CHAPTER VII

IMPLEMENTATION DETAILS OF THE PROPOSED MODELS
AND AN ANALYTICAL CASE STUDY

7.1 Introduction

In the previous chapters, we presented various architectures and models for solving grid security and access control requirements such as direct authorization, delegation, dynamic access control and fine-grained access control. But enforcement of these models through suitable mechanisms is equally important. This chapter is a logical extension of the Chapters III and IV as we elaborate the implementation details of the proposed architectures. The first step in implementation is to prepare ground for a middleware platform [190], [77] on which the implementation can be done [122]. As grid is a heterogeneous environment, we need to have a middleware to project its federated nature. We explored the possibility of using established grid middlewares like the Globus, the Gridbus etc. We found that Globus is a middleware platform with support for security-related interfaces like authentication, authorization etc where as other middlewares lacked security interfaces. Our models being solutions for security issues in grids needed security-supportive interfaces for implementation. Accordingly, we opted for the latest Globus Toolkit 4.0 as the middleware platform [72], [135]. Next, we need to decide the prerequisites in terms of the software and the technologies. The operating system used is Redhat Linux 9.0 distribution.

This chapter is organized as follows. Section 2 contains the implementation details of various interfaces like the Admin GUI and Client GUI. We also provide a GUI by which the administrator can update or modify new XACML policies [145], [96]. Appendix A describes the installation of Globus Toolkit and shows the steps to write a grid service. Appendix B gives an overview of XACML [34]. In section 3, we show as to how we can implement the architectural frameworks for cross-domain (inter-domain) authorization and delegation. Section 4 describes
the LDAP architecture which we have proposed for the grid environment with its implementation. Appendix C gives a general description of LDAP. In section 5, we give a detailed case study conducted on the security aspects of Garuda Grid and suggest ways to incorporate better security mechanisms in Garuda.

7.2 Implementation of the Interfaces and Main Modules for Single-Domain Grid Enterprise

Globus is a middleware for grid environment. It enables us to write grid applications. Installation of Globus and how to write grid services [57] is explained in Appendix A.

Technologies used

- Java - for Interfaces design, grid service implementation
- mysql - Database for storing user-role, user-delegation information.
- LDAP - server to store details about additional attributes of the users and resources in the organization.
- XACML - to express Access Control Policies [176]

To demonstrate the authorization procedure, we have written a simple grid service called the Mathservice which includes the following operations:

1. add - for adding two numbers
2. sub - subtracts two numbers
3. mul - multiplication of two numbers
4. div - division of two numbers

The whole project is put in the /home/globus/project folder. The mathservice contains two modules:

- MathClient.java
- MathService.java

Both are placed in the /home/globus/project/mathservice/src/org/globus/mathservice/ folder. We then deployed this grid service in globus container using apache ant.

Then run ant deploy (run this command where your build.xml file lies)
7.2.1 Administrator and Client GUI

- **Service Provider:**
The service provider deploys the service in the container and specifies the constraints to access the service based on roles. The constraints are specified in plain language

1. student role allowed to access only addition and subtraction. Time-range to access mathservice is 9am to 5pm.
2. reader role has access to addition, subtraction and multiplication. Time-range to access mathservice is 9am to 7pm.
3. professor role has access to all operations of the mathservice. Time-range is 9am to 11pm.

- **Administrator:**
The administrator has a separate interface to do operations like specifying the organizational hierarchy, user-role assignments, creating and modifying the access control policies etc. The administrator creates access control policies according to the service provider specifications using XACML [116]. The administrator interfaces with the LDAP server through the LDAP browser. Figure 40 shows the graphical user interface for the administrator to create or modify RBAC policies. Figure 41 shows the interface through which the administrator can create or modify access control policies in the XACML framework.

- **Client:**
The Client GUI as shown provides support for the clients to get details of user-role relations, log information about the user’s past delegations and works as an interface for the user to delegate his role to another user. Clients can also query ldap using the LDAP browser. The client GUI for interaction to administrator as well as for delegation are depicted in Figures 42 and 43 respectively.

The attributes we have considered in formulating the access control policies are

- Role of the user
• Requested service
• Operation with regard to that service and
• Time-range to access the service.

XACML decision engine operates with these attributes to make the decision. The custom authorization takes the final decision based on two things.

1. First it checks the decision from XACML decision engine
2. Then checks the system dynamic attributes (free memory, cpu load)

Figure 44 shows the screenshot of user-role assignment as done in Role Based Access Control.

7.3 Main Modules

1. MainPDP.java:
   The custom authorization module is plugged in the globus toolkit.
Figure 41: Administrator GUI for Creating New Policies in XACML

Figure 42: Client GUI
Figure 43: Client GUI for Delegation of Roles Based on Delegation Constraints

Figure 44: User-Role assignment
The custom authorization module implements the interface org.globus.wsrf.security.authorization.PDP;

This module can be found at /home/globus/project/serviceprovider/

**input:** user’s details and request information.

**Output:**

- *True:* If the decision from the XACML decision engine is *permit* and the system dynamic attributes (free memory, cpu load) are below the specified limit.
- *False:* Decision from XACML decision engine is *deny*, or *permit* but the free memory and cpu load are above the specified limit.

```java
//some of the imported packages
import org.globus.util.I18n;
import org.globus.wsrf.security.authorization.PDP;
import org.globus.wsrf.security.authorization.PDPConfig;
import org.w3c.dom.Node;
import org.globus.wsrf.security.authorization.PDPConstants;

//This method takes the final authorization decision
public boolean isPermitted(Subject peer, MessageContext context, QName op) throws AuthorizationException {
    Parameters:
        peer - authenticated client subject with credentials and attributes
        context - holds properties of this XML message exchange
        operation - operation that the subject wants to invoke

    Throws:
```
AuthorizationException

The MainPDP sends the details to the XACML decision engine which is running on a different system using socket function calls.

2. PEP.java:
The PEP receives the request, and tries to get the user’s explicitly selected role or the original role.

```java
//connection establishment to database
try {
    Class.forName("com.mysql.jdbc.Driver").newInstance();
    connection = DriverManager.getConnection("jdbc:mysql://localhost:3306/rbac?","root","");
}

//get the original Role of the user
//if he not selected any role explicitly
try {
    PreparedStatement statement1 = connection.prepareStatement("select * from usr_role where usr = ?;");
    statement1.setString(1,username);
    ResultSet rs = statement1.executeQuery();
    while(rs.next())
        role = rs.getString("role");
}

//get the Role if he explicitly
//selects one of his delegated roles
try {
    PreparedStatement statement2 = connection.prepareStatement("select * from delegate where user = ?;");
    statement2.setString(1,username);
    ResultSet rs = statement2.executeQuery();
    while(rs.next())
        role = rs.getString("role");
}
```
("select * from usr_selected where usr = ?;");

statement2.setString(1,username);

ResultSet rs = statement2.executeQuery();

while(rs.next())
    delrole = rs.getString("role");

If the user selects his delegated role, the PEP tries to get other information like expiry date and time for that delegated role. Compare the expiry date and time with current date and time. If the role’s time limit is not expired, the PEP formulates the RequestContext.

    //formulating Request Context
    RequestCtx request =
        new RequestCtx(subjects, resourceAttrs, actionAttrs);
    Parameters:
        Attributes of the subject, resource and Action
        subjectct has attributes like
        Role, name and Address etc.

If the access control policies are too complex and need more subject attributes than just the subject role, then the PEP contacts the LDAP server. It queries the server for more details about subject using the subject’s identity and sends the RequestContext to the PDP.

    //send the Request Context to PDP
    //and get the Response Context
    ResponseCtx response = pdp.evaluate(request);

3. **PDP.java** This is the core class for the XACML engine, providing the starting point for request evaluation. Before evaluating the RequestContext, we have to initialize the PDP with information like the location of the policies. The supported classes for PDP are
• **PolicyFinder.java**
  This class is used by the PDP to find all policies used in evaluation. It finds a policy based on the request’s context. This involve using the request data and matching this data with the Target element in the policy to decide whether the given policy applies or not.

  //Tries to find policies applicable
  //to the given RequestContext
  public PolicyFinderResult
    findPolicy(RequestCtx context)

XACML policies contains rules, which may specify the conditions also (example: time-range, age > 20 etc). So, we need modules to support comparisons, string matching and boolean logic etc. Some of the classes are

• **AddFunction.java** This class implements all the add functions. It takes two or more operands of the appropriate type and returns the sum of the operands.

• **ComparisionFunction.java** This class implements all of the standard comparison functions like greater-than, less-than and greater-than-or-equal etc., on most of the data types.

• **TargetMatch.java** Represents the SubjectMatch, ResourceMatch, or ActionMatch XML types in XACML, depending on the value of the type field. This is the part of the Target that actually evaluates whether the specified attribute value in the Target matches with the corresponding attribute value in the RequestContext.

  //Determines whether this TargetMatch
  // matches the input request
  public MatchResult match(RequestCtx context)
    context - the representation of the request
    returns - the result of trying to match the
              TargetMatch and the request

• **TimeRange.java** The PDP upon receiving the request, tries to find the
appropriate timerange policy for that RequestContext. If the request to service comes within the specified range the PDP continues to identify the next applicable policies for that request otherwise it returns ResponseContext to the PEP with decision as Deny.

```java
//Parse the timerange policy to check
//the time limits to access the resource
public boolean TimeRange(String attrs[])
```

- **CombiningAlgorithm.java** This class is the base type for all combining algorithms.

  ```java
  //Combines the inputs based on the
  //context to produce some unified result.
  public abstract Result combine(RequestCtx context,
  List inputs)
  context - the representation of the request
  inputs - the things to combine (Policies or Rules)
  returns - a single unified result based on the
  combining logic.
  
  ```

- **DenyOverridesPolicyAlg.java** This is the standard Deny Overrides policy combining algorithm. It allows a single evaluation of Deny to take precedence over any number of permit.

  ```java
  //Applies the combining rule to the set of
  //policies based on the evaluation context.
  public Result combine(EvaluationCtx context,
  List policies)
  Parameters:
  context - the context from the request
  policies - the policies to combine
  Returns:
  the result of running the combining algorithm
  ```

- **PermitOverridesPolicyAlg.java** This is the standard Permit Overrides policy combining algorithm. It allows a single evaluation of Permit to take
precedence over any number of deny, not applicable or indeterminate results. Note that since this implementation does an ordered evaluation, this class also supports the Ordered Permit Overrides algorithm.

```java
//Applies the combining rule to the set of policies based on the evaluation context.
public Result combine(EvaluationCtx context,
                     List policies)
```  
Parameters:
context - the context from the request
policies - the policies to combine

Returns:
the result of running the combining algorithm

For rules also there are combining algorithms like policies
- DenyOverridesRuleAlg
- PermitOverridesRuleAlg

After evaluating the RequestContext the PDP will get a unified result, the PDP then formulates ResponseContext and send that to PEP.

4. ResponseCtx.java

```java
//Represents the response to a request made to the XACML PDP.
public void encode(OutputStream output, Indenter indenter)
```  
Parameters:
output - a stream into which the XML-encoded data is written
indenter - an object that creates indentation strings

When the PEP gets the ResponseContext which is encoded in XACML format, it parses the ResponseContext and gets the decision. The PEP then sends the decision to the custom Authorization module in the globus Toolkit. The PEP also sends the log information like the role in the Request context, time when
the request reached PEP, reasons if the decision is Deny and so on. All this communication is done through sockets over network.

The Authorization module (MainPDP.java) extracts the system specific attributes like free memory and cpu load dynamically and uses these attributes in the final decision making along with the decision from the XACML decision engine.

```java
//extracts system dynamic attributes
public double loadavg() {
    // reads the load average form /proc/loadavg

public int freemem() {
    // reads the memory details from /proc/meminfo

If these criteria are satisfied, the Authorization module allows the client’s request to the service. Otherwise, it denies the request.

7.3.1 Referencing the MainPDP from a security descriptor

The Mathservice uses the custom Authorization Module to control the access to the operations in the service. The service security descriptor file specifies the security configuration for a service.

In the project the Mathservice security descriptor file is located at /opt/globus-4.0.4/etc/math_service/math-service-security-descriptor.xml. After modification the file will be like this:

```xml
<securityConfig xmlns="http://www.globus.org">
<auth-method>
    <GSITransport/>
</auth-method>
<authz value="ascope:org.globus.sampleauthz.MainPDP"/>
</securityConfig>
```

Our service should refer to the security descriptor file first. We have to add the following parameters to the WSDD file, where the path to the security descriptor
The file is relative to GLOBUS_LOCATION. The WSDD file for Mathservice is placed at

/opt/globus-4.0.4/etc/math_service/server-config.wsdd

<!-- This configuration enables the authentication and authorization options for this service found in the specified securityDescriptor -->

<parameter name="securityDescriptor"
value="/etc/math_service/math-service-security-descriptor.xml"/>

7.4 Implementation of Cross-Domain Authorization

The modules which are required to be implemented for cross domain role mapping and authorization are as shown in Figure 23. The code shown below is a part of Authorization Server (AS) of the client domain to which the request for role mapping comes from its redirection server. Here, essentially the code tries to find the type of the message received from the Redirection server. It finds out the domain of the client whose role is being requested and then retrieves its role. Then it creates a client certificate as shown in the sequence diagram and then adds the rank of the role retrieved and sends it back to the AS.

if (msg instanceof ClientNotFoundRequest)
{
    ClientNotFoundRequest im = (ClientNotFoundRequest) msg;
    String domainC = im.getDomainClient();
    System.out.println(domainC);
    if (domainC.equals("nitw.ac.in"))
    {
        String ClientName = im.getName();
        int RankC = getRank(ClientName);
    }
ClientCertificate request = new ClientCertificate(1, RankC, 1);
sendMessage(s, request);
}

if (msg instanceof ClientNotFoundRequest)
{
    ClientNotFoundRequest im = (ClientNotFoundRequest) msg;
    String domainClient = im.getDomainClient();
    // get the domain of the client
    String domainService = im.getDomainService();
    System.out.println("the message received is" + domainClient + domainService);
    if (domainClient.indexOf("ac.in") != -1)
    // check if the domain is a sub domain
    {
        String serverAddress = getAddress(domainClient);
        int serverPort = getPort(domainClient);
        System.out.println("the server and the port are" + serverAddress + serverPort);
        try {
            sock = new Socket(serverAddress, serverPort);
        } catch (IOException ioe)
        {
            System.out.println("ServerWindow() : mainsock : " + ioe);
        }
    }
    sendMessage(sock, im);
    OQMessage msg1 = receiveMessage(sock);
    if (msg1 instanceof ClientCertificate)
    {
        ClientCertificate cert = (ClientCertificate) msg1;
        cert.setRank(2);
        cert.setRank2(cert.getRank2() * getRank(domainClient));
cert.setRank1(cert.getRank1()*getRank(domainService));
sendMessage(s,cert);
}
}
}

The above code is a module from the Redirection server of the ac.in domain. It does the following:

1. gets the domain name of the client and also the name of the client
2. gets the domain for which the service is requested
3. gets the address of the AS of the client domain
4. creates a socket for the same and sends the clientNotFoundRequest message to the AS
5. multiplies the client ranking (Rank1) with the rank of the client domain
6. multiplies the service ranking (Rank2) with the rank of the Service Domain
7. changes the variable rank to 2. This is to indicate that the next AS or RS should multiply the rank of its sub domain with the Rank2
8. passes this client certificate to the server from which request has come

7.5 Implementation of Lightweight Directory Access Protocol

Lightweight Directory Access Protocol or LDAP [2] helps in introducing additional user attributes in the access control mechanism thereby helping in implementing finer access control on the grid resources. We have proposed an LDAP architecture for integration with the grid environment. The LDAP server contains the full details of users and resources in the environment and is helpful in making our access control policies more fine grained with the help of additional attributes. The Grid Resource Information Protocol (GRIP) of Globus [135], makes use of LDAP as the standard resource information protocol. LDAP is also used as a catalog access protocol. But, we mainly associate LDAP with the user attributes to achieve certain level of finer attribute representation and faster access.
7.5.1 LDAP Implementation in Grids

The technology requirement for LDAP implementation in the grid access control include:

- Java - for Interfaces design and service implementation
- mysql - Backend database for OpenLDAP
- OpenLDAP - server to store details about users and resources in the organization

The steps involved in implementing LDAP are:

1. The LDAP server accepts the request from the grid Client. (information on different users and resources in the organization)

2. When a grid client requests for a service, the Policy Enforcement Point (PEP) before forming the XACML request queries the LDAP server for more user attributes
3. The LDAP server then returns the result of the query to the PEP

4. The administrator is also allowed to add new users, change attributes and modify existing users through a administrator GUI

The PEP searches the LDAP server when the access control policies are too complex and also when more user attributes (not just the user’s role) are required.

7.5.2 Advantages of LDAP in Grids

The use of LDAP alongside a database improves the performance of the grid access control mechanism. Using a directory service is a performance enhancing practice. In a grid environment, reads must be performed very efficiently so as to service the access requests as fast as possible. As mentioned earlier, a grid mostly spans a large geographical area comprising of many domains with databases for each such domain. But the complex functions supported by such a database management system are generally not required in a grid environment. As directories are optimized for read accesses it becomes advisable to use them in the grid environment. Secondly, the very structure of LDAP is hierarchical and hence offers a tree-like view of the whole grid environment. This is particularly necessary in organizations so that the client can get to know the organizational structure. Combining a relational database management system with LDAP is unadvisable as the grid environment is vast and complex and also because the data models are very different. Representing directory data with a relational database requires splitting the data into multiple tables. This means that accessing data, from even one entry requires seeking/searching on different disk areas which in turn results in reduced system performance.

7.6 Case Study on the Security Aspects of Garuda Grid

As this thesis work deals with the core security challenges in grid computing environment, it is felt necessary to conduct a study on any existing grid infrastructure. We opted for the National Grid Computing Initiative of India - code named as GARUDA [4] - for this purpose. This chapter provides an overview of the GARUDA grid, which include its strategic objectives, key deliverables and major components (both physical as well as technological). Next, we discuss various
approaches adopted in GARUDA grid to address the security issues. Some of these techniques have already been deployed and initiated while some are in the design and implementation phases. The organizations and the standards which help implement GARUDA security have been briefly explained as per the available details. Then, it explores ways to incorporate some of the access control and authorization methodologies suggested in this thesis with GARUDA grid security. The chapter ends with a summary of the GARUDA grid security.

7.6.1 GARUDA - An Introduction

GARUDA is a collaboration of science researchers and experimenters on a Nationwide grid of computational nodes, mass storage and scientific instruments in India that aims to provide the technological advances required to enable data and compute intensive applications for the 21st century. The Nationwide computational grid GARUDA aims to connect 17 cities across the country in its Proof of Concept (PoC) phase with an aim to bring “grid” networked computing to research labs and industry. An effort under the Center for Development of Advanced Computing (C-DAC) [3], GARUDA aims to accelerate India’s drive to turn its substantial research investment into tangible economic benefits by strengthening and advancing scientific and technological excellence in the area of grid and peer-to-peer technologies. The strategic objectives of GARUDA are to:

• create a test bed for the research and engineering of technologies, architectures, standards and applications in grid computing
• bring together all potential research, development and user groups to develop a national initiative on grid computing
• create the foundation for the next generation grids by addressing long term research issues in grid computing

The following are the key deliverables identified as important to achieve the GARUDA objectives:

• Grid tools and services to provide an integrated infrastructure to applications and higher-level layers
• A Pan-Indian communication fabric to provide seamless and high-speed access to resources

• Aggregation of resources including compute clusters, storage and scientific instruments

• Creation of a consortium to collaborate on grid computing and contribute towards the aggregation of resources

The major components of GARUDA as shown in Figure 46 include the computing resources, high-speed communication fabric, middleware and security mechanisms, tools to support program development, collaborative environments, data management and grid monitoring and management. Access portals and specialized problem solving environments provide a seamless user interface to the grid.

![Figure 46: Core Components of GARUDA](image)

The technology components of GARUDA include:

• Access methods and problem solving environments

• Data management
• Program development
• Collaborative environment
• Grid middleware and security
• Grid monitoring and management

The GARUDA High-Speed network is a Layer 2/3 MPLS Virtual Private Network (VPN) connecting select 45 institutions across 17 cities at 10/100 Mbps with stringent service level agreements with the service provider. This grid is a precursor to the Gigabit speed nation-wide Wide Area Network (WAN) connecting high performance computing resources and scientific instruments for seamless collaborative research and experiments. The high speed network is being established at all the Garuda partner institutes in close collaboration with ERNET which is also responsible for the operation, maintenance and management of this network.

7.6.2 Approaches to Handle Security Issues in GARUDA

The following section highlights the existing practices for implementing security in GARUDA-grid.

• evolving security practice statements and GARUDA security policies
  – evolve responsibilities for resource providers as well as users
  – carry out risk assessment and fix the weakness
  – form a group to evolve policies

• participation in International Grid Trust Federation (IGTF)
  – Asia Pacific Grid Policy Management Authority (PMA)
  – European Grid PMA
  – The Americas Grid PMA

• GARUDA - Computer Emergency Response TEAM (G-CERT)
  – To be a collaborative effort by all/many partners
  – To establish points of trusted contacts for computer security threats in GARUDA
– To provide certain mandatory services like announcements, vulnerability analysis, technical analysis and reports, education, incident tracing, intrusion detection, audits and penetration testing, security consultancy, risk analysis etc

– Create documents pertaining to best practices and policy guidelines (site security and inter-site security handbook)

– To develop capabilities to support activities such as tracking and tracing intruder activity and active detection of such activities

– Interface with CERT-IN and other CERTs

– GRIP - Guidelines and Recommendations for Security Incident Processing

7.6.3 User Registration in GARUDA

GARUDA uses a Portal Based User Registration Service - PURSE for user registration purposes. The registration authority is called RA - Registration Authority (GARUDA Contact Person) and the certificate authority is called CA - Certificate Authority (CDAC-KP). As a portal based user registration service, PURSE helps in registering users through web-based applications using the Grid Security Infrastructure, based on PKI and X.509 certificates. GARUDA uses PURSE for registration of new users. GARUDA management has identified from each partner institute, one or more GARUDA Contact Persons (GCP), who act as RA from the Partners side. A certificate is a digital document that certifies that a certain public key is owned by a particular user. This document is signed by Certificate Authority (CA). CA will send the information about the requestor to RA (GCP). CA will have no knowledge about the requestor and will wait for the GCP. A Registration Authority (RA) is an authority in a network that verifies user request for a digital certificate and tells the Certificate Authority (CA) to issue the certificate. The following figure shows the PURSE architecture.
7.6.4 Summary of GARUDA Grid Security

GARUDA security has been implemented in the form of two core mechanisms namely authentication and authorization. The authentication features include

- GARUDA Certificate
  - Subject Name
  - Public Key of the Subject
  - Identity of GARUDA CA
  - Digital Signature of GARUDA CA

- Adheres to GSI
  - Credential Delegation
  - Single Sign-On

The authorization features are

- User Mapping
- DN to Pool of Unix accounts
7.6.5 Suggestions for Improving Garuda Grid Security

Authentication is being used as the basic form of user identification and validation in Garuda Grid. Authorization attempts at providing restricted access rights to the users seems to be at the very initial stages in Garuda. The Moab Scheduler from Cluster Resources interfaces with Globus for user management and security. We suggest the use of an authorization-centric security mechanism in Garuda. The Role-Based Grid Authorization Model which has been proposed in Chapter III of this thesis could be incorporated in Garuda. This model, on one hand provides support for all the proven techniques of access control and authorization (as in RBAC), for a grid environment. Additionally, it facilitates delegation which can be a very useful indirect authorization mechanism for securing Garuda resources. Our framework is complete with features like revocation, multi-step delegation and dynamic access control.

7.7 Chapter Summary

The chapter summarizes the experimental set up and implementation details of the contributions made in Chapters III and IV of this thesis. We show as to how we can write a grid service, design an authorization mechanism for the service and then enforce the mechanism for secured resource access. We provide authorization techniques for single domain as well as cross-domain environment. We have made use of various tools and techniques for implementation of access control like the Globus middleware, eXtensible Access Control Markup Language (XACML), Lightweight Directory Access Protocol (LDAP). We used mysql as the back-end database for representing the RBAC policies. The entire implementation was done on the Redhat Linux platform. We supported the implementation with a case study on Garuda grid’s security features.