adding/removing their information in the role-map files because the CAS server can just grant/revoke their memberships on the VO roles. Furthermore, the resource providers can grant or refuse the access requests of specific users by maintaining their authorization information separately in the role-map files. This enables the resource providers to have the ultimate authority over their resources.

We have implemented the proposed system and analyzed its performance. In our implementation, users obtain CAS credentials based on user credentials. The user credential is formed by an X.509 certificate and the associated public/private keys and is issued by a Certificate Authority (CA) trusted by all entities in a Grid [44]. The CAS credentials contain the authorization information for the user in terms of his/her VO roles. We have extended the client-side implementation of OGSA-DAI to pass the CAS credential. The server-side has been extended to parse the CAS credential to obtain the VO roles. The server also verifies the capabilities associated with that VO role against the local policies of the resource provider and maps it to a local database role. We have evaluated our solution in terms of the overheads incurred when security contexts are set up between a client and a server. This has been done with respect to the original security mechanism in OGSA-DAI.

The organization of the chapter is as follows: Section 3.1 describes how users can access a data resource using OGSA-DAI Services. In Section 3.2, we describe the drawbacks for the existing approach for RBAC with CAS. In Section 3.3, we present our RBAC system using CAS in OGSA-DAI. Section 3.4 describes the implementation details, and Section 3.5 describes the results of performance analysis.

3.1 OGSA-DAI Services

In order to expose physical data resources to the Grid, by extending the interfaces defined by Open Grid Services Infrastructure (OGSI) [45], OGSA-DAI introduced the following services [46]: (1) Grid Data Service Factory (GDSF): Represents a data resource, and exposes its capabilities and metadata. (2) Grid Data Service (GDS): Created by a GDSF and holds the client session with the data resource. (3) DAI Service Group Registry (DAISGR): Clients can discover service/data by locating GDSFs registered with a DAISGR.

Figure 3.1 shows how clients can access data resources using OGSA-DAI. The client first contacts the DAISGR and gets information about the registered GDSFs. The client then contacts the desired GDSF and makes a request for the creation of a GDS. Once the GDS is created, it authorizes the client and establishes a JDBC connection to the underlying database. The client can then submit queries on the database and retrieve results. The client authorization process is discussed in detail in Section 2.1.

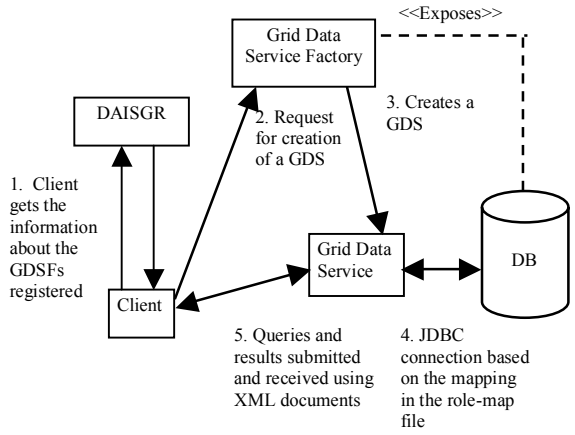


Figure 3.1: Accessing a Data Resource through OGSA-DAI

CAS was initially designed to record user groups and their permissions, but in our system, CAS has been incorporated to support RBAC within the OGSA-DAI framework. With our system, the resource providers can delegate the fine-grain authorization to CAS which will grant users' memberships on VO roles and then authorize them in those roles.

3.2 Drawbacks of the Existing Approach for RBAC with CAS

A proposed approach for supporting RBAC with CAS is the use of rights associated with a role to access role-specific resources [41]. The role of a user is presented in a hierarchical form. For example, Alpha/admin indicates the administrator role of a virtual organization Alpha. Alpha could be the name of a project undertaken by collaborating organizations. 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. One of the key aspects of RBAC is that it allows the specification of constraints on roles [10]. However, the approach proposed in [41] does not address this aspect of RBAC. Another drawback is that for a user to act in multiple roles, multiple CAS proxy certificates have to be created.

Most systems do not enforce the Principle of Least Privilege [47]. An application must be delegated only those privileges required for completing a certain set of tasks, otherwise the application should be totally trusted to do no more than required. In Grids, 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. When the method proposed in [41] is used in CAS, the Principle of Least Privilege is not always enforced. Users authorized to act in a role may be granted some privileges in addition to those assigned to that role. This is because both roles and privileges are set up in the same way. In particular, a role is considered as a resource, and a user group is given the "member" right on a role, in the same way that a user group is given the "read" right on a resource such as a file. With the current implementation of CAS, a user belonging to multiple groups can request and be authorized any combination of roles and privileges from one or more of those groups at the same time.

The following example illustrates this problem. The VO Alpha may have a policy in which programmers are allowed only read access to a particular file while supervisors are allowed read/write access. To implement this policy based on the method proposed in [41], two roles, "Alpha/programmer" and "Alpha/supervisor", can be created as shown in Figures 3.2 and 3.3. The "Alpha/programmer" role can be assigned the "read" right on "ftp://localhost/tmp/fileA.txt", and the "Alpha/supervisor" role can be assigned the "read" and "write" rights on "ftp://localhost/tmp/fileA.txt". As users of userGroup1 are given the "member" right on the role "Alpha/programmer", they can acquire the "read" right on "ftp://localhost/tmp/fileA.txt." Similarly, as users of userGroup2 are given the "member" right on the role "Alpha/supervisor", they can acquire the "read" and "write" rights on "ftp://localhost/tmp/fileA.txt". If a user "user1" is in both userGroup1 and userGroup2, and makes a request to act in the "Alpha/programmer" role with the "read" and "write" rights on "ftp://localhost/tmp/fileA.txt", CAS will authorize the request because "user1" is a member of both user groups. This authorization decision clearly violates the VO policy in terms of the Principle of Least Privilege, since a programmer is granted write access to a file while he/she is allowed only read access to it. If there is an application that analyzes data for programmers, it must be delegated only the read access to the file. Delegating the write access to the application can potentially result in an alteration of the file.

A possible refinement to this method is distributing the VO policies to the resource providers while keeping only the assignment of users to the VO roles within CAS. This method is applicable if roles and privileges do not change often and VOs have a long

lifespan. CAS is not used to associate the privileges with roles to access role-specific resources. Instead, the VO role is mapped to a local role, and the assignment of fine-grain privileges to the local role is the responsibility of the resource provider. The fine-grain privileges associated with the local role can be negotiated between the VO and the resource provider.

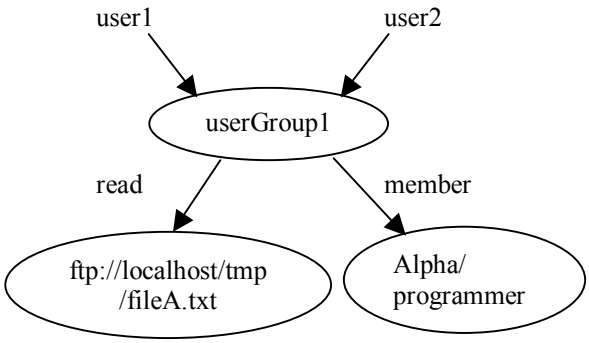


Figure 3.2: userGroup1 with a Role Alpha/programmer and Read Access to ftp://local host/tmp/fileA.txt

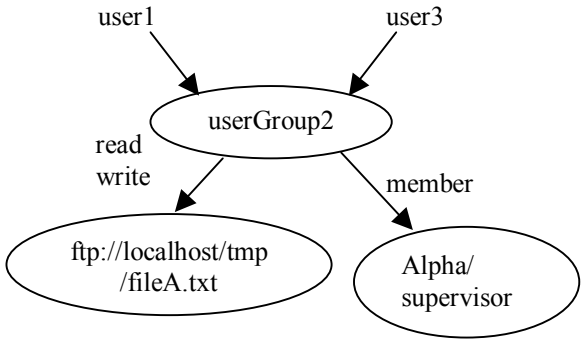


Figure 3.3: userGroup2 with a Role Alpha/supervisor and Read/Write Access to ftp://local host/tmp/fileA.txt

A user can delegate a subset of his/her authorized VO roles to certain applications and services. In this case, the privileges associated with the delegated VO roles are the privileges associated with the corresponding local roles. As shown in Figure 3.4, after a negotiation with a VO, a resource provider could decide to map the "Alpha/supervisor" role to a local "Supervisor" role that allows the function updateInventory( ) to be performed on a database (DB_1). Another resource provider could decide to map the same VO role to a local "Employee" role that allows the function viewInventory( ) to be performed on another database (DB_2). This method enforces the Principle of Least Privilege since a user can receive no more privileges for a VO role other than those tied with the corresponding local roles. However, a VO does not have the flexibility to update VO policy without contacting the resource providers because they control the assignment of privileges to the local roles.

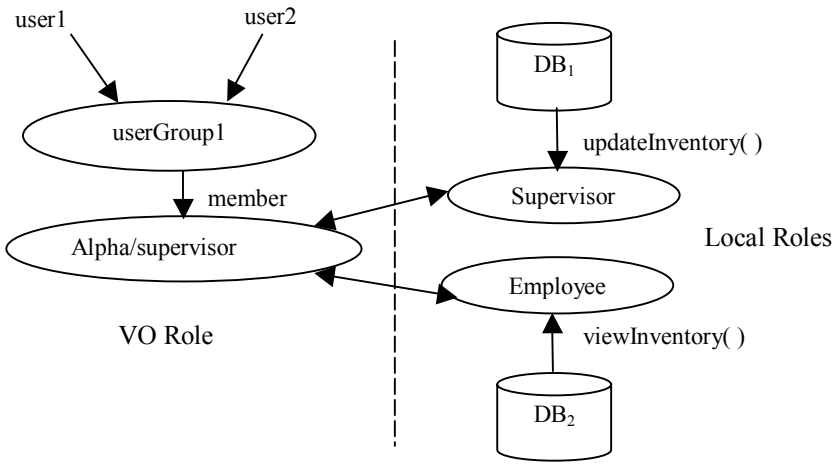


Figure 3.4: Distributing VO Policies to the Resource Providers

3.3 Our Proposed System for RBAC Using CAS

Specification of policies at the VO level allows authorization decisions to be made based on the user's request and VO policies. In case the user does not possess the required privileges, the access can be denied at the VO level itself without involving the resource providers. This eliminates authentication, mapping and connection overheads on the resource providers in case the request is not valid. Our proposed system is implemented using a newer version of CAS which supports SAML. Participating organizations within VOs may have different security models. So, it is important for these models to interoperate at different levels of trust, and SAML can be used to uniformly express the authorization assertions between different security domains.

The CAS server contains policy statements that specify who (which user or group) has the permission, which resource or resource group the permission is granted on, and what permission is granted [14]. The permission is denoted by a *service type* and an *action*. The *action* describes the operation (e.g., read, write or execute program), and the *service type* defines the namespace in which the *action* is defined (e.g., file). Different resource providers may recognize different service types, but all resource providers that recognize the same *service type* should have the same interpretation of that service type's actions [14].

To support RBAC using CAS, we define the role as a new *service type*, and each role name in the form of "[VOName{,SubgroupName}][,RN=rolename]"[1] as an *action*. Roles can be specified for any subgroup within a VO. For example, "Alpha,RN=Manager" indicates the Manager role for the Alpha VO, whereas "Alpha,Data,RN=Manager" indicates the Manager role for the Data subgroup of the Alpha VO.

For each role name, we can specify the actions (privileges and constraints) and some junior roles. Resources represented in the form, "URI{.Subcomponent}" are associated with usergroups in the CAS database. Thus, fine-grain authorization for resources can be allowed, where access control can be specified not only for the entire resource (e.g. database) but also for the subcomponents of a resource (e.g. table). For example, "http://130.108.17.176:8080/ogsa/services/ogsadai/SecureGridDataServiceFactory.Employee" indicates the Employee table in the database represented by the specified URI. This permits the members of a usergroup to access a resource in a specific role. We propose new service types and actions to assign privileges on roles, and to specify timing constraints as described in the following subsections. With these proposed ideas, privileges can be specified at fine-grain levels.

[1] The curly brackets {} indicate zero or more occurrences of their content and the square brackets [] indicate only one occurrence of their content.

3.3.1 Specifying Privileges and Timing Constraints on VO Roles

To assign privileges on a role, we define the role name as a *service type* and each privilege in the form of "privilege:operation" as an *action*. This allows the specification of a privilege in terms of the operation permitted for a specific role. For example, the role name "Alpha,RN=Manager" could have "privilege:select" as a privilege to execute the SELECT operation. Obviously, not only the basic database operations, but also complex operations, such as transactions and stored procedures, can be assigned as privileges.

To specify a timing constraint on a role, we define the role name as a *service type* and the timing constraint in the following form as an *action*:

"timing_constraint:[local/GMT] [Date#Day#Time]{;Date#Day#Time}" where

- Date can be "[FromDate-ToDate]{,FromDate-ToDate}"
- Day can be "[FromDay-ToDay]{,FromDay-ToDay}" or "[Day]{,Day}"
- Time can be "[FromTime-ToTime]{,FromTime-ToTime}"

For example, the role name "Alpha,RN=Manager" could have "timing_constraint:GMT#10.01.2005-07.30.2006#Mon-Fri#1:00-5:00,17:00-21:00", indicating that the user can act in that role only within the time intervals 1:00–5:00 and 17:00–21:00 GMT from Monday to Friday during 10.01.2005–07.30.2006.

3.3.2 Specifying Role Hierarchies

A role hierarchy defines a seniority relation between roles, whereby senior roles automatically acquire the permissions of the junior roles. In the role hierarchy diagrams [48], senior roles are placed at the top of the junior roles. According to the NIST standard for RBAC [48], there are two types of role hierarchies: limited hierarchy and general hierarchy. In the limited hierarchy, each senior role cannot have more than one junior role. On the other hand, in the general hierarchy, each senior role can have multiple junior roles. However, in both types, a junior role can have multiple senior roles. Examples of a limited hierarchy and a general hierarchy are illustrated in Figures 3.5 and 3.6, respectively.

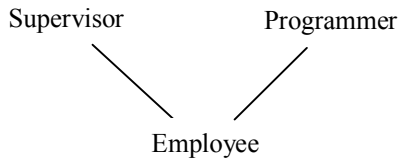


Figure 3.5: An Example of Limited Role Hierarchy.

The selection of the type of role hierarchy is made by the VO. To specify a role hierarchy, we define each senior role name as a *service type* and each junior role name in the form of "junior_role:[VOName{,SubgroupName}][,RN=rolename]" as an *action*. For example, in Figure 3.6, the senior role name "Alpha,RN=Manager" has "junior_role:Alpha,RN=Supervisor" and "junior_role:Alpha,RN=Programmer" as junior role names, and thereby inherits their privileges.

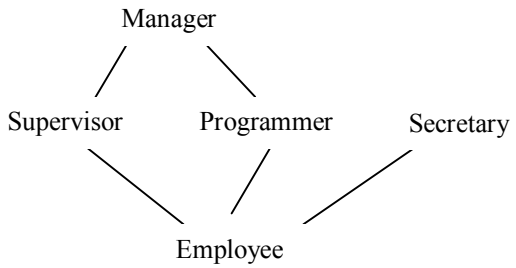


Figure 3.6: An Example of General Role Hierarchy.

The constraints on a junior role are also inherited by a senior role [28]. In our method, the timing constraint specified on a senior role would override those on the junior roles. If a timing constraint is not specified on a senior role, then it inherits the timing constraints of its junior roles. However, there should be no conflicts between the timing constraints on the junior roles. If such conflicts exist, then the concept of limited inheritance [10] can be used. With limited inheritance, a senior role can inherit only a subset of privileges of a junior role. The following example illustrates the concept of limited inheritance. As shown in the hierarchy of Figure 3.6, the Manager role is senior to both the Supervisor and Programmer roles. Managers can be prevented from inheriting specific privileges of the Supervisor role by defining a new role Supervisor′ as shown in Figure 3.7. Only those specific privileges not to be inherited by the Manager role can be assigned to the Supervisor′ role and the rest of the original set of privileges can be retained by the Supervisor role. The Supervisor′ role can inherit the privileges from the Supervisor role, thus acquires the entire set of privileges originally held by the Supervisor role. The Manager role can then inherit the privileges of the Supervisor role but not the privileges of the Supervisor′ role. Similarly, by creating the Programmer′ role, the

Manager role can be prevented from inheriting specific privileges originally held by the Programmer role.

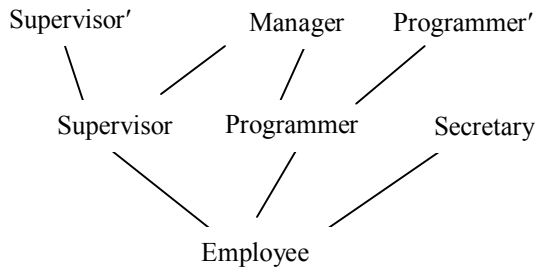


Figure 3.7: Limited Inheritance

To deal with conflicting constraints on junior roles, some modifications to the approach illustrated above are required. If conflicting timing constraints exist on the Supervisor and Programmer roles shown in Figure 3.6, then new timing constraints with no conflicts can be specified on those roles while each retains the entire set of its privileges. The Manager role can then inherit the privileges and new constraints on the Supervisor and Programmer roles. The original timing constraints of the Supervisor and Programmer roles can be specified on the Supervisor′ and Programmer′ roles, respectively. These timing constraints will then override the new constraints specified on the Supervisor and Programmer roles. The Supervisor′ and Programmer′ roles are not assigned any privileges and can inherit all the privileges from the Supervisor and Programmer roles, respectively. Thus the Supervisor′ and Programmer′ roles possess the set of timing constraints and privileges originally associated with the Supervisor and Programmer roles, respectively. While the original information of the Supervisor and

Programmer roles is retained, the Manager role can still inherit their privileges without any conflicts due to the newly specified timing constraints.

3.3.3 Authorization Decision Statement in the CAS credential

Once the roles and their privileges and constraints are specified in the CAS database as described above, a SAML authorization decision statement is included in the CAS credential. An example of the SAML authorization decision statement is shown in Figure 3.8, and its components, denoted by (1), (2), (3) and (4), are explained as follows:

(1) specifies the time period during which the authorization decision is valid.

(2) specifies the URI of the resource on which the permissions are granted.

(3) specifies the identity of the user to whom the permissions are granted.

(4) specifies what permissions are granted.

The user identified by the Subject in (3) is authorized in the role "Alpha,RN=Manager" with the privilege to execute the UPDATE operation on the resource specified in (2). Also, "Alpha,RN=Manager" inherits, from its junior role "Alpha,RN=Supervisor", the privilege to execute the SELECT operation on the same resource with the specified timing constraint "gmt#10.01.2005-07.30.2006#MON-FRI#19:00-5:00". The timing constraint specifies the duration for which the user can access the resource in the authorized role. This authorization decision is valid for the time period specified in (1).

```
<Assertion xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
AssertionID="8b53a37e-3116-44e2-a499-67e2d0fe49f1"
IssueInstant="2005-12-02T19:58:23Z"
Issuer="O=Grid,OU=GlobusTest,OU=simpleCA-
motive.cs.wright.edu,CN=Globus Simple CA" MajorVersion="1"
MinorVersion="0">

<Conditions NotBefore="2005-12-02T19:58:23Z" NotOnOrAfter=          (1)
"2005-12-02T21:20:53Z"></Conditions>

<AuthorizationDecisionStatement Decision="permit"                   (2)
Resource="http://130.108.17.176:8080/ogsa/services/ogsadai/SecureGrid
DataServiceFactory.Employee">

<Subject> /O=Grid/OU=GlobusTest/OU=simpleCA-                        (3)
motive.cs.wright.edu/OU=cs.wright.edu/CN=John  Doe
</Subject>
……..
<Action Namespace="role">Alpha,RN=Manager</Action>
<Action Namespace="Alpha,RN=Manager">privilege:
update</Action>
<Action Namespace ="Alpha,RN=Manager">junior_role:                  (4)
Alpha,RN=Supervisor</Action>

<Action Namespace="Alpha,RN=Supervisor">privilege:
select</Action>
<Action Namespace="Alpha,RN=Supervisor">timing_constraint:
gmt#10.01.2005-07.30.2006#MON-FRI#19:00-5:00</Action>

</AuthorizationDecisionStatement>
……..
```

Figure 3.8: SAML Authorization Decision Statement Issued by CAS

3.4 Implementation Details

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

memberships on roles to other users, and those users can exercise that right only within the user groups to which they belong.

CAS provides a set of APIs for managing fine-grain access policies for resources in a VO [17]. The Service API of CAS provides an administrative interface for managing the user groups and associated privileges. This API supports the user's role assignments in our method. CAS also provides a Client API through which users can obtain a signed SAML assertion and present it to the resource provider for authorization. The OGSA-DAI client program uses the Java Generic Security Services API (GSSAPI) to delegate the CAS credential to a Grid Data Service (GDS).

```
<?xml version="1.0" encoding="UTF-8" ?>

<!--  (c) International Business Machines Corporation, 2002 - 2004.  -->
<!--  (c) University of Edinburgh, 2002 - 2004.  -->
<!-- See OGSA-DAI-Licence.txt for licencing information.  -->
<DatabaseRoles>
    <Database name="jdbc:mysql://130.108.17.176/ogsadai">
      <User dn="No Certificate Provided" userid="ogsadai" password="ogsadai" />
      <User dn="/O=Grid/OU=GlobusTest/OU=simpleCA-
      motive.cs.wright.edu/OU=cs.wright.edu/CN=John Doe"
      userid="ogsadai" password="ogsadai" />

      <User dn="ER,RN=Physician" userid="username" password="password"/>

      <Role Name="ER,RN=Physician">
        <Action Namespace="role">privilege:select</Action>
        <Action Namespace="role">privilege:update</Action>
        <Action Namespace="role">timing_constraint:GMT#10.01.2005-
        07.30.2006#MON-FRI#19:00-5:00</Action>
      </Role>
  </Database>
</DatabaseRoles>
```

Figure 3.9: Modified Role-map File

We configured CAS to incorporate the proposed RBAC method as described before and modified the OGSA-DAI implementation to make use of the CAS credentials. The

modifications are made at both client-side and server-side. The client is modified to delegate the CAS credential instead of the user proxy credential. The server is modified to recognize the CAS credential delegated by the client and to obtain the VO role from it using the GSSAPI libraries. The modified server also verifies the privileges and constraints associated with the VO role against the local policy, and performs the mapping based on that role via the role-map file. The role-map file has been extended to include the mapping from a VO role to a database username and a password. Also included in it are the local policy details as shown in Figure 3.9. The role-map file can also include a list of users for whom access would be denied based on their Grid identity.

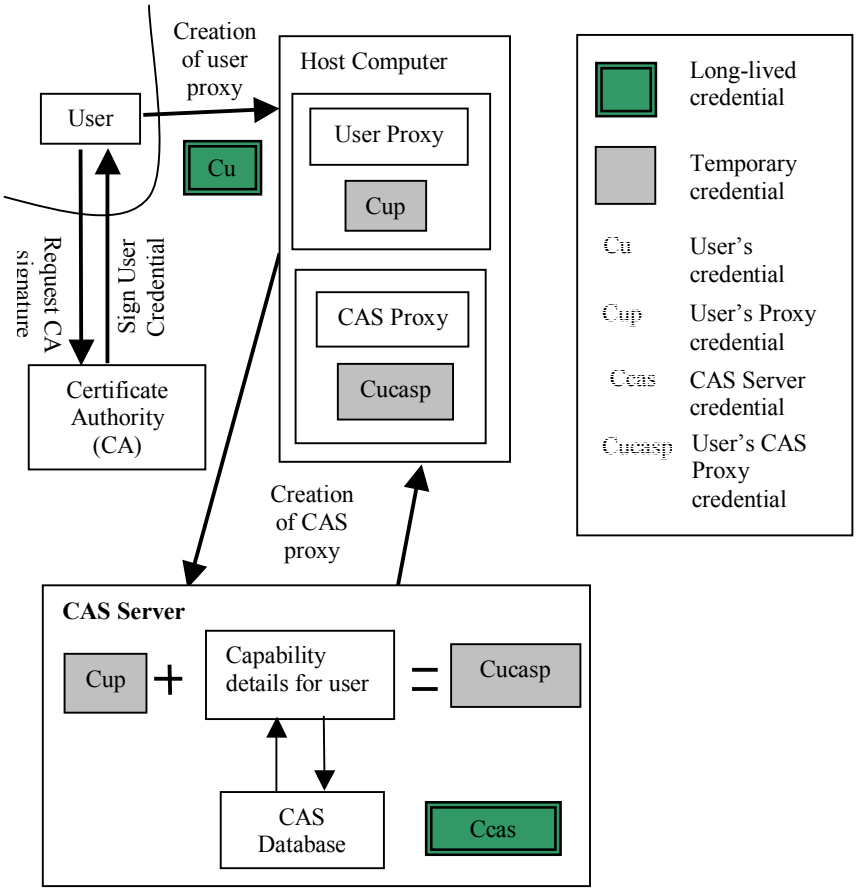


Figure 3.10: User's Normal Proxy Credential and CAS Proxy Credential Creation

The following is the sequential process of a Grid user obtaining a CAS credential for accessing a resource. As shown in Figure 3.10, a user generates a certificate (Cu) by making a request to a Certificate Authority (CA) which is trusted by all the entities within the Grid, i.e., all users and resources. If a user needs to gain access to a resource, the user generates a proxy credential (Cup) which is signed by his/her user certificate (Cu). This generated proxy credential's lifetime will be less than the lifetime of the user certificate. The lifetime of a proxy credential generated using the Globus Toolkit is 12 hours. In order to use a CAS credential, the user makes a request to the CAS server to initiate a CAS proxy based on the user's proxy credential. The CAS server authenticates the user and obtains the user's capability details present in the CAS database. The CAS server then creates a CAS proxy credential (Cucasp) which contains the CAS policy assertions to represent the user's capabilities and restrictions as an extension to the existing user proxy credential (Cup).

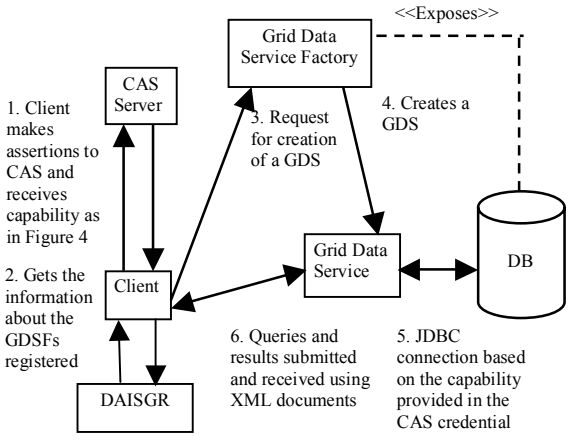


Figure 3.11: Accessing a Data Resource through OGSA-DAI Using a CAS Credential

As shown in Figure 3.11, once the user has obtained the CAS credential with the requested assertions, the user can contact the desired GDSF to create a GDS. The GDS gets the CAS credential delegated by the user, and verifies the capabilities against its local policy present in the role-map file. The GDS also checks if any specified timing constraint is violated.

```
#Initiate a User Proxy
% grid-proxy-init
Your identity: /O=Grid/OU=GlobusTest/OU=simpleCA-
motive.cs.wright.edu/OU=cs.wright.edu/CN=John Doe
Enter GRID pass phrase for this identity:
Creating proxy... Done
Your proxy is valid until: Fri Dec 2 21:20:53 2005

#Initiate a CAS Proxy
%cas-proxy-init -c
http://localhost:8080/ogsa/services/base/cas/CASService -t tag

#Contacting a specific GDSF using CAS capabilities
%java uk.org.ogsadai.client.Client -mls -role ER,RN=Physician -t tag -factory
http://130.108.17.176:8080/ogsa/services/ogsadai/SecureGridDataService
Factory examples/GDSPerform/JDBC/query/select1Row.xml
```

Figure 3.12: User Session Accessing a GDS Using CAS

Figure 3.12 depicts a typical user session using the command-line tools provided by the Globus Toolkit, CAS and OGSA-DAI, which shows the initiations of the user proxy and the CAS proxy. We have modified the OGSA-DAI client to accept the CAS credential and the desired VO role specified as shown in Figure 3.12. Based on the VO role of the user, a JDBC connection is established between the GDS and the database exposed by the GDSF. If no role is provided in the CAS credential, then the user's identity is used for mapping. The client can submit queries to the GDS and obtain the results in XML documents.

In our implementation, the GDS checks the local policy in the role-map file against the policy assertion in the CAS credential only before connecting the client to the database. After a CAS credential is issued, if a set of privileges is deleted from a VO role on the CAS server and/or the timing constraints on the role are changed then the credentials have to be expired before the new policy takes effect on the resources. We have not implemented mechanisms to revoke current credentials containing old policy assertions. However, if the same set of privileges deleted from the VO role is also deleted from each of the corresponding local roles, or the same changes to the timing constraints are made, then the access to the resources can be restricted immediately based on the new policy. The resource providers can use the local database management systems (DBMSs) to update the privileges on the local database roles and modify the triggers to accommodate the new timing constraints. The changes in the local policy information also have to be made in the role-map files. Since the privileges and timing constraints on the local database roles are enforced by the local DBMS itself, they will come into effect immediately. If the client submits a query, but the query fails due to the new policy, then the client can be notified via an error message. The client can request new credentials and then restart the application.

If a VO role is updated independently of the corresponding local roles after the credentials are issued, one way to restrict the access immediately is to have the CAS server notify the GDSFs of the updates. This information can be passed on to the GDSs and cached by the GDSFs for up to the maximum lifetime of the credential. Any credential issued before the notification and containing an authorized VO role, which has

been updated on the CAS server, can then be rejected. If a connection to the resource has been already established based on that role, then it should be discontinued.

If the local policy is changed to deny the access of a particular Grid user after that user has already been connected to the resource, then this new policy is not enforced because the role-map file is not rechecked. One possible solution to enforce the new local policy immediately is to notify the GDS every time the local policy information in the role-map file is updated, so that the user identity and policy assertion in the CAS credential can be checked against the local policy information.

With our method, a user who wants to perform the tasks associated with multiple roles does not need to generate multiple CAS proxies. The user can just delegate a single CAS credential containing all those roles. For example, a user may want to read from one database in one role and write to another database in another role. In this case, a single CAS credential containing both roles can be delegated, and then the user can be authorized by each resource provider with respect to the corresponding role.

3.5 Performance Analysis

The existing implementation of the OGSA-DAI client has been modified to delegate a CAS credential, and the server has been modified to obtain the user's capabilities present in the CAS credential. The overheads incurred with our implementation are compared with those of the existing implementation of OGSA-DAI,

which does not use the CAS credential. OGSA-DAI Release 4.0 was deployed on a Jakarta Tomcat 5.0.27/Globus Toolkit 3.2.1 (GT3) stack running on a Linux machine with a 2.6 GHz Intel Pentium IV processor and 1 GB of RAM. The *littleblackbook* MySQL database table distributed with OGSA-DAI was used as a test database, and it contains 10,000 tuples. The *perform* document consisting of a request for a single tuple was used for the purpose of analysis.

3.5.1 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, which logs time to a log file in milliseconds, is used. For more accuracy, the tomcat container was shutdown and restarted before each client request in order to minimize the caching effects within GT3 and OGSA-DAI [49]. The main changes from the original configuration are the way the mapping is done at the server-side and how the credential is delegated at the client-side. So, only the security aspects of the client and the server are profiled and analyzed. The following types of Grid Data Services are used in the analysis as in [49]:

1) *Signature*: GDS enforcing GSI Secure Conversation with Signature. This enforces message integrity being established between the client and the server.

2) *Encryption*: GDS enforcing GSI Secure Conversation with Encryption. This enforces message privacy being established.

3) *None*: GDS which does not enforce any security. The GDS does not provide a secure conversation.

3.5.2 Client-Side Security

A call is made to each of the above GDSs with and without using a CAS proxy credential. In case of using a CAS proxy credential, an additional overhead for its creation is incurred. In the performance analysis, we do not show this overhead because it is incurred only once before the client contacts the GDSs. Thereafter, the client can submit any number of queries before the CAS proxy credential expires. The lifetime of the CAS proxy credential is equal to the time remaining for the expiration of the user proxy credential, which can last up to 12 hours. The time taken for the creation of the CAS proxy credential depends on several factors such as network bandwidth and workload of the CAS server. In our system, the average time taken for the creation of a CAS proxy credential is around 600 milliseconds.

The *findServiceData* method of a GDSF returns the information about its corresponding data resource. Three consecutive calls to *findServiceData* are required: The first call returns the database schema, the second returns the activities permitted, and the third returns the product type (for example, the type of DBMS). The *perform* method of a GDS takes the *perform* document, which contains the query, and returns the results to the client. GSI Secure Conversation requires a security context to be established between the client and the server. The overheads incurred in setting up this security context are analyzed based on the following:

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

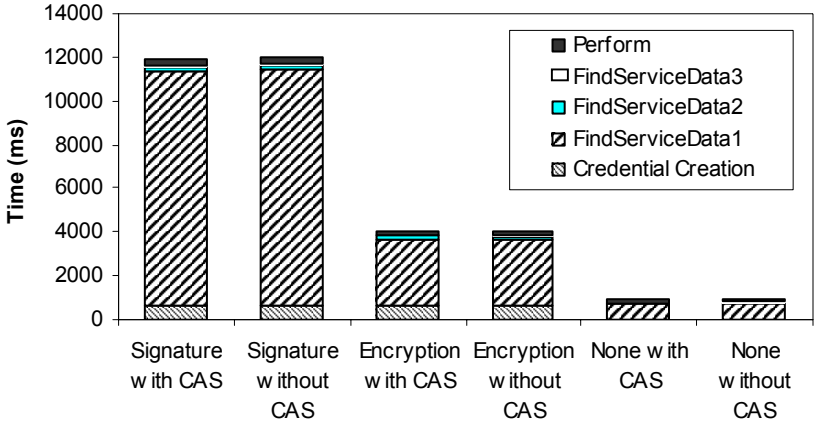


Figure 3.13: Client-Side Security

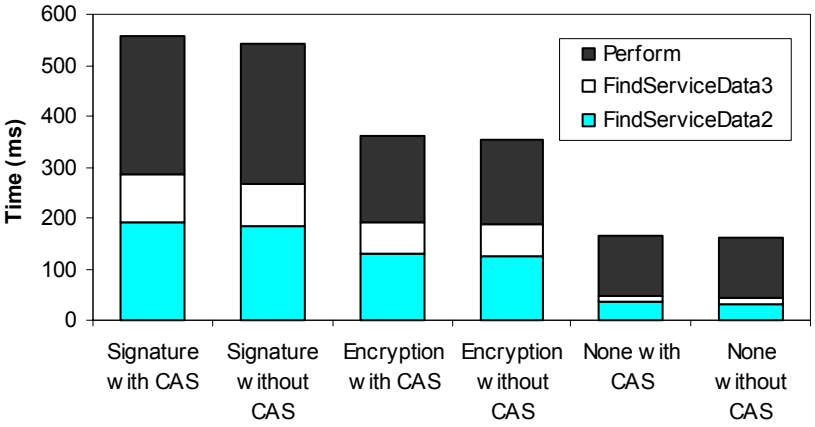


Figure 3.14: Obtaining Service Data and Query Execution

The corresponding times are shown in Figure 3.13, and as observed, the time for creating the credential object is almost the same regardless of the security enforced by the GDS. In case of *None*, there is no such overhead as the credentials are not used. The first call to the *findServiceData* takes longer than the subsequent calls because it includes the initialization of the GDS regardless of the security type used. Figure 3.14 clearly shows

the times taken for the subsequent calls to the *findServiceData* and the *perform* methods. The times recorded in the case of using a CAS proxy credential and those without using a CAS proxy credential are almost the same. The reason is because all the security functions on the client-side remain unchanged except for the use of a CAS proxy credential instead of a user proxy credential.

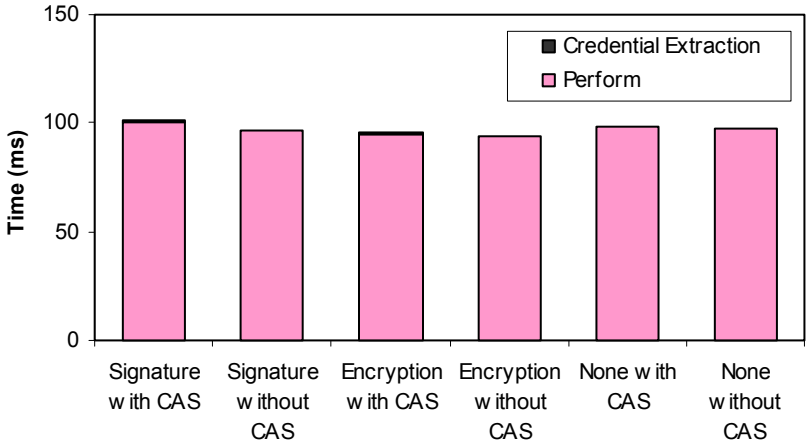


Figure 3.15: Server-Side Security

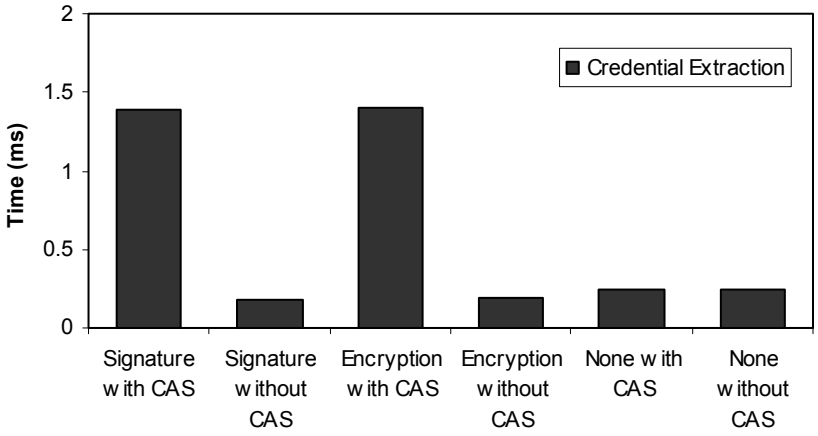


Figure 3.16: Security Overheads on the Server-Side

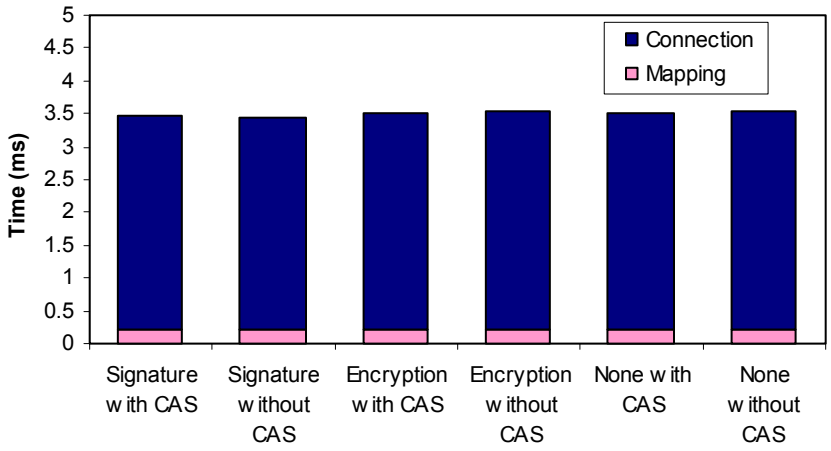


Figure 3.17: Mapping and Database Connection

3.5.3 Server-Side Security

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

1. The client credentials accessed using the GT3 infrastructure.

2. Extracting the VO role or Grid identity from the credential. If the VO role is extracted, its capabilities are compared against the local policy.

3. Mapping a user to a database username and a password, and creating a JDBC connection.

4. The *perform* operation.

As shown in Figure 3.15, the time for the credential extraction, which includes policy comparison, is very small compared to the time for executing the *perform* operation. The time for executing the *perform* operation remains constant for all the GDSs. The *perform* operation is done only after the credential extraction process is completed; and as a result, its execution time is not affected by the type of credential

used. The credential extraction times are shown more clearly in Figure 3.16, and we can see that the credential extraction takes more time when a CAS proxy credential is used for contacting a GDS that enforces the secure conversation. An overhead is incurred because of the time taken for obtaining the user identity and the policy assertion from the CAS credential and then comparing it against the local policy in the role-map file. However, this overhead is in the order of a few milliseconds and is insignificant compared to the overall time taken for performing the client's query. When a CAS proxy credential is not used, the user proxy credential is used instead, and then the credential extraction involves obtaining only the user identity. In case of contacting a nonsecure GDS, since credentials are not used, there are no overheads incurred for credential extraction.

Figure 3.17 shows that there is a constant overhead for mapping a user to a database username and a password and then subsequently setting up the database connection. The processes of mapping and connection are done after the credential extraction process is completed; and as a result, their execution times are not affected by the type of credential used. If there are a large number of entries in the role-map file, the mapping would still not take much time because a hash table is used to store those entries.

In summary, on the client-side, our method incurs small overheads in the security setup as additional steps are involved for requesting and using the CAS credential. However, as seen from the performance results, the time taken for the individual OGSA-

DAI method calls are the same whether a CAS credential is used for authorization or not. This is because all the security functions remain unchanged, except for the use of a CAS proxy credential instead of a user proxy credential. On the server-side, the additional overheads incurred in our credential extraction process are very small compared to the time taken for executing the client's queries. These overheads in setting up the security context are insignificant when we consider the benefits of our method, such as scalability in managing VO policies and reduced administration overheads for resource providers.

Chapter 4

RBAC with Shibboleth in the OGSA-DAI System

In the last chapter, we described how the Community Authorization Service (CAS) [14] can be used to enhance the security mechanism in OGSA-DAI. 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 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. In this chapter, we propose for OGSA-DAI an RBAC system using Shibboleth, GridShib and the Object, Metadata and Artifacts Registry (OMAR) [50]. OGSA-DAI has recently 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, which can be any entity that acts as a source and/or a sink of data [13]. Shibboleth is designed to provide user attributes to the resources for access control, and it mainly targets the internet-based resources. In our system, it is also used as a Role Enablement Authority (REA), which is responsible for assigning roles to users and for enabling roles within a user's session [17]. OMAR provides an implementation of the OASIS e-business eXtensible Markup

Language (ebXML) registry specifications. The ebXML registry specifications are developed to achieve interoperable registries and repositories with an interface that enables submission, query and retrieval [51].

Our system is scalable in terms of the number of access requests as well as the number of users and VOs; and it is robust as there is no single point of failure. It supports the management of roles and privileges; and also supports dynamic delegation of rights via roles. It also supports fine-grain attribute release policy and provides privacy protection for users within VOs that employ OGSA-DAI. Furthermore, similar to the previous system it can support a wide range of security policies using role-privileges, role hierarchies, delegations, and constraints. Resource providers need to maintain only the mapping information from VO roles to local roles and local policies, thus their administration overhead is reduced. When users join/leave a VO, the resource providers do not have to bother about individually adding/removing their information in the role-map files, because OMAR can be used to directly grant/revoke their memberships on the VO roles. Moreover, the resource providers can permit or deny the access requests of specific users by maintaining their authorization information separately. This enables the resource providers to have the ultimate authority over their resources. Also, unnecessary mapping and connections can be avoided by denying invalid requests at the VO level.

We have implemented our proposed system and analyzed its performance. The server-side of OGSA-DAI has been configured to use Shibboleth through GridShib. The GridShib software provides two interfaces, one for the Grid services and the other for

Shibboleth [33]. The OGSA-DAI server has been modified to obtain the user's attributes from the Shibboleth service, verify them against the local policies, and map the user to a local role based on the role-map file. The role-map file has been extended to include the mapping from a VO role to a local role. Our performance analysis shows that the proposed system incurs a small overhead in setting up the security context between the client and server. This overhead is quite acceptable when we consider the benefits of our system, such as scalability in managing VO policies and reduced administration overhead for resource providers.

The organization of the chapter is as follows: In Section 4.1, we propose a RBAC system with Shibboleth and GridShib in OGSA-DAI. In Section 4.2, we show how to manage VO policies using XACML and OMAR. Section 4.3 describes the results of performance analysis.

4.1 Architecture of the RBAC System Using Shibboleth

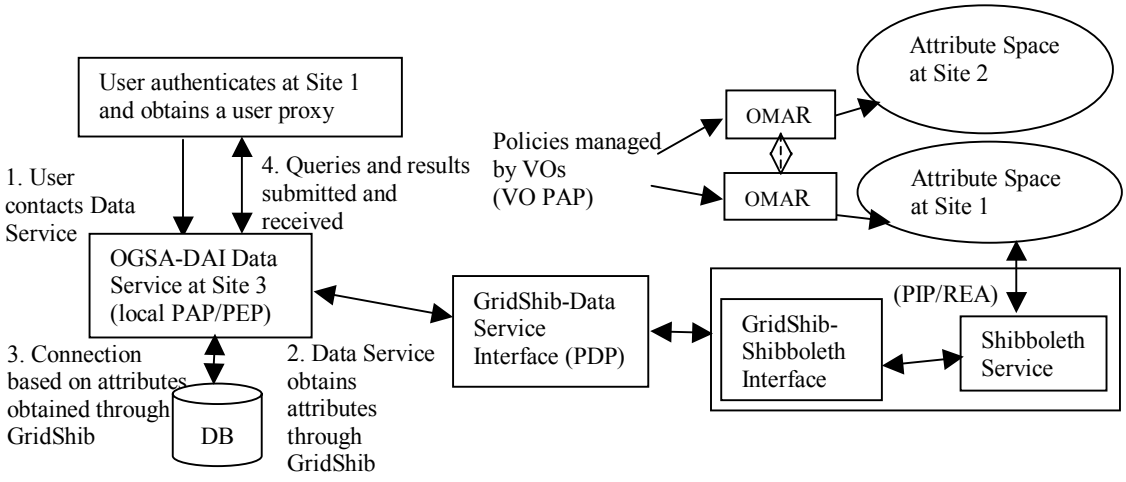


Figure 4.1: Accessing a data resource using GridShib and Shibboleth

Each site participating in the Grid maintains its own attribute space and Shibboleth service. The GridShib software provides two interfaces, one for the Grid services and the other for Shibboleth. Shibboleth and GridShib support the pull model in which a target resource (a) authenticates a user by using the Grid Security Infrastructure (GSI) of Globus Toolkit [14]; (b) determines the address of the appropriate Shibboleth service in the process; and (c) obtains the selected user attributes (that the resource is authorized to see) from the Shibboleth service [33].

In our system, as shown in Figure 4.1, the GridShib-Shibboleth interface and the Shibboleth service together function as the Policy Information Point (PIP) and Role-Enablement Authority (REA). A PIP releases attribute values related to the subject (such as a user, application or Grid service), the resource and the environment. A REA is responsible for assigning roles to users and for enabling roles within a user's session [17]. Attributes are released in order to authorize users not only based on their entitlements and affiliations, but also based on their requested roles, role memberships and user credentials. The user credential is formed by an X.509 certificate and the associated public/private keys, and is issued by a Certificate Authority (CA) trusted by all entities in a Grid [44].

The user submits a request to the OGSA-DAI Data Service, and it retrieves the user's roles and attributes from the PIP/REA (i.e., GridShib-Shibboleth interface and the Shibboleth service) based on the user's identity. The GridShib interface for the OGSA-DAI Data Service functions as the Policy Decision Point (PDP) and returns the

authorization decision, such as “permit” or “deny”, to the requesting Data Service. The request is evaluated based on the attributes released by the PIP/REA and the attributes of the local policies maintained by the Data Service. The OGSA-DAI Data Service functions as a Policy Administration Point (PAP) at the local level and also as a Policy Enforcement Point (PEP). A PAP manages the policies and policy sets, and makes them available to the PDP (i.e., the GridShib interface for the OGSA-DAI Data Service). A PEP executes the decision of the PDP by either performing or denying the client’s request. If the decision is “permit”, the PEP (OGSA-DAI Data Service) sets the security context based on the user’s role, and then a connection is established between the Data Service and the requested data resource. The user can then submit queries to the Data Service and obtain the results. If the decision is “deny”, then an error message is returned to the user, indicating that the user is not authorized to perform the operation. PIP, PDP, PAP, and PEP are terms used in the XACML authorization model [18].

The attribute space at each site, shown in Figure 4.1, is composed of the attributes related to the subject, the resource and the environment; and can also hold the attributes pertaining to VO policies. Shibboleth does not store or manage attributes, so a data store, such as a Lightweight Directory Access protocol (LDAP) directory or a database, is required. We propose the use of OMAR as the PAP at the VO level to administrate the portion of the attribute space pertaining to VO policies at the individual sites. This is explained in the following section.

4.2 Managing VO Policies Using XACML and OMAR

In order to specify VO policies in the form of VO roles, role hierarchies, privileges and constraints, we used the Core and Hierarchical RBAC profile of XACML. We also used OMAR for the storage and distributed administration of the VO policies. Specification of policies at the VO level allows authorization decisions to be made based on the user's request and VO policies. In case the user does not possess the required privileges, the access can be denied at the VO level. This eliminates mapping and connection overheads on the resource providers in case the request is not valid.

4.2.1 eXtensible Access Control Markup Language (XACML)

XACML is an OASIS standard for describing access control policies uniformly across different security domains [18]. XACML 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.

An XACML <PolicySet>, <Policy> and <Rule> may contain a <Target> element. A <Target> element specifies the set of subjects, resources, actions and environments to which the <PolicySet>, <Policy> and <Rule> applies [18]. The Core and Hierarchical RBAC profile of XACML specification defines how ANSI core and hierarchical RBAC standard [19] can be specified in XACML. The Core and Hierarchical profile further defines the following components:

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

4.2.2 Specifying VO Policies Using XACML

In this section, we explain how the VO policies can be expressed with the Core and Hierarchical RBAC profile of XACML. For example, consider a VO Alpha which could be a project undertaken by collaborating organizations. Assume that the VO uses two roles: manager and employee. An employee has permission to read the data from a database only from 9:00 AM to 5:00 PM. The manager has all the permissions of the employee, and additionally has permission to update the database. The manager role is therefore senior to the employee role. In multi-domain environments such as VOs, it is necessary to manage the attributes across different domains and is often needed to aggregate the attributes for making authorization decisions. For this purpose, it is necessary to distinguish the VO's attributes from local attributes and also from attributes

of other VOs. For example, the employee role in the Accounting subgroup of Alpha may have different permissions from the employee role in the Accounting subgroup of Beta, where Beta is another VO. Hence, when making an authorization decision, it is not only important to know the role name employee but also the VO name, either Alpha or Beta.

In order to manage and identify 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. Scope identifies the domains and sub-domains in which the values are defined. For example, a scoped attribute may be "faculty@abcuniv.edu", which identifies the value "faculty" in the scope "abcuniv.edu". However, the XACML profile does not support these scoped attribute values for subjects such as roles. Mapping SAML to XACML allows the systems using XACML to store SAML attributes [62].

```
<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" Scope="Alpha.Accounting">&roles;employee</AttributeValue>
              <SubjectAttributeDesignator AttributeId="&role;" DataType="&xml;anyURI"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
   </Target>
   <PolicySetIdReference>PPS:employee:role</PolicySetIdReference>
</PolicySet>
```

Figure 4.2: RPS of the employee role

We show how the scoped attribute values can be specified in the RBAC profile of XACML to represent role names specific to VOs and VO subgroups. For example, the

employee role in the Accounting subgroup of Alpha can be represented in XACML as shown in Figure 4.2 which illustrates the RPS for this role. The VO and subgroup names "Alpha.Accounting" represent the scope of the employee role. The RPS of the employee role references the PPS of the employee role via <PolicySetIdReference>. The PPS of the employee role is shown in Figure 4.3, where the resource is represented in a hierarchical form. 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 the PPS:manager:role shown in Figure 4.4.

In order to support fine-grain authorization for resources, where access control can be specified not only for the entire resource (e.g. database) but also for its components (e.g. table), the hierarchical resource profile of XACML [53] can be used. This profile specifies how XACML provides access control for resources that are organized as a hierarchy, such as file systems, XML documents, and databases. For example, for non-XML data, the profile specifies the URI of the following form: <scheme>://<authority>/<pathname> where <pathname> is of the form <root name> {/<node name>}, and <scheme> identifies the namespace of the URI and may further restrict the syntax and semantics of identifiers using that scheme. The scheme can be a protocol such as "ftp" or "http", and a file system resource can have "file" as the scheme. <authority> is typically defined by an Internet-based server or a scheme-specific registry of naming authorities, such as DNS. The sequence of <root name> and <node name> values should correspond to the components in a hierarchical resource. For example,

"https://localhost:8484/ogsadai/DataService/Employee" indicates the Employee table in the database represented by the specified URI.

```
<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:employee:table" Effect="Permit">
   <Target>
    <Resources>
     <Resource>
      <ResourceMatch MatchId="&function;string-equal">
        <AttributeValue DataType="&xml;string">https://localhost:8484/ogsadai/DataService/Employee</AttributeValue>
        <ResourceAttributeDesignator AttributeId="&resource;resource-id" DataType="&xml;string"/>
      </ResourceMatch>
     </Resource>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch MatchId="&function;string-equal">
          <AttributeValue DataType="&xml;string">select</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>
</PolicySet>
```

Figure 4.3: PPS of the employee role

The PPS of the employee role shown in Figure 4.3 grants the permission to execute the SELECT operation (specified within <Action>) on the resource identified by the URI "https://localhost:8484/ogsadai/DataService/Employee" (specified within <Resource>) only from 9:00 AM to 5:00 PM (specified within <Condition>). Obviously, not only the

basic database operations, but also complex operations, such as transactions and stored procedures, can be permitted. The PPS of the manager role shown in Figure 4.4 grants the permission to execute the UPDATE operation on the Employee table. It references the PPS of the employee role via <PolicySetIdReference>, thereby inherits all the permissions of the 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:update:data:from:employee:table" Effect="Permit">
    <Target>
     <Resources>
      <Resource>
       <ResourceMatch MatchId="&function;string-equal">
        <AttributeValue DataType="&xml;string">https://localhost:8484/ogsadai/DataService/Employee</AttributeValue>
        <ResourceAttributeDesignator AttributeId="&resource;resource-id" DataType="&xml;string"/>
       </ResourceMatch>
      </Resource>
     </Resources>
    <Actions>
      <Action>
           <ActionMatch MatchId="&function;string-equal">
                    <AttributeValue DataType="&xml;string">update</AttributeValue>
                    <ActionAttributeDesignator AttributeId="&action;action-id" DataType="&xml;string"/>
           </ActionMatch>
      </Action>
     <Actions>
    </Target>
  </Rule>
 </Policy>
  <PolicySetIdReference>PPS:employee:role</PolicySetIdReference>
</PolicySet>
```

Figure 4.4: PPS of the manager role

4.2.3 Object, Metadata and Artifacts Registry (OMAR)

For storing and managing the XACML policies, we make use of the Object, Metadata and Artifacts Registry (OMAR) 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 [51]. 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 the sharing of its content and metadata between organizational entities in a federated environment [54].

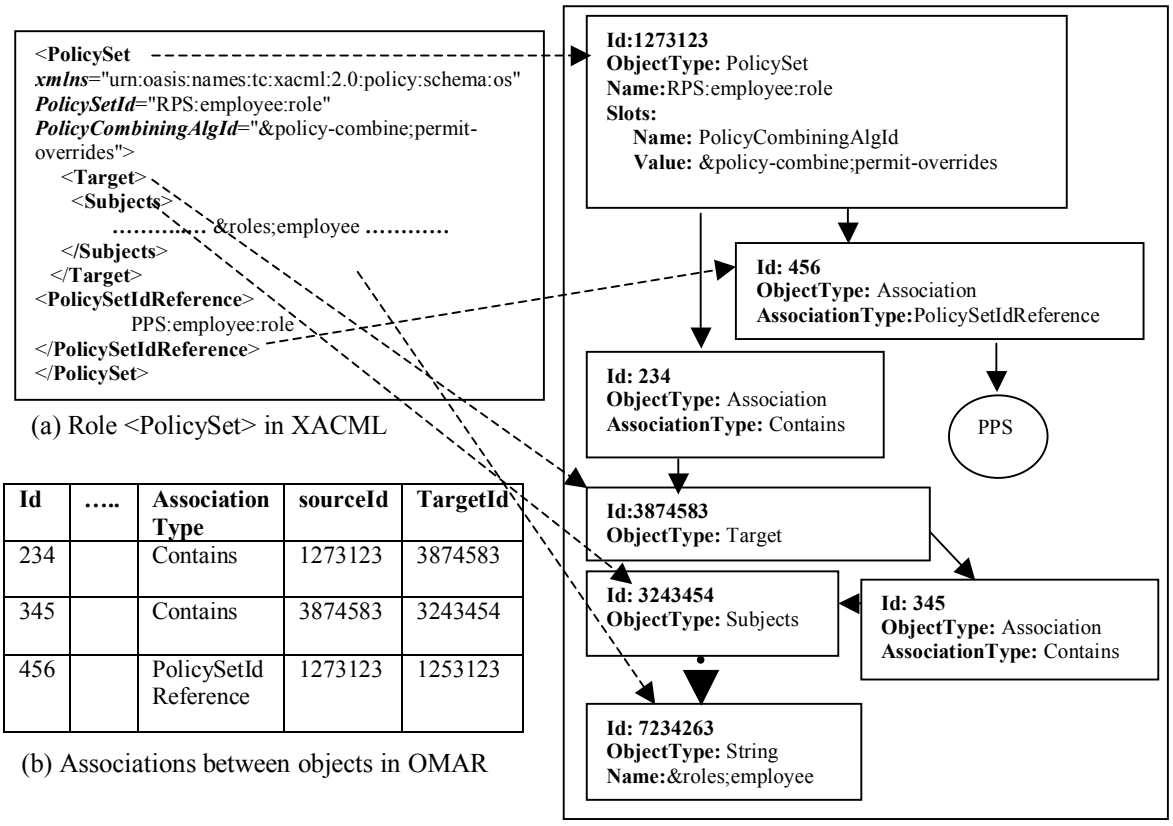
OMAR stores data in a repository and stores the associated metadata as registry objects [54]. The relationship between the registry objects is represented by an association object. OMAR allows many-to-many associations between the registry objects. OMAR uses an object, called slot, to add attributes dynamically to registry objects.

OMAR stores the XACML policies in their entirety as repository items 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 described below.

4.2.4 Managing the RBAC Policies in XACML Using OMAR

OMAR allows the creation of new objects and their classification. The information pertaining to these objects is stored in a relational database. A Role <PolicySet> (RPS) is represented in XACML as shown in Figure 4.5(a). In OMAR, we represent a

<PolicySet> as an XACML PolicySet object with the <PolicySetId> attribute value used as the object's name. The other attributes of the <PolicySet> are represented as slots. For the other components, such as <Target> and <Subjects>, we create new objects, as shown in Figure 4.5(c).



(a) Role <PolicySet> in XACML

Id	…..	Association Type	sourceId	TargetId
234		Contains	1273123	3874583
345		Contains	3874583	3243454
456		PolicySetId Reference	1273123	1253123

(b) Associations between objects in OMAR

(c) Role <PolicySet> stored as objects in OMAR

Figure 4.5: A part of the employee RPS in XACML and corresponding storage in OMAR

The role name is stored as a String object. To represent the relationship between the various components, we use the associations, such as 'Contains', which are defined by OMAR. In addition, we create a new association 'PolicySetIdReference' to represent the association between a Role <PolicySet> and a Permission <PolicySet> (PPS). The

associations between the various objects are captured in a relational table as shown in Figure 4.5(b). For example, the association between a PolicySet object and a Target object is stored as a tuple with Id '234'. A representation of a PPS in XACML and it corresponding storage in OMAR are shown in Figure 4.6.

```
<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:action2:on:resource2" Effect="Permit">
   <Target>
     <Resources>….resource2… </Resources>
     <Actions>…action2… </Actions>
    </Target>
   </Rule>
  </Policy>
</PolicySet>
```

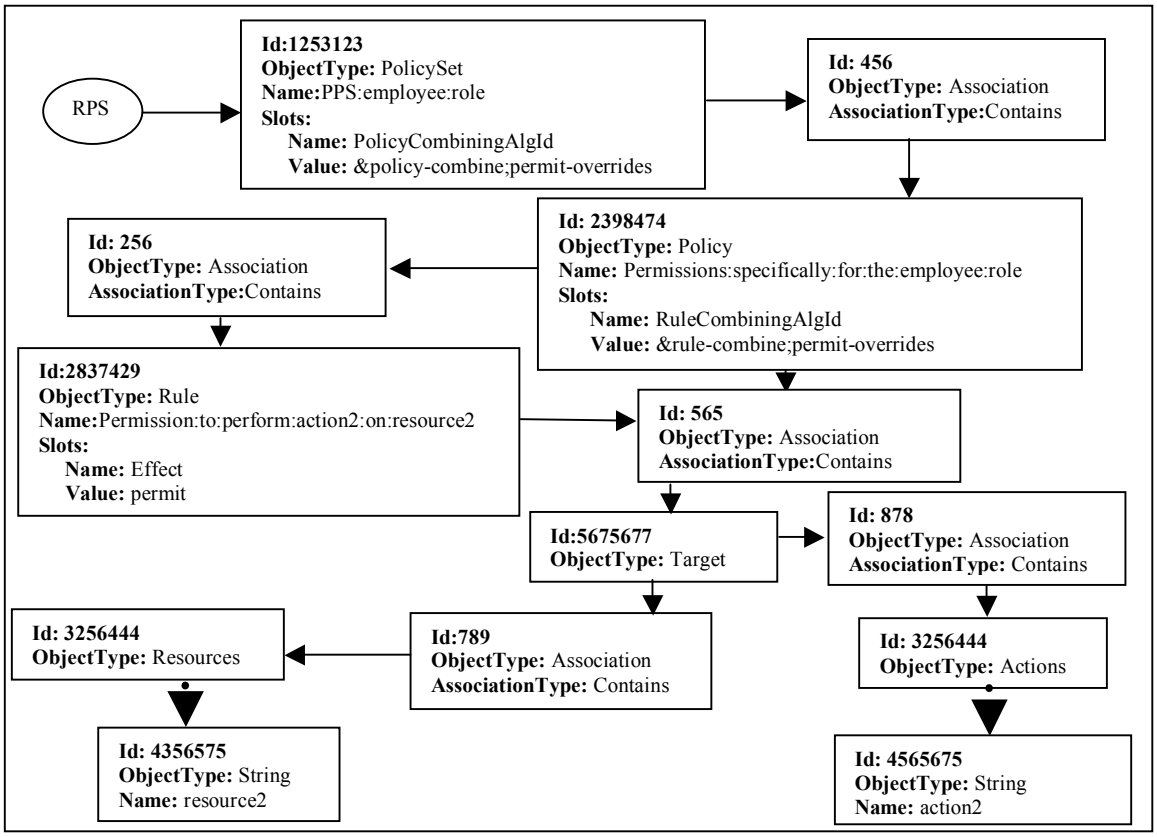


Figure 4.6: A part of the employee PPS in XACML and corresponding storage in OMAR

OMAR supports the federation of registries by defining a federation object and the 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 Grid, while each site maintains its own registry and Shibboleth service. Hence, there is less potential for bottlenecks and no single point of failure in the system as authorization queries are distributed among the Shibboleth services. OMAR provides a Java browser user interface and a Web user interface for managing the registry objects and repository items. It also provides an API for creating new objects and associations. The XACML-based policies can be easily managed through this API since all the policy information is stored in relational databases. Figure 4.7 shows the policy objects in Figures 4.5 and 4.6 in a Java browser, and Figure 4.8 shows how the attributes and their values of the PolicySet object in Figure 4.6 are stored using slots in OMAR.

4.2.5 User-Role Assignments

The RBAC profile of XACML allows the specification of constraints on specific user-role assignments through the Role Assignment <Policy>/<PolicySet>, but it does not maintain the roles assigned to each user. The user-role assignments are typically made by an identity provider at each site by adding the roles to a user's attribute list. The identity provider can release the attributes of the users to the resource providers so that access control decisions can be made.

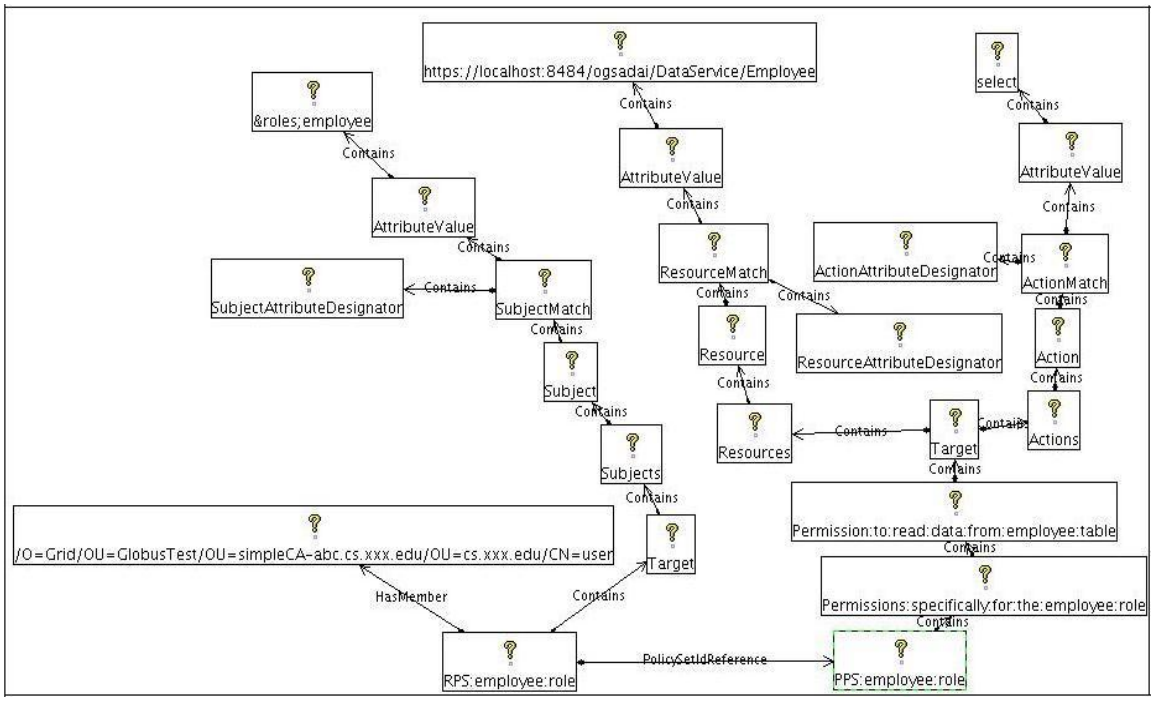


Figure 4.7: Policy objects of the employee role and their associations stored in OMAR

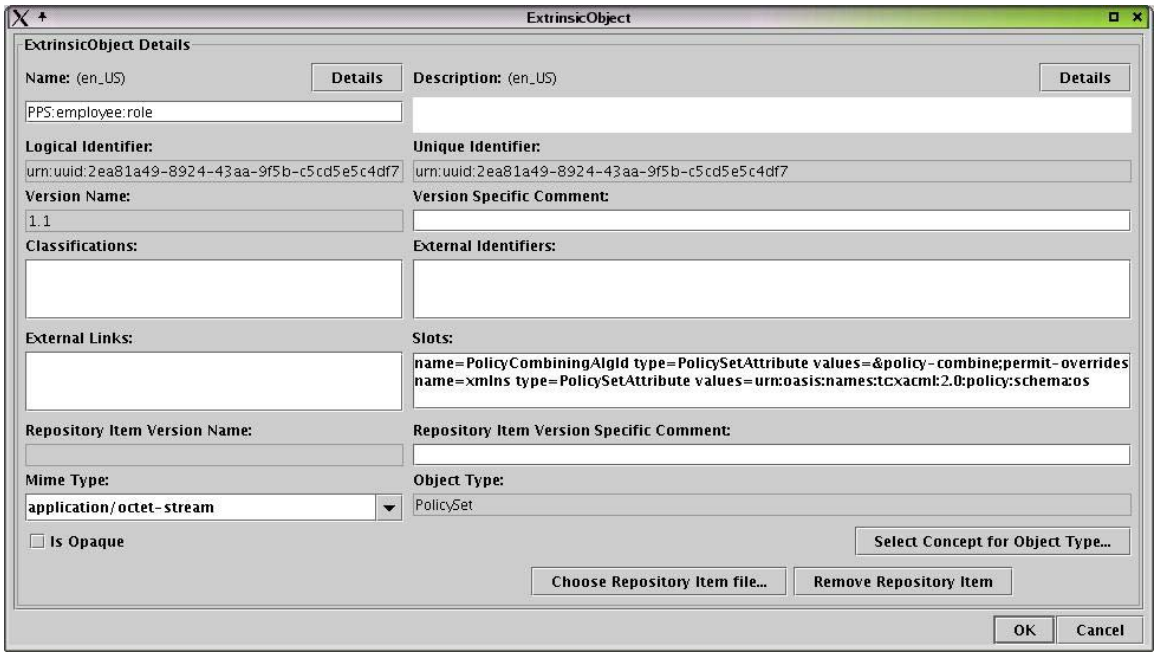


Figure 4.8: Details of the PPS:employee:role object in Figure 4.6 as stored in OMAR

In a VO, the user attributes may have to be managed across multiple sites, and in that case a portion of the attribute space at each of those sites should be delegated to the VO. This can be supported by Signet [55] which is a privilege management system being developed by a working group of the Internet2 Middleware Initiative for the distributed administration of privileges. By interfacing Signet to the Shibboleth and enabling the user access through GridShib, we can delegate the management of a portion of an attribute space at a site [33].

The VO can then make the role assignments at each site for the users who are VO members. However, the Grid identity of the user has to be mapped to a local identity in order to retrieve the attributes of the user from the attribute space of the site. The GridShib interface for Shibboleth handles this mapping by maintaining a text file that maps the Grid identity of a user to a local identity. However, this method is not scalable as the mapping information should be maintained for each user. In our system, roles are assigned to the user's Grid identity, and it can be done within OMAR itself. A registry object is created for each user, and it contains the user's Grid identity. The "HasMember" association is specified between the RPS object of the role and the user object as shown in Figure 4.7, thus allowing the REA in our system to enable the user's role.

Moreover, since Shibboleth can be interfaced to local data stores, like LDAP directories and databases, providing user attributes, the VO can delegate the user-role assignments to individual sites. The sites can assign certain VO roles to local users based on their attributes. For example, only the employees of a particular department can be

given a membership on the role VO-guest. The REA in our system can also make user-role assignments dynamically. For example, students enrolled in a particular course can be given a membership on the role VO-student, based on their user attributes in the local data stores and the Role Assignment <Policy>/<PolicySet> registry objects.

4.2.6 Administration of RBAC Policies and Dynamic Delegation of Rights with OMAR

One of the key features of RBAC is the ability to manage itself through administrative roles and permissions [10]. Users in administrative roles can create roles and role hierarchies; make user-role and permission-role assignments; and specify constraints. Furthermore, they can assign 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.

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 privileges for creating/accessing certain registry objects to that role. A registry role is an OMAR component similar to a database role. 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 to create 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.

The RBAC profile of XACML does not address dynamic delegation of rights, which is important in Grids. A user should be able to dynamically delegate his/her rights to other users, applications and Grid services without administrative intervention. For example, a VO supervisor with access rights to a new resource may wish to delegate his/her rights to applications run by certain users. The Grid Security Infrastructure (GSI) of the Globus Toolkit achieves such delegation through the use of temporary proxy credentials which are generated based on user credentials [44]. However, it does not allow the specification of constraints on the delegation of rights. For example, a VO supervisor may be allowed to delegate his/her rights only to programmers, but not to operators. Such constrained dynamic delegation can be easily achieved with OMAR. The VO supervisor can be assigned to a registry role that allows him/her to only create policy objects representing a new VO role, but not to create user objects. The VO supervisor can also be allowed to create associations only between the PPS object of the new VO role and the PPS object of the VO supervisor role. This allows the delegation of privileges from the supervisor role to the new VO role. Furthermore, the VO supervisor can be allowed to create associations only between the RPS of the new VO role and the user objects associated with the RPS of the VO programmer role. This allows only VO programmers to be assigned to the new VO role. Thus, the VO supervisor can delegate its

own privileges only to VO programmers, whose applications can then access the new resource as soon as the new VO role is enabled by the REA.

4.3 Performance Analysis

We have implemented the proposed RBAC system and integrated it with OGSA-DAI. We have configured the server-side of OGSA-DAI to use Shibboleth through GridShib. The OGSA-DAI server has been modified to obtain the user's attributes from the Shibboleth service, verify them against the local policy, and map the user's VO role to a local role by using the role-map file. The role-map file has been extended to include the mapping from a VO role to a local role. Furthermore, the resource providers can grant or refuse the access requests of specific users by maintaining their authorization information separately in the Access Control List (ACL) maintained by the OGSA-DAI server. This enables the resource providers to have the ultimate authority over their resources. The Shibboleth service has been modified to use a custom data connector for the retrieval of the user's attributes based on his/her Grid identity from OMAR.

The overheads incurred with our RBAC system are compared with those of the existing authorization infrastructure of OGSA-DAI. In our system, OGSA-DAI WSRF Release 2.0 was deployed in the Globus Toolkit 4.0.1 container running on a Linux machine with a 2.6 GHz Intel Pentium IV processor and 1 GB of RAM. Shibboleth identity provider (IdP) 1.3c was configured to run on 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. The

littleblackbook MySQL database table distributed with OGSA-DAI was used as a test database, and it contains 10,000 tuples. The *perform* document consisting of a request for a single tuple was used for the purpose of analysis.

4.3.1 Profiling Details

A Java method *System.currentTimeMillis*() is used to get the current system time in milliseconds. Also, for the server-side analysis, the Apache Log4j logger, which logs time to a log file in milliseconds, is used. For more accuracy, the tomcat, apache and Globus toolkit containers are shutdown and restarted before each client request, in order to minimize the caching effects within Globus Toolkit, OGSA-DAI and Shibboleth server. 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 Grid Security Infrastructure (GSI) which allows two levels of security: transport-level and message-level. In the transport-level security, the complete communication channel between the client and server is encrypted. In the message-level security, only the message is encrypted, so it has the 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. The performance is analyzed based on these different levels of security established between the client and the server as shown

below. For each of these levels, we recorded the times taken with and without using Shibboleth.

1) *MLS*: Enforces GSI Secure Conversation between the client and the server of OGSA-DAI with message-level security and privacy.

2) *TLS*: Enforces GSI Secure Conversation between the client and the server of OGSA-DAI with transport-level security and privacy.

3) *None*: Does not provide a secure conversation between the client and the server of OGSA-DAI and does not enforce any security.

4.3.2 Client-Side Security

A call is made to each of the above OGSA-DAI Data Services with and without using Shibboleth. In case of using Shibboleth, additional overhead exists for contacting the Shibboleth service, retrieving user attributes, and verifying the validity of the attribute assertions. The *getversion* method of the Data Service returns the version of OGSA-DAI used. The *listResources* 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 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 made to manage the communication with the configurable Data Service.

2. Calls to the *getversion, listResources* and *perform* methods.

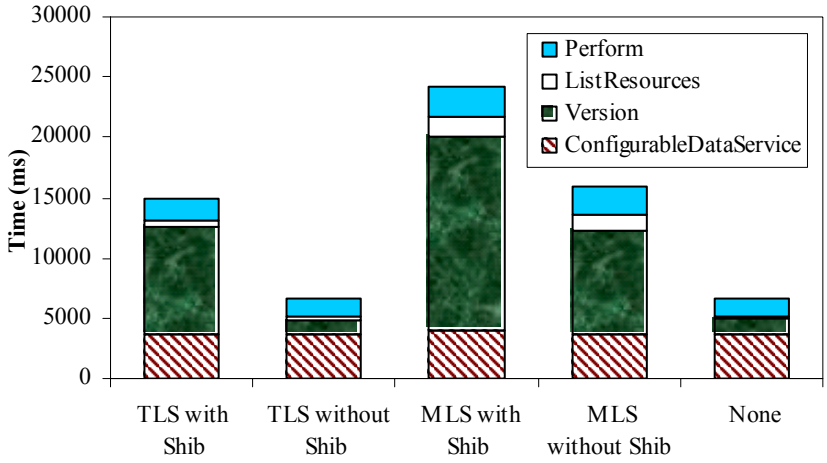


Figure 4.9: Client-Side Security

The corresponding times are shown in Figure 4.9, and as observed, the time for creating the proxy which manages the communication with the configurable data service is almost the same. This process involves connecting to the service at the given URL, retrieving its information, and determining the specification of web services used for the OGSA-DAI distribution. As this step 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, listResources* and *perform* take longer when Shibboleth is used because the server has to contact it, retrieve the user information, and verify whether the user is authorized to perform the requested operation. In case of *None,* it takes less time because there is no such authorization overhead.

The times for executing the *getversion* method are longer compared with other methods, because it is the first call made by the client and takes time to retrieve the user attributes from the Shibboleth service. The later calls take less time as they may require the same attributes that are obtained previously and cached by the GridShib interface for the Data Service.

4.3.3 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*, *listResources* and *perform* method calls.
2. Time to establish the JDBC connection.
3. A call to the *getSecurityContext* method which returns the security related information.

Figure 4.10 shows the times taken for retrieving the attribute information during the *getVersion*, *listResources* and *perform* method calls. The GridShib interface for the Data Service can cache received attributes from call to call, which allows multiple methods to be executed by the same client without making repeated callouts to the attribute authority. Hence, the attribute retrieval from Shibboleth during the *getVersion* method call involves contacting the attribute authority and takes longer time than subsequent method calls which can use the cached user attributes. When Shibboleth is not used, this overhead is not incurred as attribute retrieval is not performed.

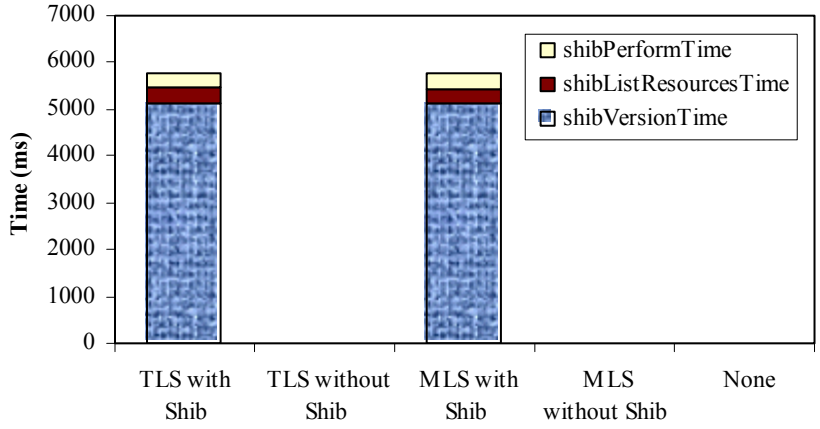


Figure 4.10: Server-Side Security

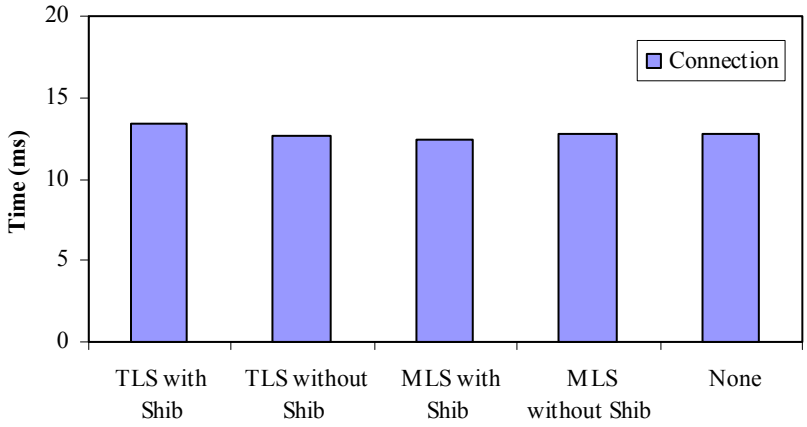


Figure 4.11: Database Connection

Figure 4.11 shows the time taken for setting up the database connection between the server and the resource. As no authorization is required, this step takes almost the same time regardless of the security enforced by the Data Service. Figure 4.12 shows the time taken for obtaining the security related information on the server-side. In this step, obtaining the user credential and extracting the identity of the user can be done only when the GSI Secure Conversation is used. In case of *None*, the overhead is bigger

because of the unsuccessful attempts made to retrieve the credential and the identity of the user.

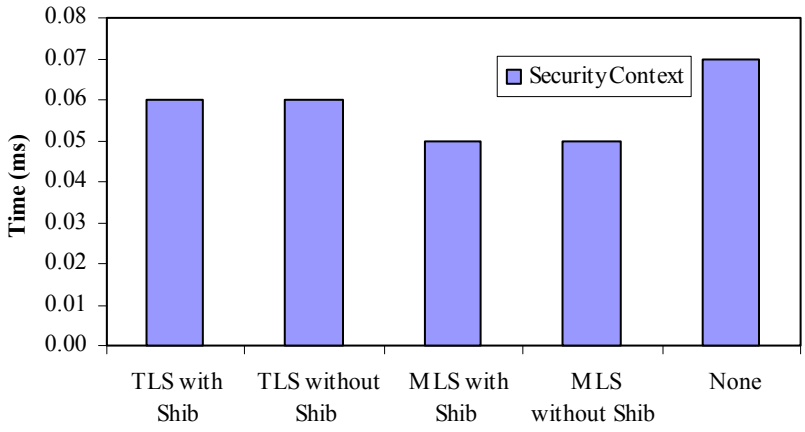


Figure 4.12: Retrieval of Security Context

In summary, on the client-side and server-side, additional overheads exist when setting up the security context because the server contacts the Shibboleth service, retrieves user attributes, and verifies the validity of the attribute assertions in order to authorize the client. The client can execute multiple methods without repeated callouts being made to the attribute authority by the server. This is because the GridShib interface for the Data Service can cache received attributes and those same attributes may be required for subsequent method calls. On the server-side, no additional overheads are incurred in the credential extraction process and setting up of the database connection, as no authorization is required during these steps. The overheads in setting up the security context are quite acceptable when we consider the benefits of our method, such as scalability in managing VO policies and reduced administration overheads for resource

providers. For multiple large queries, this overhead is almost negligible compared to the execution time of the *perform* document.

Chapter 5
Conclusions

In this research, we enhanced the role-based access control (RBAC) mechanism of OGSA-DAI by using (1) the Community Authorization Service (CAS), and (2) the Shibboleth, GridShib, XACML and OMAR, so that users are granted memberships statically and dynamically on virtual organization (VO) roles for Grid database services. Interoperability between different security mechanisms, dynamic delegation of rights, privacy protection, and the centralized and distributed management of privileges are supported. The resource providers need to maintain only the mapping information from VO roles to local database roles and the local policy information; thus, the number of entries to be managed in the role-map file is reduced dramatically compared to the case of using the identity-based mapping. The specification of policies at the VO level eliminates unnecessary authentication, mapping and connections by denying invalid requests at the VO level itself. When users join/leave a VO, the resource providers do not need to add/remove their information individually in the role-map files because we can just grant/revoke their memberships on VO roles. Furthermore, the resource providers can grant or refuse the access requests of specific users by maintaining their authorization

information separately in the role-map files. This enables the resource providers to have the ultimate authority over their resources.

Our performance analysis shows that the proposed RBAC systems incur a small overhead in setting up the security context between the client and server. 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 VOs and reduced administration overheads of resource providers. For multiple large queries, this overhead is almost negligible compared to the execution time of the *perform* document.

References

[1] 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.

[2] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, “The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," Journal of Network and Computer Applications, vol. 23, no. 3, pp. 187–200, 2001.

[3] 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.

[4] A. Grimshaw, “The ROI Case for Grids,” Grid Today, vol. 1, no, 27, 2002.

[5] 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.

[6] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke, “Security for Grid Services,” *Proc. of the 12th Int’l Symp. on High-Performance Distributed Computing*, pp. 48–57, 2003.

[7] C. Neuman, "Security, Accounting, and Assurance," in *The Grid: Blueprint for a New Computing Infrastructure*, I. Foster and C. Kesselman (Eds.), Morgan Kaufmann, pp. 2–48, 1999.

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

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

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

[11] 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.

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

[13] 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.

[14] 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.

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

[16] 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.

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

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

[19] Secretariat of Information Technology Industry Council (ITI), "American National Standard for Information Technology — Role Based Access Control," available at http://csrc.nist.gov/rbac/rbac-std-ncits.pdf, 2003.

[20] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," *Proc. of the 5th ACM conference on Computer and Communications Security*, 1998.

[21] N. V. Kanaskar, U. Topaloglu, and C. Bayrak, "Globus Security Model for Grid Environment," *SIGSOFT Software Eng. Notes*, vol. 30, no. 6, 2005.

[22] 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.

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

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

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

[26] 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.

[27] Organization for the Advancement of Structured Information Standards (OASIS), "Web Services Security: SOAP Message Security 1.0," available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss, 2004.

[28] 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.

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

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

[31] 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.

[32] 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.

[33] 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.

[34] 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.

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

[36] J. Smith, A. Gounaris, P. Watson, et al., "Distributed Query Processing on the Grid," *Int'l Journal of High Performance Computing Applications*, vol. 17, no. 4, pp. 353–367, 2003.

[37] H. Stockinger, "Distributed Database Management Systems and the Data Grid," *Proc. of the 18th IEEE Symp. on Mass Storage Systems and the 9th NASA Goddard Conf. on Mass Storage Systems and Technologies*, 2001.

[38] 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.

[39] 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.

[40] 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.

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

[42] L. Pearlman, C. Kesselman, V. Welch, I. Foster, and S. Tuecke, "The Community Authorization Service: Status and Future," *Proc. of Int'l Conf. for Computing in High Energy and Nuclear Physics*, 2003.

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

[44] 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.

[45] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, and P. Vanderbilt, "Grid Service Specification, Draft 4," Open Grid Service Infrastructure Working Group, Global Grid Forum, 2002.

[46] A. Anjomshoaa, M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, et al., "The Design and Implementation of Grid Database Services in OGSA-DAI," *Proc. of UK e-Science All Hands Meeting*, 2003.

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

[48] 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.

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

[50] Object, Metadata and Artifacts Registry, available at http://ebxmlrr.sourceforge.net/3.0/

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

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

[53] Organization for the Advancement of Structured Information Standards (OASIS), "Hierarchical resource profile of XACML v2.0," available at http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-hier-profile-spec-os.pdf, 2005.

[54] Organization for the Advancement of Structured Information Standards (OASIS), "ebXML Registry Information Model Version 3.0," available at http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rim-3.0-os.pdf, 2005.

[55] Signet, available at http://middleware.internet2.edu/signet