UNIVERSITY OF CALIFORNIA, SAN DIEGO

A Service-oriented Architecture for Authentication and Authorization

A Thesis submitted in partial satisfaction of the

requirements for the degree Master of Science

in

Computer Science

by

Hamidreza Hamedtoolloei

2009

The Thesis of Hamidreza Hamedtoolloei is approved and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California, San Diego

2009

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT OF THE THESIS

A Service-oriented Architecture for Authentication and Authorization

by

Hamidreza Hamedtoolloei
Master of Science in Computer Science
University of California, San Diego, 2009
Professor Ingolf H. Krüger, Chair

Many applications require access to large quantities of data and computational resources that are often distributed over a wide-area network. Grid computing infrastructures provide a platform to run these applications, but their heterogeneous nature makes security a vital component of grid systems. The standard technology for grid security is the Grid Security Infrastructure (GSI). Although GSI is well-known in academic and government settings, GSI-based security systems are usually difficult to deploy and use. The Grid Account Management Architecture (GAMA [18]) was developed to make Grid security easy for system administrators and users by extensive use of web-services technology. Nevertheless, GAMA has a few limitations that make it unattractive to many communities. For example, it does not offer a usable resource authorization mechanism. Moreover, it is tightly coupled to the selected technologies.

We propose GAMA 2.0, which is a service-oriented architecture that addresses both authentication and authorization concerns. Moreover, the multi-tier architecture is pluggable to cope with the rapidly evolving relevant technologies. We have implemented the GAMA 2.0 reference infrastructure using well known programming techniques such as polymorphism and the Command pattern. In addition, a comprehensive testing strategy, which includes unit-testing and scenario-testing, as well as detailed exception handling has been employed to ensure correctness and robustness of the in-

frastructure. Although performance was not the driving factor, we have managed to increase GAMA 2.0's performance by applying a few optimization techniques.

Chapter 1

Introduction

1.1 Motivation

There are many problems in engineering and scientific fields that cannot be resolved using current personal and supercomputers, although we have witnessed a tremendous performance increase in computers and networks in the recent years. In fact, since these problems are large-scale and computationally intensive, their solutions demand a range of heterogeneous resources. Nevertheless rapid advances in relevant technologies have resulted in ubiquity of the Internet, availability of powerful computers, and high speed network technologies. These advances, in turn, have led to development of computing platforms with a wide variety of distributed resources. There are various types of distributed platforms such as Clusters [21], Grids [24], and peer to peer (P2P) networks [31], which share the following main benefits:

- Functional Separation: purpose of each component in the system is determined based on the functionality it provides.
- Inherent distribution: system entities such as users, resources, and sites are inherently distributed.
- Reliability: data can be stored and replicated at different locations to increase reliability.

- Scalability: addition of more resources or users to increase performance or availability.
- Economy: resources can be shared by many entities to reduce the ownership cost.

These features allow parallel and possibly independent entity operations in a distributed system. In addition, proper entity communication leads to coordinated actions, and task completions at well-defined states. Since entities are heterogeneous, failures are independent, and usually no single entity has the knowledge of the entire state of the system.

To further support the case for distributed computing platforms, G. V. Laszewski presents the following weather forecasting example promoting usage of grid environments [45].

1.1.1 Meteorology Case Study

In the meteorology domain, extensive calculations based on weather observations are required for precise weather predictions. The first modern idea for numeric weather prediction was introduced in 1922, and within two decades, a prototype of a predictive system was developed by Von Neuman and Charney [17, 40]. Since then, numerical weather prediction has become mature as computers become more powerful. While early models rely solidly on observations, scientists recently have realized that complex chemical processes and their interactions with land, sea, and atmosphere have to be considered for more detailed and accurate analysis (see Figure 1.1). Introducing climate variables makes prediction models extremely complex for the following reasons:

- Sophisticated sensor networks must be employed to collect climate variables and weather observations.
- Enormous amount of data need to be fed into complicated models for accurate analysis. In addition, predictions must be delivered to consumers.

The increased in prediction system complexity requires a massive amount of computational and storage resources. Despite the enormous power of supercomputers, they

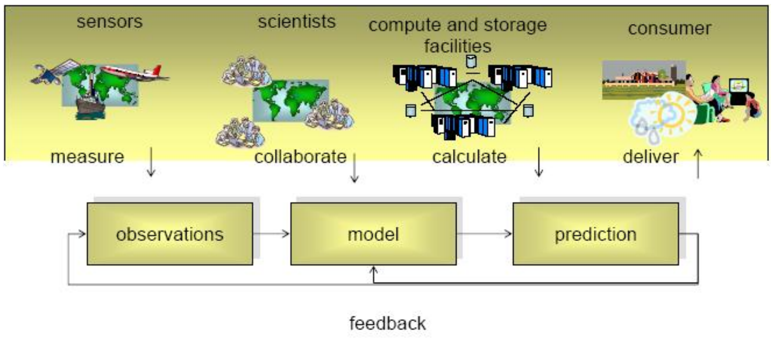


Figure 1.1: weather forecasting is a complex process (adopted from [45])

are likely to be inadequate for complex forecasting models; promoting the usage of distributed computing to realize flexible, secure, and coordinated sharing of a wide variety of resources.

Several enabling factors allow for development of sophisticated forecasting models on distributed computing platforms.

As technology advances, sensor infrastructure has expanded from surface temperature measurements to Doppler radars and weather satellites. In addition, remote access to a variety of databases collecting measurement data is realized via the Internet and communication satellites. Moreover, computer power has increased steadily, while network production and hardware cost has decreased. Computer speed, for instance, has doubled in every eighteen months for the past couple of decades supporting Moore's law [34]. Also, over the past few years, network bandwidth has increased at a much higher rate and some experts suggest that it doubles every nine months [45].

Change in modality of computer operation is another enabling factor encouraging the usage of distributed computing platforms. The first generation of supercomputing enterprise was mostly based on high-end mainframes and vector processors in a single institution and an administrative domain. As new network technologies and the Internet [5] promote connectivity of computers and organizations, distributed computing environments are gaining attention. As a part of transition to decentralized computing, it has become natural to collect geographically dispersed and heterogeneous computer resources typically as networks of workstations and supercomputers.

Among the enabling factors are increase in capacity, capability, and change in modality that has resulted in a shift from centralized computing center to a decentralized environment.

1.2 Security Challenges

Despite the fact that distributed platforms offer numerous advantages over centralized environments, they need to address several challenges. Some of the important issues are presented below:

- Heterogeneity: various entities in the system must be able to inter-operate with one another despite differences in hardware architectures, operating systems, communication protocols, and security models.
- Transparency: communication between the entities should be hidden from the end user and the entire system should appear as a single logical unit.
- Fault tolerance and failure management: the entire system should not fail with failure of one or more components.
- Scalability: adding resources to the system should enhance system's performance or availability. In addition, the system should work efficiently with increasing user population.
- Concurrency: shared access to resources should be possible.
- Migration and load balancing: distribute computation load among available resources to enhance performance without affecting the operation of users or applications
- Security: access to resources should be secured to ensure only known users are able to perform allowed operations.

Among the challenges mentioned above, security is one of the key issues. Security is a vital component of distributed systems and is somewhat different from operating system security. In distributed systems there is no central, trusted authority that arbitrates interaction between users and processes. Instead, a distributed system usually runs on top of a large number of connected independent hosts. Each host may run a different operating system and has its own set of internal security rules and policies. A distributed environment security solution needs to address a variety of issues such as group membership, authentication, and access control.

Considering grids as one of the popular distributed computing platforms, Ian Foster performs the following detailed security analysis of grid systems [25].

1.2.1 The Grid Security Problem

Figure Figure 1.2 illustrates a scientist, member of a multi-institutional scientific project, running a data analysis program. Since the data is stored at site C, a distant location, site A sends a data analysis request to site C. Also, to successfully execute the analysis program, a simulation needs to run to compare some experimental results to predictions. Consequently, site C contacts site D where a resource broker is maintained to locate idle resources that can be used for a computation. The resource broker, in turn, initiates computation on computers at two sites (E and G). These two sites contact site F to access required parameter values on its file system. In addition to communicating amongst themselves, sites (E and F) also communicate with the resource broker, the original site, and the user. Although the described example is somewhat contrived, it illustrates many unique features of the grid computing environment. A few of these features are discussed below:

- User participants in a grid environment include members from many institutions and will change regularly over time. (i.e. the user pool is large and dynamic).
- Individual users and institutions determine whether and when to share resources Therefore, sites and the quality of resources change frequently. (i.e. the resource pool is large and dynamic).
- During its execution, a computation can dynamically acquire and release re-

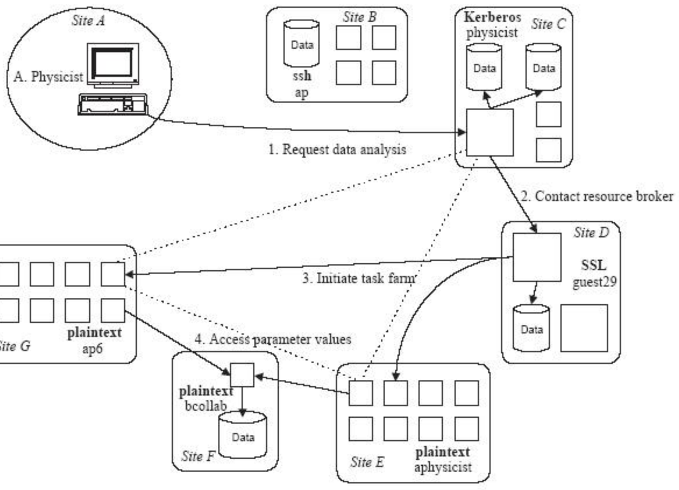


Figure 1.2: Example of a large-scale distributed computation (adopted from [25])

sources. In the above scenario, for instance, the data analysis computation acquired and released resources at five locations. (i.e. computation is a dynamic collection of processes running on different resources and sites).

- In grids, processes forming a computation may communicate using a well-defined mechanism. (i.e. during the lifetime of a computation, communication connections such as TCP/IP sockets may be created and destroyed dynamically).
- Individual institutions enforce their own set of authentication and authorization rules. In Figure Figure 1.2, institution dependent access control is illustrated by showing the local polices that apply at the different sites. These include Kerberos, plain-text passwords, etc.
- At different sites, a participant user will have different local accounts and credentials for the purpose of proper access control. A regular user of one site can be an administrator of another site.
- Resources and users may be located in different countries.

To summarize, the challenge we are facing is to provide a security solution that enables sharing and coordinating of a wide variety of resources that are geographically

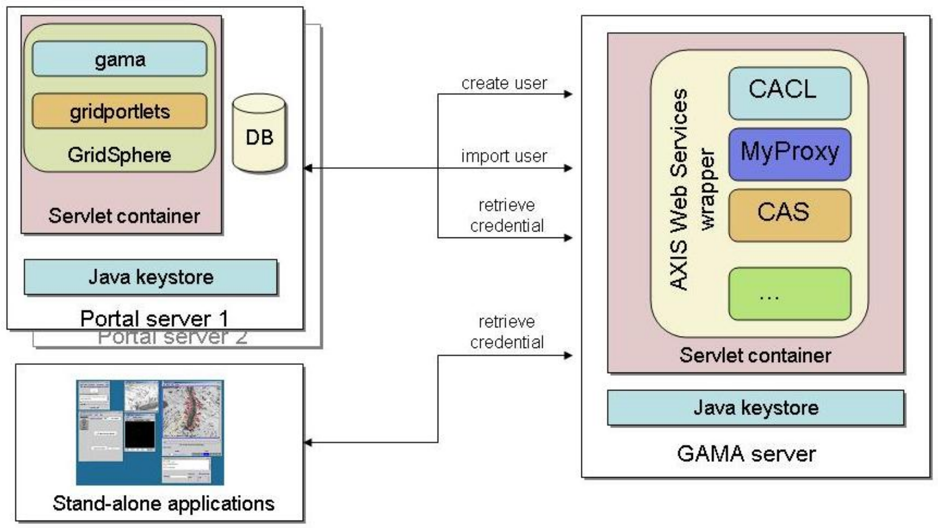


Figure 1.3: GAMA Architecture (adopted from [29])

distributed and owned by different organizations.

1.3 GAMA 1.X

Grid systems depend on a collection of back-end software packages and technologies to create and manage Grid credentials for users. Therefore, system administrators spend a significant amount of time and effort installing and maintaining these technologies. Moreover, users often are exposed to security details as they are required to explicitly issue and manage their own credentials. The Grid Account Management Architecture (GAMA [18]) has been designed and implemented to overcome these challenges. It bundles many of the security-related technologies that are not primarily designed for interoperability in order to achieve a complete end-to-end security system that works out-of-the-box. Figure Figure 1.3 depicts GAMA's architecture, which consists of two main components: a back-end server component that manages users' identities and credentials, and a set of portal components that provide different web interfaces for both users and administrators.

The GAMA back-end server provides a set of services that run on a remote, protected machine. There is no end-user access to this machine and the services are exposed to public only via web-service calls. User management services such as addUser, deleteUser run in a secure web-service container that requires mutual authentication between the GAMA server and GAMA clients. Other services such as login run in a less secure container, which does not require mutual authentication. The back-end services use CACL[30] for issuing users certificates, MyProxy for storing and retrieving credentials, and CAS [38] for defining and using roles for authorization. These software packages and services can be difficult to install and some require a substantial amount of configuration. To make the installation and configuration process easy for system administrators, the Rocks [37] clusters management software has been employed to create a GAMA server roll [37]. Rocks installs all GAMA components and performs post installation configuration to provide a fully functional GAMA back-end serverappliance "out-of-the-box" with minimum system administrator intervention.

The remainder of the GAMA system consists of a set of portlets that allow end-users (administrators or users) to interact with GAMA. Users, for example, can perform several activities such as request an account, login, and access web-based applications. At the same time, administrators have the ability to easily define policies and perform user management tasks such as delete a user.

1.4 Problem Statement & Solution Proposal

Existing grid security systems are **difficult to deploy and use**. These systems are usually based on the Grid Security Infrastructure (GSI) technology, which is well-known in academic and government settings. GSI-based system administrators are usually required to invest significant time and effort in installing and configuring many software packages and technologies. Furthermore, end-users are exposed to low level security-related tasks such as issuing and managing their credentials. Another limitation of GSI-based systems is the lack of flexible authorization mechanisms. These problems are even more apparent in academic settings where mid-size scientific projects spanning few organizational sites desire efficient deployment, user friendly interfaces and flexible access control solutions.

In recent times, **service-oriented architectures** have witnessed a lot of attention in designing and constructing distributed systems. Grid computing, in particular, has shifted toward service-oriented architecture leveraging web-services technologies. The shift will likely continue with the recent development and release of Web Services Resource Framework (WSRF)[14] which allows for easier aggregation and composition of data and applications in novel ways. At the same time, analyzing several grid infrastructures suggests that many grid systems have **similar security requirements** despite the fact that they have different goals, and are usually tailored for a particular set of problems. In addition, detailed requirements elicitation sessions augmented with complex categorization and prioritization procedures have revealed that many mid-size, academic-related projects spanning few domains share various similar requirements [33].

Merging the two trends mentioned above, GAMA (Grid Account Management Architecture) was born to **overcome many challenges in GSI-based security systems** (see section 1.3). GAMA has been successfully employed in various communities mainly because of its efficient deployment and simple usage. At the same time, many projects were hesitant to use GAMA 1.X because it did not offer a usable authorization solution and did not support LDAP-based authentication. In addition, GAMA 1.X had explicit reliance on the employed technologies such as MyProxy and CACL. As grid security technologies are rapidly changing and improving, it is very difficult to incorporate new relevant technologies into the old versions of GAMA.

To address GAMA 1.X's shortcomings, GAMA 2.0 was introduced. GAMA 2.0 offers a flexible, fine-grained resource authorization solution based on RBAC models. Moreover, to attract the projects that desire to use LDAP server for authentication, GAMA 2.0 is designed to support LDAP-based user authentication. Similar to GAMA 1.X, it aggregates many of the existing trusted tools that support building a GSI-based security system, but are not designed for interoperability. However, the multi-tier architecture is pluggable to cope with the rapidly evolving relevant technologies, and is flexible to be employed in various non-crossing projects.

1.5 Related Work

In this section, we are going to present a few existing identity management solutions. We only document these systems without analyzing or comparing them to GAMA 2.0. Later, in the evaluation chapter, advantages and disadvantages of these systems with respect to GAMA 2.0 are discussed.

1.5.1 PURSE

Portal-based User Registration Service (PURSE [26]) consists of set of tools and technologies to automate difficult security-related tasks such as user registration, credential creation, and credential management. By employing PURSE, end-users are freed from the need to create or manage public key credentials, consequently simplifying the Grid experience and reducing opportunities for error.

The PURSE's back-end is a collection of Java APIs that stores user information, issues and stores user certificates, and allows for subsequent use of those certificates to access Grid resources. Moreover, the system has functionality to support credential renewal and revocation. PURSE integrates several common tools to provide these functionalities. For example, it uses MYSQL to store user data and MyProxy repository to persist user credentials. Also, depending on application requirements, it employs either SimpleCA or an external certificate authority to generate and sign user credentials.

The back-end is complemented with a front-end user interface, typically a Web portal, to ease registration and credential management tasks. The portal runs on the same machine as the back-end tools such as MyProxy certificate authority. Therefore, the front-end web portal can easily access the back-end services via Java library calls.

1.5.2 GridAuth

GridAuth [6] is a user credential management solution for distributed systems with wide range of heterogeneous data and resources like Grids. The pluggable GridAuth architecture is easily configurable as well as extensible. Therefore, it is appealing to many communities that require credential management, advanced authorization, and secure authentication. GridAuth system supports a wide variety of use cases such as authentication, authorization, single-sign-on, and delegation. It comprises two major components: client API available in Perl, Java, and PHP and back-end server which is written in Perl.

GridAuth provides client APIs used by external applications or grid nodes. These APIs hide the low-level implementation details that are handled on the serve side and provide high-level functions to application developers for authentication and authorization. The client APIs are extremely lightweight, secure, and portable as they use standard HTTP over SSL to communicate with the central backed server.

The GridAuth back-end server consists of Service Handler and Plug-in Stack. The service handler receives requests from any client API then it passes the request to the plug-in stack. The service handler manages a customized list of plug-ins defined for the specific grid instance. The stack consists of predefined and custom built plug-ins, each one implementing required interface functions such as login and logout. To process a login request from client API, for example, the service handler will call login methods on each of the plug-ins in the stack. When all plugin calls return (i.e. successful login) the service handler sends the response back to the client.

1.5.3 FusionGrid

FusionGrid [19] is a computational grid that consists of geographically distributed services and data repositories. These services are used by members of the National Fusion community who are scientists and collaborators from various universities and labs. The development team has put significant time and effort testing and evaluating a wide range of security-related technologies and tools. They have also created the

FusionGrid security system that supports both user authentication as well as resource authorization. In addition, the FusionGrid system fulfills many of the pivotal grid security requirements such as delegation and single sign-on. Considerable attention has been devoted to make the system as simple and usable as possible to the benefit of fusion scientists who are not grid security experts.

The FusionGrid security team has examined several approaches to user authentication. Early FusionGrid investigations suggested employing X.509 credentials and the Globus Security Infrastructure (GSI) as means to uniquely identify users and to provide a single sign-on capability. Although the system was a success, researches experienced a difficult time managing their own certificates. Several quick solutions such as replacing the web interface with a set of command-line scripts and providing certificate management training did not fix the problem. Consequently, developers decided to remove the burden of certificate management from scientists. They created a credential manager which consists of a DOEGrids CA, a MyProxy server, and a custom Web interface. They all run on a dedicated and secured host to provide user registration and authentication.

To decide on authorization technologies, similar to authentication, the FusionGrid security team has experienced an iterative process investigating several alternatives. Eventually, they decided to create a new authorization system that would meet the needs of resource stakeholders while being as simple and easy as possible for scientists to use. They developed Resource Oriented Authorization Manager (ROAM [20]) which is a custom, pull-mode authorization system enforcing consistent authorization scheme over the distributed resources of FusionGrid. ROAM is a two-tier architecture system. The back-end is a relational database containing all users, resources, and authorization information. It consists of three pivotal entities: resources, permissions, users. These entities collectively represent FusionGrid authorization by indicating user X has permission Y on resource Z. The front-end is a web interface that provides authorization management interface. Users such as stakeholders and resource providers access the authorization decision interface via HTTPS protocol.

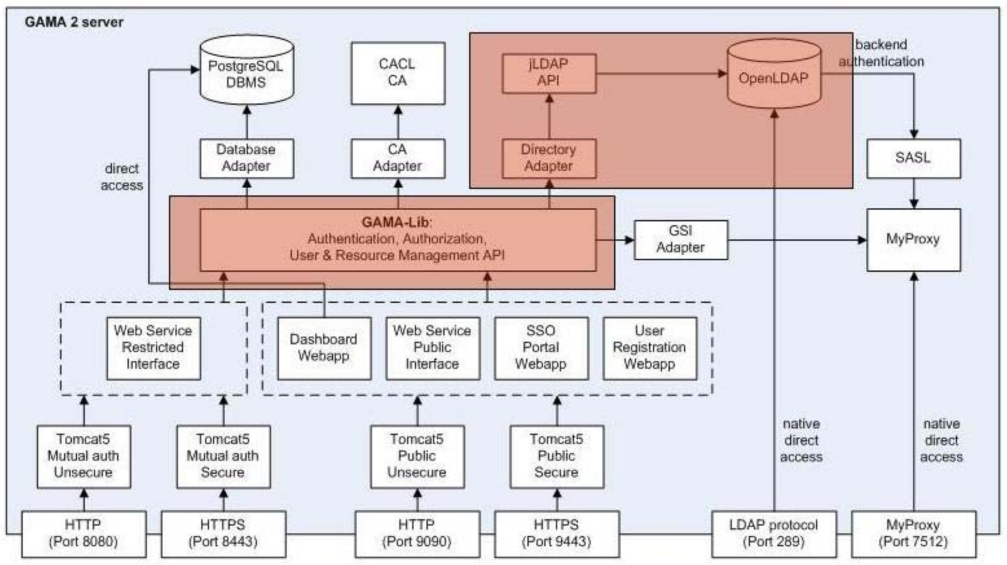


Figure 1.4: Graphical Representation of the Author's Contributions (highlighted rectangles)

1.6 Thesis Organization and Author Contribution

In chapter 1, the author of this document presents advantages and challenges of distributed computing platforms such as grids. After identifying security as one of the essential grid components, he discusses well-known authentication and authorization schemes in chapter 2. In the remaining chapters, the author discusses GAMA 2.0's requirements, design, and implementation, while focusing on his contributions in these areas (see Figure 1.4). In chapter 3 and 4, he presents several GAMA 2.0' usage scenarios and security requirements respectively. Then he discusses the architecture of GAMA 2.0 in chapter 5. He emphasizes his main contributions in architecture design such as security models, back-end services, and use cases. The next chapter highlights the author's contribution to code implementation, which include developing and testing many back-end services. In chapter 7, the author compares GAMA 2.0 to other identity management solutions and evaluates the back-end services in terms of their execution time. Finally, he provides a summary of and direction for future work in chapter 8.

Chapter 2

Grid Authentication & Authorization Models

2.1 Introduction

In a grid context, there is a sharp distinction between authentication and authorization due to dynamic and large number of users and resources with different management policies. Authentication is concerned with the identity of an entity such as a user within a given context such as a virtual organization [35]. On the other hand, given an entity with identity I, authorization captures what I can access and do [35]. In this chapter, we discuss popular authentication and authorization schemes. First, we present three models of authentication, while noting that the certificate based model is the most common approach. Also, we discuss several authorization schemes and technologies. We conclude this chapter by briefly mentioning GAMA 2.0's authentication and authorization models.

2.2 Authentication Models

Recalling many authentication schemes discussed in [41, 35, 15, 10], we describe the main authentication models in more details in this section.

2.2.1 Certificate Authentication

Perhaps the most popular and prevalent grid authentication mechanism is the certificate based authentication. It relies on a public key infrastructure (PKI [28]) to provide means for trusted authority to sign information to be used for authentication purposes. In this approach, each entity has a public-key based cryptographic credential in the formulation of a certificate such as X.509 certificate. A trusted certificate authority (CA) signs and certifies these certificates which typically include the following information:

- A subject name usually in form of distinguished name (DN). A DN uniquely identifies the person or object that the certificate represents.
- The subject's public key.
- The identity of a Certificate Authority (CA) that has signed the certificate. This is required to certify that the public key and the identity both belong to the subject.
- The digital signature of the named CA.

To request a certificate a user starts by generating a public-private key pair. The private key is stored encrypted with a pass phrase the user provides, while the public key is put into a certificate request. Then the user sends the certificate request, often encrypted using the CA's public key, to the CA. The CA usually includes a Registration Authority (RA) which applies appropriate policy rules that verifies identity of the user. After successful verification, the CA creates a certificate with the appropriate information such as user public key, and certificate expiration date. Moreover, it signs the certificate using its own private key.

Many grid systems that are based on PKI use certificates differently. Instead of employing long-lived digital certificates, they take advantage of proxy certificates. A proxy certificate is a special kind of X.509 certificate that does not require a signature of a certificate authority. Because proxy certificates are short-lived, usually several hours, and can be generated during login stage, they can be used to provide users with mechanisms of single sign-on (SSO) and credential delegation. However, they present new challenges such as lack of revocation mechanism to revoke existing proxy certificates. To reduce the management and maintenance burden on users, MyProxy Server is created to manage user credentials. MyProxy is a widely deployed online credential repository for grid that runs on a secured host. It enables users to store their certificate and private key in a MyProxy repository and retrieve a short-lived proxy certificate from the MyProxy later when needed. This is great advantage over storing user's X.509 certificate and private key on user's own machine. Nevertheless, one of the key drawbacks of the PKI is that current tools and technologies for certificate management are too complicated for users.

2.2.2 Kerberos Authentication

Kerberos [41] is a secret key based mechanism for providing authentication in the network. It is a third party authentication solution that includes a trusted central authentication service, Key Distribution Center (KDC), to authenticate users. The KDC consists of two major components: the Ticket Granting Server (TGS) and the Authenticating Server (AS). These two work collectively to provide a centralized user authentication solution. Moreover, each user and service share a secret key with KDC and KDC generates tickets indicating identity of their owners [35]. A ticket has a limited lifespan and relates a user to an end-service.

Figure Figure 2.1 depicts the required operations to request a service. The following steps are performed:

1. The client connects to the Authentication Server (AS) asking access to the Ticket Granting Server (TGS).

2. The AS generates the TGS ticket and encrypts it with the TGS's public key. It also issues a random session key which later will be used between the client and The TGS. The entire information is encrypted using the client's private key, which can be the client's password.

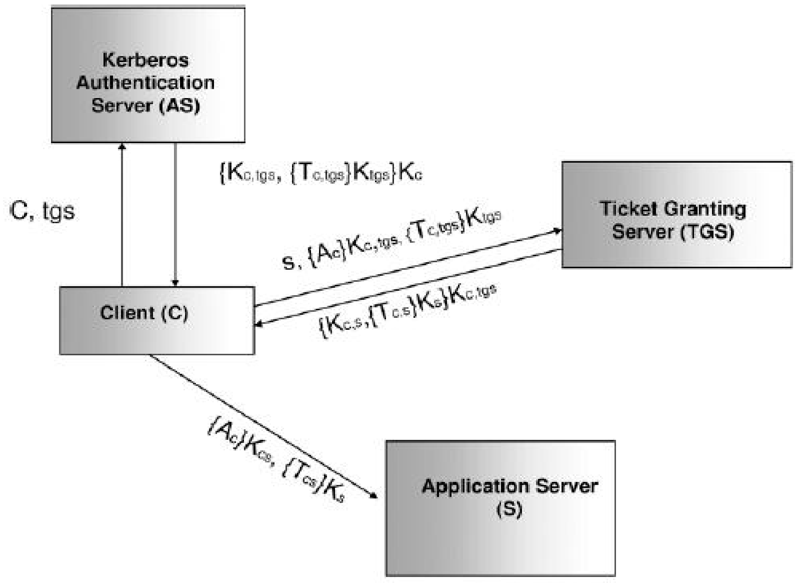

Kerberos Authentication Server (AS)
$\{K_{c,tgs}, \{T_{c,tgs}\}K_{tgs}\}K_c$
C, tgs
Ticket Granting Server (TGS)
$s, \{A_c\}K_{c,tgs}, \{T_{c,tgs}\}K_{tgs}$
Client (C)
$\{K_{c,s},\{T_{c,s}\}K_s\}K_{c,tgs}$
$\{A_c\}K_{cs}, \{T_{cs}\}K_s$
Application Server (S)

Figure 2.1: Overview of the Kerberos system (adopted from [15])

3. The client sends the ticket information along with the authentication information and name of the target server to the TGS. The authentication information typically includes time, client information, etc. The client encrypts the client-TGS ticket with the TGS's public key.

4. The TGS replies with the ticket that can be subsequently used by the client to access the server.

While Kerberos meets many of the basic requirements for grid authentication, it presents a major problem. Namely, every authentication among users and services involves KDC intervention as it holds a list of all users and services. Moreover, implementing inter-organizational authentication (called cross-realm authentication) is extremely difficult with Kerberos. As each realm is served by an independent KDC, establishing a cross-realm trusted relationship is one the most serious obstacles that prevent Kerberos from being deployed in Grid systems. However, Kerberos has become a de-facto standard for local security infrastructures operated by many institutions [35].

2.2.3 Password Authentication

Many computer systems use passwords to authenticate users since they are simple to implement and computationally inexpensive. Authentication consists of checking supplied user password against a hash of the password stored in some database. The checking process is simple and does not require any complicated computation. However, password based authentication systems are vulnerable in several ways. Due to the lack of confidentially, for example, passwords need to be encrypted to prevent adversaries from tapping into the system. Moreover, choosing a password itself is a difficult task because there are many automated tools that can guess a password using methods such as brute force. To overcome some password based authentication shortcomings, researches have developed One Time Password (OTP) technology [10]. Unlike a static password, a one-time password changes each time a user logs in. Consequently, it is difficult for malicious adversary to tap into a system.

Despite the recent advances in password based authentication such as OPT technology, it is hardly employed in grid systems as it offers several new challenges. It is extremely difficult to fulfill some grid requirements such as Single Sign-on and credential delegation via password authentication, although recently there have been some advances in this area [35]. Nevertheless, passwords are usually used to login to a grid infrastructure (i.e. to access private keys in MyProxy repositories, obtain Kerberos tickets, etc.).

2.3 Authorization Models

Grid Security Infrastructure (GSI [43]) is the de facto standard security infrastructure on the grid and its authorization is based on a grid-mapfile. This file provides mapping information between grid-wide user identities and local user identities such as Unix account. Upon successful mapping, the local identity can be used to enforce local policy decisions such as file access. Although the grid-mapfile authorization is easy to implement, it offers several shortcomings such as lack of scalability and consistency

[38].

To overcome grid-mapfile authorization limitations, many alternative authorization solutions have been proposed. These solutions generally are based on two well-known authorization models: push or pull model. After reviewing related authorization modes discussed in [38, 39, 16, 20, 27], the author summarizes the essential ones in this section.

2.3.1 Push Model

In the push model, an authorization subject that corresponds to a grid user first contacts an authority to obtain his/her authorization rights. The authority issues and returns a token or a message that contains the user's access rights. The subject can use the token to contact a resource and request a specific service. Using the token, the service owner many accept or reject the request and will report this back to the requesting subject.

CAS

A Community Authorization Service (CAS [38]) server is designed to maintain authorization information for all community entities and enforce fine-grained access control policies. CAS is a centralized trusted third-party authorization system that stores users' rights in a backend database. The CAS server contains policy statements that indicate what permissions are granted to which users and which resources the permissions are granted on. It delegates all sets of rights to a user based on the user's role in the community. The user's set of rights is the intersection of the set of rights granted to the community by the resource provider and the set of rights granted to the user by the community. Using CAS, authorization decisions are made at a resource level when a user presents his access rights to the resource.

CAS is designed and implemented to work with the public key authentication and delegation mechanisms of the Grid Security Infrastructure (GSI). To access a resource, a community user first generates a proxy certificate that is signed by his user credential. Then the user presents his proxy certificate to the CAS server which issues new certificate called CAS proxy certificate. This certificate contains the user’s access rights and capabilities. Moreover, the CAS server uses the security assertion markup language (SAML [12]) as the format for the policy assertions. Subsequently, the user provides a resource with his CAS certificate. The resource parses the CAS policy assertions to determine the operations the user is allowed to perform.

CAS is a centralized third-party authorization solution. It reduces administrative costs and makes it easy to enforce coherent authorization policies in a community. Also, it offers scalability in terms of number of users and resource providers. Each user needs to be known and trusted by the CAS server, but not by each resource provider. Similarly, each resource provider needs to be known and trusted by the CAS server, but not by each user [39]. Nevertheless, CAS system is hard to install and configure. Moreover, because CAS issues new certificate that is different from GSI certificate, user account mapping may become complicated. Using a single CAS server may not be very scalable in terms of the number of resource access requests. If many users attempt to access the CAS server at the same time, it can be a bottleneck as well as a single point of failure.

VOMS

A Virtual Organization Membership Service (VOMS [16]) was initially proposed to provide authorization means on the DataGrid [4] project. In VOMS, group memberships and group rights are managed separately. The VOMS server only manages group memberships and group rights are maintained at the resource site. VOMS issues policy assertions to a user that contains a list of role or group memberships held by the user. Consequently, the resource provider is responsible for interpreting this assertion and determining the user’s access rights based on the user’s memberships and local policies about those memberships.

The first operation that a user must perform in order to access grid resources is to generate a proxy certificate. The VOMS generates a proxy certificate for the user.

The issued proxy certificate contains the user's role and memberships as a non-critical extension. A resource provider extracts authorization assertions from the user's proxy certificate and combines it with the site's local policies to make the authorization decision. The resource provider relies on two additional services to make the authorization decision: The Local Center Authorization System (LCAS) and Local Credential Mapping Service (LCMAPS). LCAS is a service employed on the resource site to enforce local security policies, while LCMAPS maps user to local credentials such as Unix account.

One advantage of the VOMS system is that it uses standard proxy certificate with the addition of a non-critical extension for the authorization information. Therefore, VOMS issued certificates can be used in non-VOMS-enabled services, which simply ignore the extra data. This also simplifies the user account mapping process as the resource provider can identify the user identity via his standard proxy certificate. Nevertheless, VOMS does not provide a complete centralized authorization solution (resource providers need to associate group memberships with resource permissions). This can potentially promote consistency problems in situations where policies are changing dynamically [38].

2.3.2 Pull Model

In the pull model, a subject directly contacts a resource. Subsequently, the resource contacts its Authorization Authority to determine the user's access rights. The Authority performs an authorization decision and returns the result to the resource. Based on the result, the resource grants or denies the service to the subject.

AKENTI

Akenti [42] is an authorization infrastructure developed at Lawrence Berkeley National Laboratory. It is an established authorization system that enforces fine-grained access control policies in communities with distributed resources controlled by multiple stakeholders. It presumes that X. 509 certificates are used for identification and authentication of all the entities involved in authorization. In Akenti, a resource can have multiple stakeholders possibly from different domains that independently define authorization policy for the resource. A resource policy is represented by a set of certificates digitally signed by the resource stakeholders. These certificates are usually distributed across multiple sites and can be stored remotely. To make an authorization decision, policy engine gathers all the relevant certificates for the user and the resource, verifies them, and determines the user's rights with respect to the resource.

Akenti uses XML to express authorization policy for a resource. The policy is stored in three types of certificates: use-condition certificates, attribute certificates, and policy certificates [1]. Use-condition certificates contain conditions that limit the access to a resource as well a list of rights/privileges that are granted if the conditions are satisfied. Attribute certificates associate attributes to users that are needed to satisfy the use constraints. A policy certificate contains the name of the resource to which the policy applies. In addition, it has a list of URLs to search for use-condition certificates. Policy certificate are self-signed and must be protected in a secure place. The other two certificates can be stored at different sites as their signatures will be evaluated whenever they are used. When a user attempts to access a resource, the resource gatekeeper contacts an Akenti server to find out if the user is allowed to perform the intended operation on the resource. The Akenti server retrieves relevant certificates and verifies that each of them is signed by appropriate issuer. Subsequently, it evaluates the certificates and returns the authorization decision.

Akenti supports decentralized and distributed authorization policy to provide fine-grained authorization solution that closely controls what users can do in a gird. However, because authorization information is stored in several distributed digitally signed certificates, some access management operations are difficult to perform. In particular it is not easy to list all the users and resources of the virtual organization or to check on all the outstanding authorizations for a given resource [20].

PERMIS

Privilege and Role Management Infrastructure Standards Validation (PERMIS [22]) is a Role Based Access Control (RBAC[23]) authorization system. A User's permissions in a community are derived from roles that are assigned to that user by a site administrator. PERMIS uses certificates to store authorization information such as user identity, roles, and authorization policy. These certificates are dispersed over many sites. In order to make an authorization decision, the PERMIS's authorization engine needs to collect and verify all user's certificates and to evaluate them against the resource's local access policy.

The PERMIS authorization policy includes a Role Allocation Policy (RAP) and a Target Access Policy (TAP). RAP specifies which managers are trusted to assign which roles to which users. At the same time, TAP states which roles are authorized to perform which actions on which resources. Moreover, the application gateway comprises an Application dependent Enforcement Function (AEF) and an Application independent Decision Function (ADF). A user contacts the AEF component of a resource. The AEF checks the user's attribute certificates against the RAP and valid attributes are passed to the ADF. According to authorization policy described by the TAP, the ADP makes an access right decision and returns it back the AEF. The AEF then enforces this decision on behalf of the resource.

Since number of roles in a community is far less than number of users, PERMIS role based authorization system significantly reduces the access control management cost and promotes scalability. However, in the case of multiple communities in a federation, PERMIS authorization system has a limitation. Because each community has its own attribute certificate repository, it is difficult to maintain consistency among them [27].

2.4 Authentication & Authorization in GAMA 2.0

The subsequent chapters describe GAMA 2.0's authentication and authorization schemes in detail. However, as this chapter presents many authentication and authorization models, it is worthwhile to mention that GAMA 2.0 uses the certificate based approach to authenticate users. Moreover, GAMA 2.0 developers have designed and implemented a pull model authorization system. Similar to PERMIS, it is a role based access control system that provides a complete and flexible authorization solution.

Chapter 3

GAMA 2.0 Scenario Analysis

To better understand GAMA 2.0 infrastructure, several usage scenarios are presented in this chapter. These scenarios were derived from GAMA 2.0's architecture document [32].

3.1 GAMA 2.0 Overview

An overview of a GAMA 2.0 system is shown in Figure 3.1 where the entire system represents one administrative domain. GAMA 2.0 server provides the means for centralized authentication and authorization, while the system users are identified by short-lived identification tokes such as X.509 proxy certificates. Upon request, the GAMA 2.0 server issues these certificates after a successful credential challenge: the user provides the GAMA 2.0 server with the correct shared secret (password). Moreover, end-users or external applications usually interact with the GAMA via web front-end, mainly a portal solution. The GAMA 2.0 provides the same authentication mechanism for all portals (i.e. portals that access GAMA 2.0 directly or through a separate single sing-on (SSO) portal). After a successful authentication, a user can use his session certificate to access various domain resources such as databases, files, computation grids, and other resources through the portals.

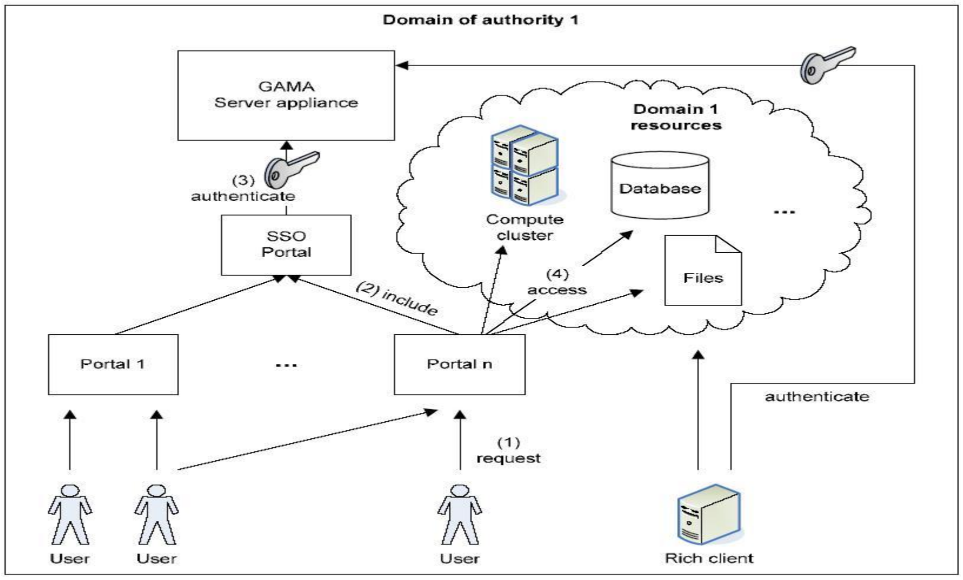


Figure 3.1: GAMA overview (adopted from [32])

3.2 GAMA 2.0 Single-site Authorization Enforcement

Authorization enforcement in GAMA 2.0 is shown in Figure 3.2. Assuming that a user has been successfully authenticated and obtained a short-lived certificate, authorization policy enforcement is performed upon accessing a resource. There exists two main authorization enforcement approaches:

- push mode: in this case, The GAMA 2.0 server issues a certificate which contains the digitally signed access permissions of a user.
- pull mode: in this case, upon a authorization request the resource, using the user's certificate, contacts the GAMA 2.0 server to obtain the user's authorization permissions.

In both cases, the GAMA 2.0 server knows about the users, resources as well as permissions granted to users, and stores this information in its authorization database.

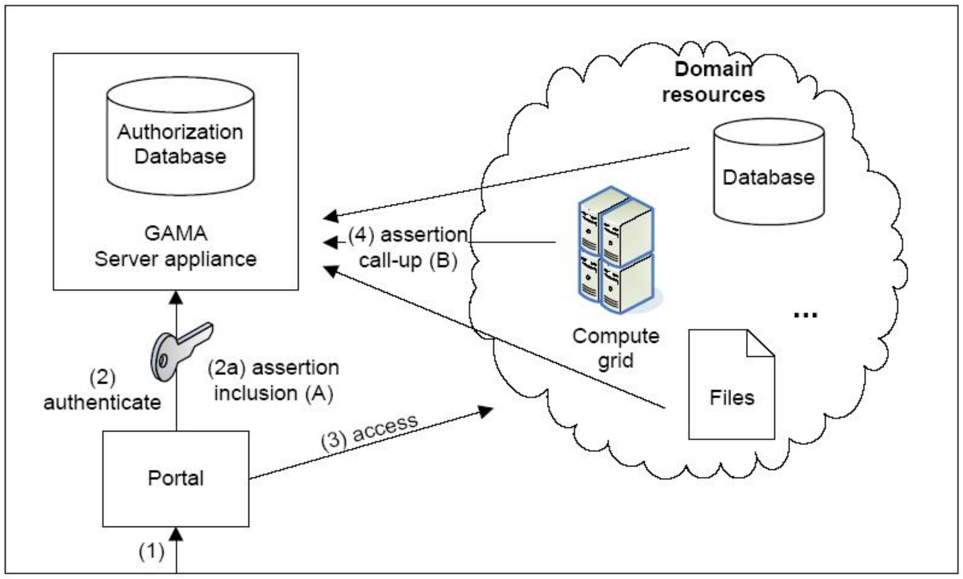


Figure 3.2: GAMA single-site access control (adopted from [32])

3.3 GAMA 2.0 Multi-site Authorization Enforcement

Figure 3.3 represent a situation where one GAMA 2.0 server is shared among multiple administrative domains, sites. Sites can be organizations or universities that work together on a collaboration project, while sharing a few common infrastructures such as a GAMA 2.0 server. Each site has a set of resources and users. In addition, it has an administrator that is responsible for managing site's resources and users. A site administrator has various responsibilities such as approving user self-registration requests and granting permissions to users. Centralized GAMA 2.0 server provides means for cross-site resource access. Similar to sites, the GAMA 2.0 infrastructure has an administrator that manages the overall GAMA 2.0 installation; however, he cannot mange users or sites belonging to a specific site.

3.4 Federated System Environment

Though it is not a common GAMA 2.0 case, Figure 3.4 shows an overview of a distributed identity management scenario which requires a complicated and specialized installation setup. In this scenario multiple GAMA 2.0 installations exist. The simple

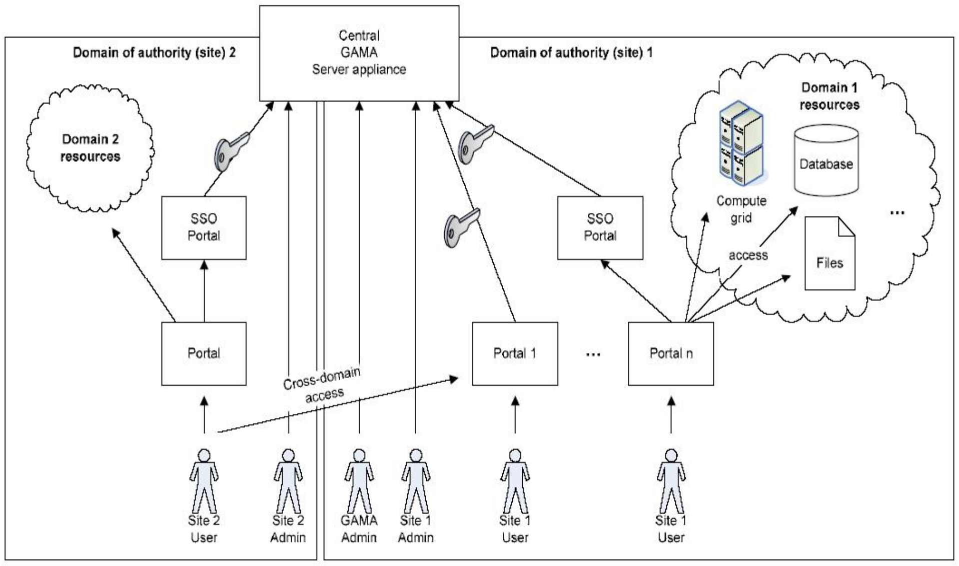


Figure 3.3: GAMA multi-site access control (adopted from [32])

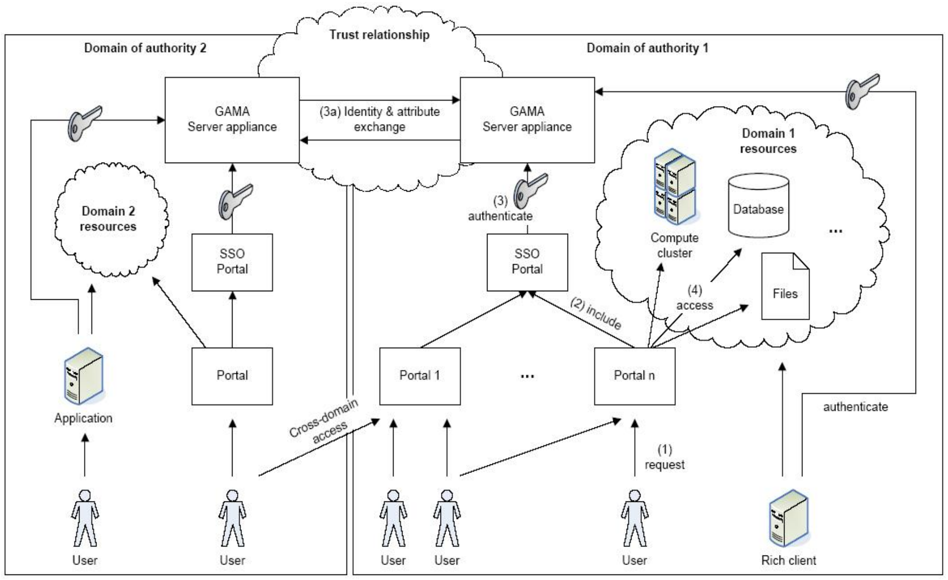


Figure 3.4: GAMA Federated system environment (adopted from [32])

case is similar to the section 3.3 scenario where users from one domain access resources in another domain. In this particular scenario, however, the two domains of interest are connected to two different GAMA infrastructures. Therefore, identity relationship needs to be established across systems by connecting the two GAMA 2.0 installations, based on a general trust relationship.

Chapter 4

GAMA 2.0 Security Requirements

Security is a vital component of distributed systems due to heterogeneous nature of resources and users. Providing a security solution for distributed systems such as grids is a complex problem which should be based on existing standards whenever possible. GAMA 2.0 identity management solution has been developed using security technologies that have stood the tests of time and repeated scrutiny. This chapter describes the Grid security requirements that GAMA 2.0 satisfies. They can be divided into project specific and generic requirements..

4.1 Generic Requirements

This section describes GAMA 2.0's generic requirements [43].

Satisfied Genetic Requirements

- Single Sign-On: users must be able to authenticate just once and then have access to any resource in the grid that they are authorized to use, without further authentication of the user

- Delegation: grid services should be allowed to act in a user's name i.e. a resource should be able to access other resources on user's behalf. Chain of delegation also should be possible.

- User-based Trust Relationships: if a user is capable to access resources in multiple domains, the intervention and interaction of site administrators should not be required in the security environment. For example, if a user has the right to use resources in sites A and B, the user should be able to access sites A and B without requiring administrators of site A and B to cooperate and interact.
- Authorization by Stakeholders: resource owners should be able to decide which subjects can access the resource, and under what conditions.

4.2 Project Specific Requirements

In addition to the genetic requirements, a complete grid security solution should address many project specific requirements. One of the main goals of GAMA is to develop a flexible architecture that can be applied to many projects. Therefore, especial effort has been made to harness requirements that many non-overlapping, academic-related projects share. By frequent requirements elicitation sessions with GAMA users and inference from the user statements in discussions, the GAMA team has discovered, categorized, and prioritized many common requirements [33]. NEES [9], CAMERA [3] and BIRN [2] were among the active participants in the requirements gathering process, although they have significantly different goals. CAMERA, for instance, aims to provide rich data repository and a bioinformatics tools to serve the needs of the microbial ecology research community, while BIRN project's goal is to create a distributed virtual community of shared resources offering tremendous potential to advance the diagnosis and treatment of disease. A short summery of the common security requirements that these projects share is presented here:

- Most users should not need to install middleware components on their desktop or laptop machines. It is desirable to allow users to access the grid resources via web or grid portal without installing any grid software on their machine. Moreover, users would like to use the familiar username/password model for logging into the system. They do not want to configure various middleware such as MyProxy, CAS, etc. similarly, users do not want to create and manage their certificates. Only in rare unusual cases, users would like to have direct access to their actual certificate

- For a given project, there are usually several types of resources such as clusters, databases, and domain-specific applications. Moreover, for each type of resource, there are multiple instances distributed in different administrative domains. Access to all project resources by a user should be enabled with a single username/password pair identifying that user.
- It would be desirable to avoid local account and gridmap updates as users are added or deleted from a project. In addition, role-based authorization mechanism should be employed to provide access control based on the identity of a user and the role to which the user is assigned (i.e. authorization for resource access is done through a user's role).
- The grid security solution should support broad range of resources, including web-based applications such as web portals; rich clients such as standalone applications; and the popular command-line tools such as ssh.

Chapter 5

GAMA 2.0 Architecture

5.1 Introduction

This chapter summaries the GAMA 2.0 architecture document [32] in a DoDAF-based format, while focuses on the author contributions to the design. In particular, it highlights its security models, library services, technology selection process, and uses cases describing important interactions between end-users and the system. Please refer to Chapter 3 for usage scenarios and workflows.

The rest of the chapter is organized as follows. First, we present the logical architecture of the system depicting operational components, system interfaces, and needlines between nodes. This is followed by a description of essential GAMA-lib services. Then we discuss logical data models that describe the GAMA's authentication and authorization policies in detail. We supplement this with a section on architecture deployment, highlighting selected technologies and tools. Finally, we end this chapter by showing a few user cases describing essential interactions between end users and GAMA.

5.2 Logical Architecture

5.2.1 Overview

Figure 5.1 displays the logical architecture of GAMA. It consists of operational nodes, communication needlines, and system interfaces. Operational nodes correspond to the logical components of the system, and communicate with each other via communication channels that carry certain type of information. Also, public interfaces allow end users to interact with the system. The main parts of GAMA are the server appliance as well as several web applications and middleware plug-ins for accessing the GAMA server.

5.2.2 Operational Nodes

A detailed description of the GAMA operational nodes is presented in this section. The operational nodes can be divided into server appliance components and nodes that are not part of the GAMA server, but belong to the GAMA framework.

GAMA Server Appliance Nodes

GAMA server appliance nodes consist of the following components:

- **Statistics Database:** it contains information about access statistics and server infrastructure. For instance, it could store the number of registered users as well as usage statistics of sites.
- **User Profile Repository:** this repository stores information specific to users and their profiles, but not their credentials and identities. For example, it may contain the email address and telephone number of a user, but not his public/private key.
- **Resource & Authorization Directory:** in a structured way, it keeps and retrieves information about resources, user roles, and user access rights. the resource & authorization directory will likely contain resource information such as resource

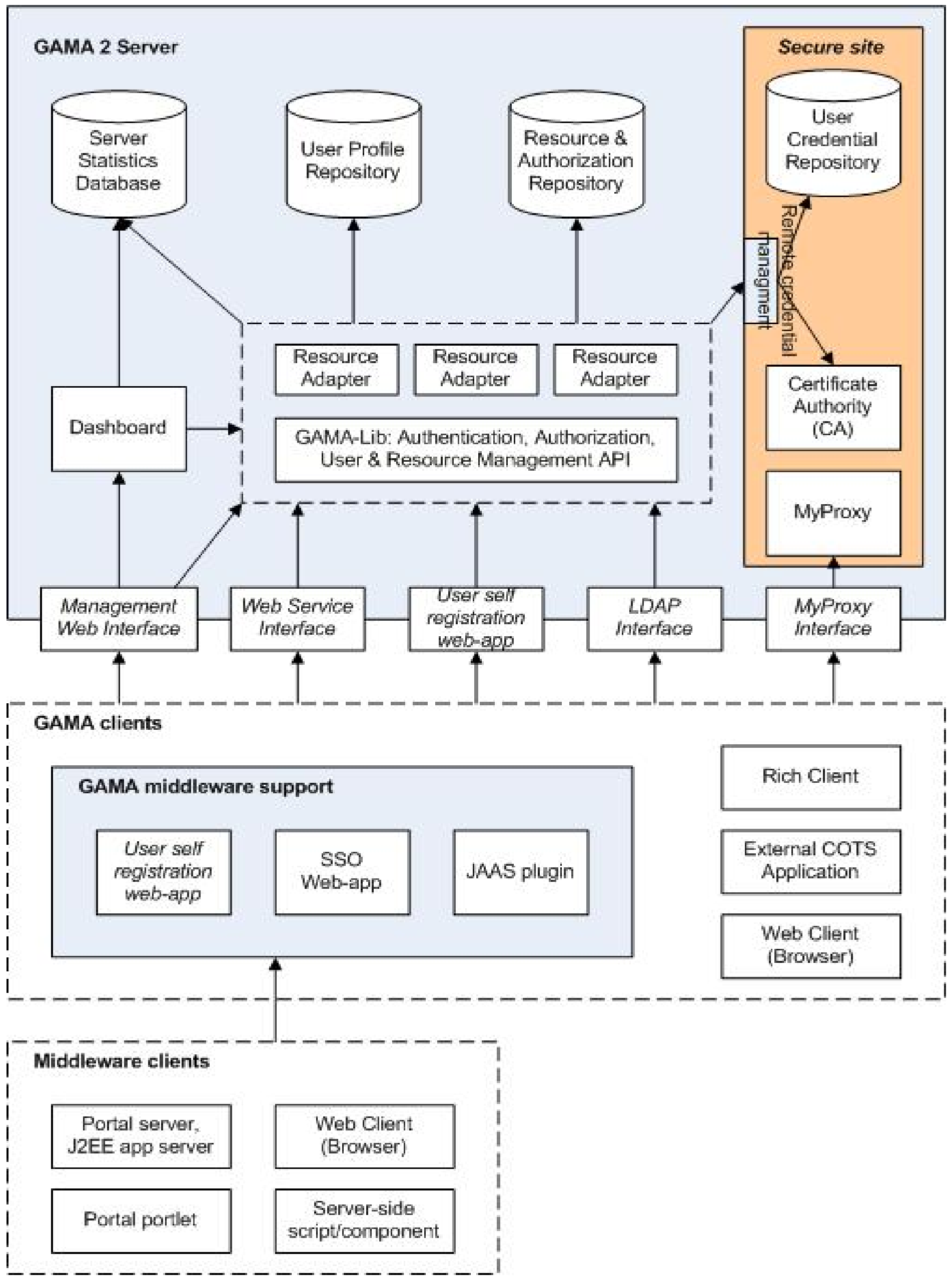

GAMA 2 Server
Secure site
Server Statistics Database
User Profile Repository
Resource & Authorization Repository
User Credential Repository
Remote credential management
Resource Adapter
Resource Adapter
Resource Adapter
GAMA-Lib: Authentication, Authorization, User & Resource Management API
Dashboard
Certificate Authority (CA)
MyProxy
Management Web Interface
Web Service Interface
User self registration web-app
LDAP Interface
MyProxy Interface
GAMA clients
GAMA middleware support
User self registration web-app
SSO Web-app
JAAS plugin
Rich Client
External COTS Application
Web Client (Browser)
Middleware clients
Portal server, J2EE app server
Web Client (Browser)
Portal portlet
Server-side script/component

Figure 5.1: GAMA Logical Architecture (adopted from [32])

names and a list of legal permissions on resources such as read, write, execute, etc.

- **User Credential Repository:** it keeps user identity and credential information such as private and public keys. For instance. it many contain X.509 credential certificates. The credential information will be used for user authentication and resource authorization.

- **Dashboard:** it aggregates and presents statistical data about GAMA server appliance. In addition, it provides a user interface to perform GAMA management functions.
- **Certificate Authority (CA):** CA is a trusted third party authority that provides digital certificate services such as issuing and signing certificates.
- **GAMA-lib:** the GAMA-lib provides functionalities for user authentication as well as resource authorization. It offers functionalities for user registration and user/resource management. It enables access to user profiles, and resource and authorization information, while it provides certificates for in-session use throughout the domain of authority.
- **MyProxy:** the purpose of the legacy MyProxy installation is to create short-lived proxy certificates to support legacy applications.
- **Resource Adaptor:** it allows the GAMA-lib to communicate with underlying nodes. It abstracts away the technology decisions that are made in the lower-level components so that GAMA-lib interface is not aware of those technologies.
- **Remote Credential Management:** it provides a single interface for credential management activities. It communicates with the user credential repository as well as Certificate Authority (CA) to process identity management requests.

Non-GAMA Server Appliance Nodes

Non-GAMA server appliance nodes consist of the following components:

- **User Self Registration Web-application:** this web-app provides an interface for user self-registration.
- **SSO Web-application:** it is a web-app component that provides a Single-Sign-On solution. Similar to the registration web-app, it will be deployed on a JavaEE

servlet container and provides many portals with the same authentication mechanism. By employing SSO, users can be authenticated once to gain access to resources that they have access permission for.

- **JAAS Plug-in:** the Java Authentication and Authorization Service (JAAS) implements a Java version of the standard Pluggable Authentication Module (PAM) framework. JAAS authentication is performed in a pluggable fashion which allows applications to remain independent from underlying authentication technologies

5.2.3 Public Interfaces

This section introduces the interfaces between GAMA components and the environment. The GAMA public interfaces allow end users and managers to interact with the system.

- **Management Web Interface:** the Management Web Interface exposes the management functionalities to the environment. It allows super users such as site administrators to interact with the GAMA server.
- **Web-service Interface:** user authentication, management, authorization functions exposed as web-services for access by external applications or users.
- **User Self Registration Web-application:** it provides a web interface for user self-registration.
- **LDAP Interface:** it provides LDAP communication protocol for integration with external applications. Many applications use LDAP for authentication purposes, and this interface allows these applications to interact with GAMA.
- **MyProxy Interface:** this interface allows MyProxy clients to interact with the legacy MyProxy installation.

5.3 Information Exchange

5.3.1 Needlines

Needlines are logical communication channels that carry a certain type of information. They also represent dependencies in the system as they are usually directed from one operational node to another. While Figure 5.1 captures all GAMA needlines, the main needlines are described below:

- **Management Web Interface to GAMA-lib:** it carries management requests to GAMA-lib, where they are processed. Management requests include resource management (e.g. add/remove resources) and user management (e.g. add/remove users) requests. Needlines connect the management interface to GAMA-lib, where these management requests are implemented.

- **Web-service Interface to GAMA-lib:** this needline carries end-user requests to the GAMA-lib, where they are processed. For example, an authentication request, which validates the identity of a user or an authorization request, which determines a user's access rights are transmitted to GAMA-lib via this needline.

- **User Self Registration Web-application to GAMA-lib:** while a user inputs required information in the user self registration web-app, GAMA-lib needs to process the data (i.e. add the user along with associated information into the database). This needline provides communication means to transfer user registration data to GAMA-lib.

- **LDAP Interface to GAMA-lib:** the LDAP interface allows LDAP-clients to interact with GAMA mainly for LDAP-based authentication purposes. This needline transfers these requests to GAMA-lib.

- **MyProxy Interface to MyProxy:** GAMA offers MyProxy and MyProxy interface to support legacy applications. The needline carries MyProxy requests such as issuing short proxy certificates.

- **GAMA-lib to User Profile Repository:** it allows GAMA-lib to store or retrieve user information. During user registration, for instance, the needline carries user profile information to be stored in the profile repository.
- **GAMA-lib to Resource&Authorization Directory:** a wide variety of information such as resource names, user roles, and access rights are stored in the Resource&Authorization directory. This information is required to process many requests like resource update or resource authorization. The GAMA-lib to Resource&Authorization Directory needline allows GAMA-lib to store/retrieve information about resources and access control in order to process a variety of requests.
- **GAMA-lib to Remote Credential Management:** this needline carries user identity management requests such as user authentication or renews myproxy certificate requests.
- **Remote Credential Management to User Credential Repository:** this needline enables access to user's credentials (e.g. certificates), which is require to process many requests like user authentication or resource authorization.
- **Remote Credential Management to Certificate Authority (CA):** it carries certificate-related requests to the CA such as issue/renew certificates.

5.4 GAMA-lib Services

This section summaries the essential services of the GAMA-lib. It is important to bear in mind that not all of the services will be made available to public as a few are designed for internal use. To provide a service, GAMA-lib typically needs to communicate and collaborate with several underlying components.

GAMA Server Management &Administration Services

Some of the important GAMA server administration services, which are usually managed by GAMA admin are captured below:

- Start/stop the GAMA server and all related tools and technologies.
- Configure the GAMA server such as setting tuning variables (e.g. database cache processing and indexing).
- Run server integrity tests to ensure GAMA is properly installed, and it communicates with relevant components.
- Create/Delete GAMA site entities such as GAMA projects or administrative domains.
- Register a user as a site administrator by, for example, granting the site administrator role to that user.
- Get usage statistics like number of users/resources. The data could be overall usage or site specific.

Resource Management Services

Some of the important resource management services, which are usually managed by resource manager are captured below:

- Define resources in the form of a resource directory tree. The context in which resources are created can be either a domain or another resource.
- Delete a resource along with all of the associated records from the GAMA server.
- Modify resource attributes (e.g. add/modify/delete resource attributes)

User Management Services

Some of the important user management services, which are usually managed by user or site manager are captured below:

- Create a site user in the GAMA server.
- Modify a user profile (e.g. add/modify/delete profile attributes)
- Retrieve user information and profile attributes.
- Delete a user along with all of the associated records from the GAMA server.
- Reset user password by administrator. The administrator decides what the new password should be.

Access Control Services

Some of the important access control services, which are usually managed by resource or site manager are captured below:

- Create/delete access control entities (i.e. capabilities or roles) in a given context (i.e. resource, site).
- Assign capabilities to roles (e.g. assign addUser capability to site administrator role).
- Grant/Deny access control entities like roles and capabilities to user entities (users/groups).

End-user Services

Some of the important end-user services are captured below:

- Create user through self-registration web-app and send email to the user with activation instruction.

- Authenticate user via shared password and issue a proxy certificate to calling service. A certificate represents a user in a current session.
- Change user password through shared secrets (e.g. old password, email address).
- Update a user profile (e.g. address, phone number, email address, etc.).

Resource Access Services

GAMA offers the following resource access service:

- Determine if a user has permission to perform the intended action on a resource.

5.5 Logical Data Model

This section describes the models that the GAMA system uses. Logical data models are essential to capture and fix the language and concepts of the architecture as they describe entities and their relationships that cover specific parts of the domain.

5.5.1 Users, Resources, Roles, and Policies

Dependencies between resources and policies are captured in Figure 5.2. It is organized in two main parts, which are separated by dashed boxes. The left hand side describes the entities related to user identity and authentication concepts, while the right hand side captures the entities related to access control and resource authorization concepts.

User Authentication:

Subjects could be users, external applications, agents, etc. The identity of a subject in GAMA is captured by the concept of principal. A principal is a system-defined identity that identifies a subject in a communication session. A subject can have multiple identities in different systems, and, thus, can have multiple principals. In addition, principals are identified by the number of credentials such as username-password pairs or smart-card identification. An authentication controller challenges a subject for correct credentials and relates a principal to the subject for authentication purposes. The GAMA authentication controller is an instance of a general authentication controller that issues short-lived proxy certificates that will serve as user identity within one domain of authority for the time of a session. Proxy certificates are signed by a trusted third party certificate authority, Since they are short-lived, they might be renewed for long running services.

Resource Authorization:

The basic entities of the authorization domain are resources, which need protection. In the authorization domain, resource is a general term that corresponds to a broad range of entities such as GAMA project, administrative site, file, database, web-service, etc. Resources can be either items or groups, where groups can contain further groups or items as composite. Resources are managed by resource providers. Also, Resources are associated with several capabilities, which are actions that can be performed on them such as read, write, delete, and more. The concept of granting or denying a capability in the context of a resource is called permission, which provides the means to reasonably limit the access to various resources. In order to perform a capability/action on a specific resource, the principal needs a granted permission. A common way to define permissions is through roles. Capabilities are assigned to roles which constitute the core of the authorization policy of the system. In addition, roles are granted/denied to principals subject to a specific resource. This means that the principal now has all permissions that are implied by the role. For instance, a “resource owner” role has various capabilities such as “create resource”, “delete resource”, etc. By granting a “resource owner” role to a principal in the scope of a resource, the principal gets all implied capabilities such as create resource or delete resource.

Resource providers enable subjects to access resources. They enforce access control restrictions using an authorization controller. After the authorization controller allows access, the resource provider performs the requested action on the resource. To make the access control decision, the authorization controller checks all permissions that are defined for the accessing principal. Moreover, it contacts the policy/rules engine

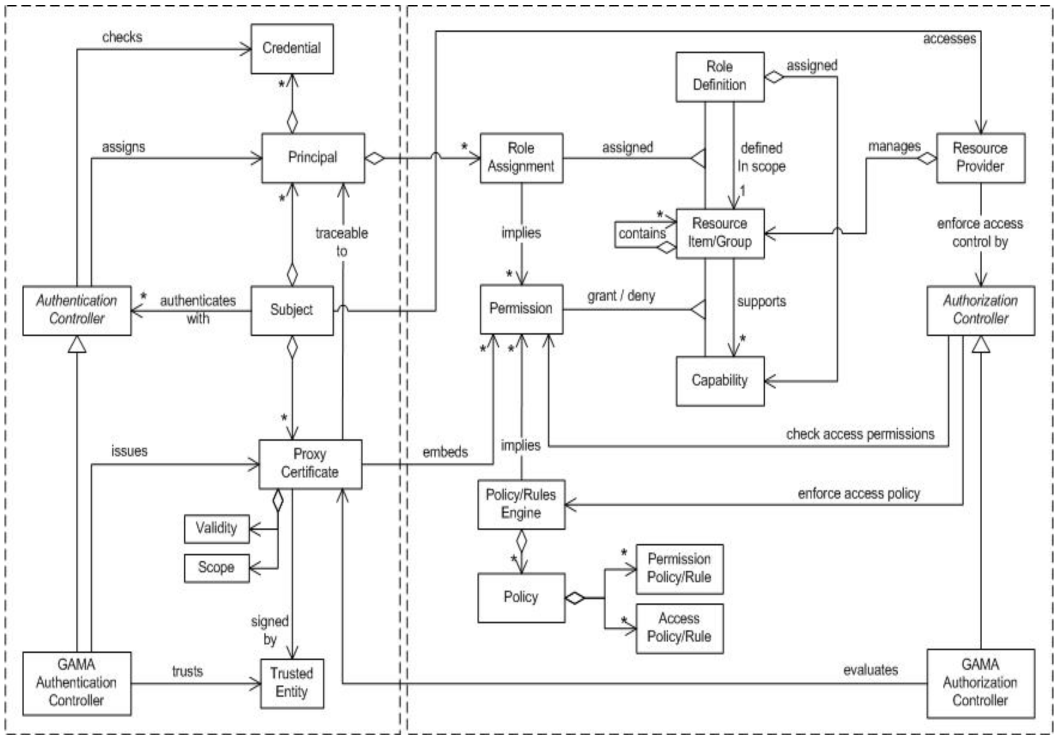


Figure 5.2: Resource-Policy Domain Model (adopted from [32])

to evaluate additional access control policies. For instance, an access policy could be "access is only permitted between 7am and 7pm." GAMA authorization controller, a specific authorization controller used for GAMA, enforces the authorization policy.

The link between the authentication and the authorization parts is twofold.

On the policy definition side, by assigning roles to principals, users have permissions to access resources. Specific permissions can be granted/denied on certain resources and permission policy rules can be defined. On the policy enforcement side, subjects are accessing resources via resource providers (for instance, by accessing a protected web URL or a web-service). Subjects carry a proxy certificate which directly or indirectly provides information about the access rights of the user. Push variant mode (directly): encodes a user's access rights in the certificate. Pull variant mode (indirectly): allows resources to use the identity from the proxy certificate to query the GAMA server for authorization permissions of users.

5.5.2 Security Data Model

While the previous section, 5.5.1, presented a high-level description of the GAMA logical data models, more detailed explanation of the GAMA security data model is presented here. Figure 5.3 captures the GAMA security policy by showing the logical entities and relationship between them. The GAMA security policy is based on role-based access control (RBAC) model, which was formalized in 1992 by David Ferraiolo and Rick Kuhn [23].

In our model, capability is simply an action that can be performed on a resource such as read, write, execute, etc. Capabilities are usually defined by administrators in a particular context. In GAMA, a context refers to a resource or a site. A resource can be either a resource item such as a grid cluster or a resource group which contains further nested resources. A site can be the GAMA root, project, administrative site, nested site, or etc. GAMA contexts are stored in a hierarchical tree-like structure. Therefore, the context of an entity also represents the scope of that entity. For example, if a capability is defined in the GAMA root, the entire tree structure is the scope of that capability and, hence, all the security entities have access to the capability. However, if a capability is defined in a GAMA project site, the scope of the capability is the sub-tree rooted at the project, so the GAMA root and other projects do not have access to the capability.

Similar to capabilities, roles are defined in a context by administrators. Likewise, the context of a role embodies the scope of it. Moreover, capabilities are assigned to roles; hence, roles group capabilities together to implement a Role-Based Access Control (RBAC) Model. For instance, a “manager” role might have capabilities such as create, delete, and write, while a “regular user” role might have capabilities like read and write.

Under the RBAC framework, users are assigned to roles based on their competencies and responsibilities. The capabilities (operations) that a user is permitted to perform are based on the user’s role. In other words, when users are assigned to roles, they inherit the capabilities granted to those roles. Therefore, there is no need to assign the same set of capabilities to each user when a role can be assigned. In addition to granting roles to users, it is possible to grant roles to user groups. A user group is simply a collection of users. By granting roles to user groups, all members of the group inherit all role's capabilities. Finally, user entities (user, user groups) are assigned to roles in a context, and the context serves as the scope of the assignment. For instance, a user can have the "site administrator" role in one context (e.g. calit2 site), while he has the "regular user" in another context (e.g. SDSC site).

Additional capabilities can be granted or denied to a user entity. Often fine grained control over authorization is required to manage exceptions to established roles. Users may inherit from several roles, yet a few capabilities granted by these roles may need to be denied for specific users. Conversely, the user may need additional capabilities that are not granted to any predefined roles that they are assigned to. Consequently, the GAMA security policy allows for direct grant/deny capabilities to user entities. As more users require these capability tweaks, administrators define new roles for managing policies rather than assigning individual capabilities to users. In addition, the GAMA security solution allows contexts to "acquire" security policies from their parent context in order to ease the burden of creating a security policy. Usually, GAMA contexts "acquire" their security policies from their parent context since it makes a given security policy easier to maintain. Only when there are exceptions to the "master" security policy, individual contexts are associated with a different security policy.

In essence, the GAMA security policy associates roles/capabilities to users in a context. In other words, it says "who" can do "what" and "where".

Role- Based Access Control (RBAC) Advantages:

Since GAMA's security policy is based on RBAC, it offers the same benefits as other RBAC solutions. David Ferraiolo and Rick Kuhn discussed the following advantages of RBAC security systems [23]:

- **Centrally Administering Security:** one important advantage of the GAMA security policy is flexible administrative capabilities. Once capabilities and roles are

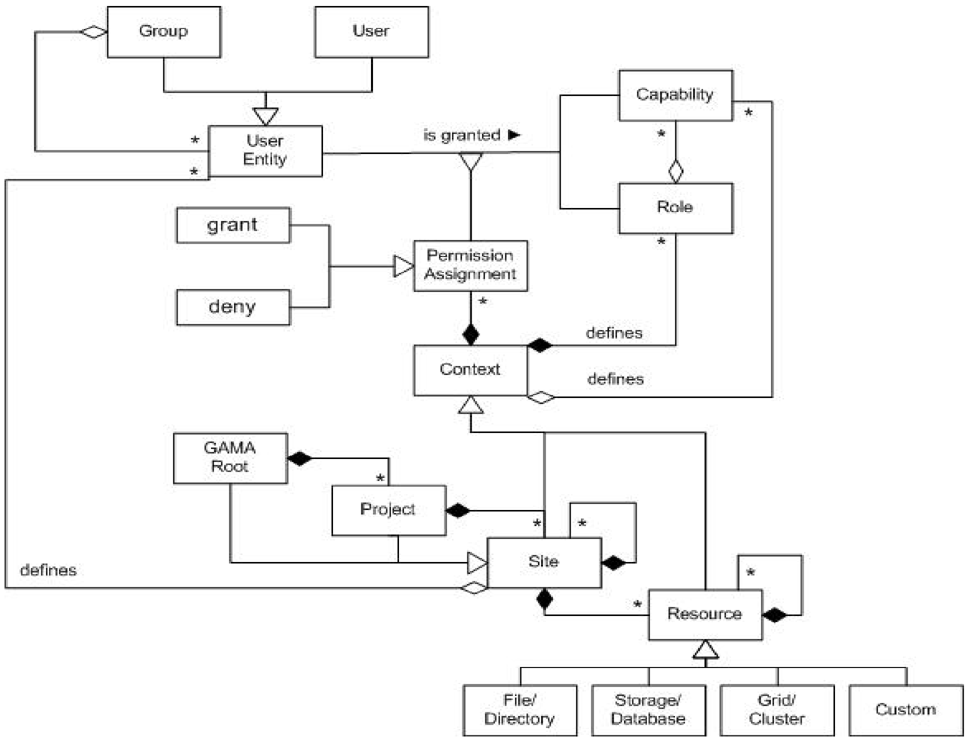


Figure 5.3: GAMA Security Data Model (adopted from [32])

defined in the system, they stay relatively constant and change slowly over time. The administrative task consists of granting/revoking roles/capabilities to users. When a new user enters the organization, the administrator simply grants existing roles/capabilities to the user. When the user's responsibilities in the organization change, the administrator revokes his existing roles/capabilities and simply grants him new ones. Finally, the "acquire" feature reduces the burden of maintaining security policies by allowing contexts to acquire their parent's policy.

- **Principle of Least Privilege:** the principle of least privilege suggests that users should be given minimum set of privileges required to complete a job. It consists of identifying the user's job, determining the minimum set of privileges required to perform that job, and restricting the user to a domain with those privileges only. Through the use of GAMA's RBAC, enforcing minimum privileges for system users can be easily achieved.

- **Separation of Duties:** separation of duties requires that, for particular sets of capabilities, no single individual be allowed to execute all capabilities within the set. An administrator can use the GAMA's RBAC to enforce a policy of separation of duties. One of the frequently used examples explaining separation of duties is the separate capabilities needed to initiate a payment and to authorize a payment. No single individual should be capable of executing both actions [23]. Considering this case in GAMA, an administrator is required to ensure that an individual who serves as payment initiator does not serve as payment authorizer. This could be implemented by ensuring that no one who can perform the initiator role can also perform the authorizer role.

5.6 GAMA Deployment Architecture

Section 5.2.1 captured the logical architecture of GAMA by describing operational nodes and needlines. It did not state any information about technologies and tools that will be used as system components and nodes. In this section, we present the deployment architecture of GAMA which depicts the mapping of the logical architecture to the physical environment. Figure 5.4 shows this mapping by displaying chosen technologies/tools and their relationships. Moreover, for the vital operational components, we describe the process and motivations for selecting their corresponding tools.

5.6.1 GAMA Technology Selection Process

The arguments for selecting some GAMA technologies and tools are explained here. We only describe the selection process of key technologies that play important role in providing GAMA vital services such as user authentication and resource authorization.

User, Resource, and Authorization Repository

The logical architecture diagram displays the repositories for user profiles, resource information, and authorization entities. One of our early technology selection

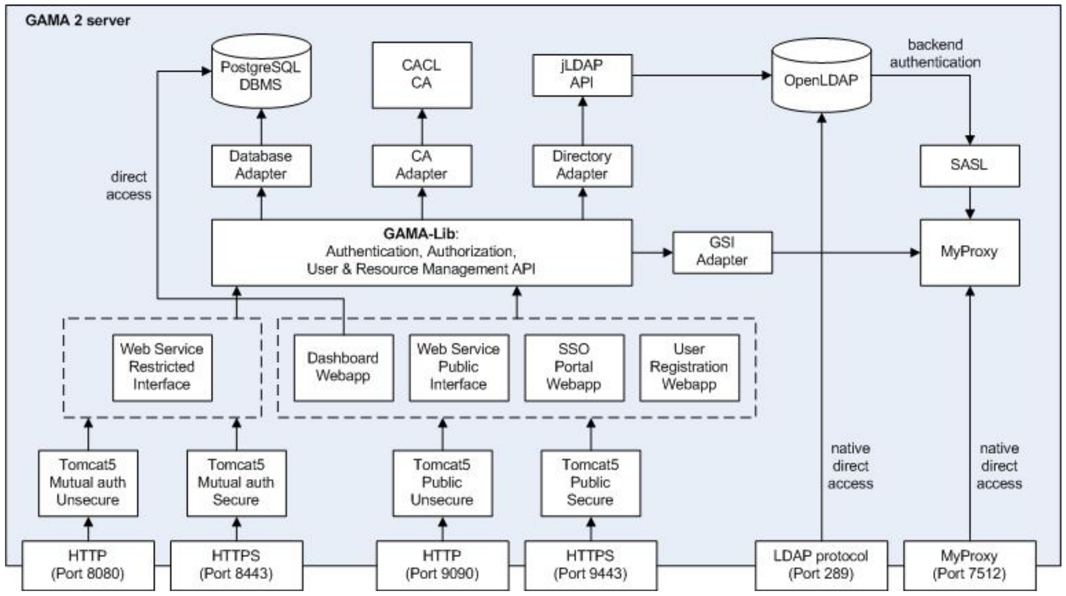


Figure 5.4: GAMA 2 Deployment Architecture (adopted from [32])

decisions was to determine what kind of storage mechanism to use to store this information. Considering a directory and a database as possible options, we have decided to store the relevant information in a directory for several reasons [44]. First, databases need to process various read as well as write requests, whereas directories support high volumes of read access. Therefore, directories are optimized for read requests. The information that will be stored in GAMA is relatively static and will not change rapidly. For instance, user profile data such as user name and email is read frequently but updated rarely. The popularity of read request in GAMA encourages employing a directory service to store related data. Another important advantage of directories over relational databases is the hierarchical nature of them. It is desirable to store GAMA information in a tree-like hierarchical structure. Users and resources among other entities are usually associated with a context such as a site, and a directory provides a repository to store this information in a hierarchical fashion.

Directories are usually accessed using a client/server model of communication. This is an advantage since it allows GAMA to collectively store user, resource, and authorization data in a separate machine. However, a well-defined message protocol is needed for accessing and updating information in a directory. The Lightweight

Directory Access Protocol (LDAP [44, 11]) defines a standard method for accessing and updating information in a directory. It has gained wide acceptance as a directory access method and is employed as a directory access protocol in GAMA. In particular, OpenLDAP [11] which is an open-source directory software suite supporting LDAP protocol is used in GAMA. OpenLDAP provides both a directory server and LDAP protocol in an out-of-the-box installation which minimizes the installation and configuration time. In addition, extensive online resources support and documentation allow new developers to quickly become familiar with the software. For example, OpenLDAP provides JLDAP [11] which is an API library for Java, enabling developers to easily access, manage, update, and search information stored in a directory.

In brief, GAMA uses OpenLDAP to collectively store user, resources, and authorization data in a single directory and to provide LDAP message protocol for accessing and managing the stored information.

Credential Management Technologies

The Certificate Authority

GAMA is a GSI-based security solution that relies on a third-party certificate authority (CA) to authenticate system entities. Although there are many certificate authorities, we have chosen CACL [30] as a certificate authority component in GAMA. In GAMA, CACL will issue and sign X.509-based user and host certificates. One of the advantages of CACL is that it is based on a client/server model. Therefore, it can run on an isolated locked-down machine with no end-user access, and its services are accessed solely through calls from web-services. In addition, CACL was developed in San Diego Supercomputer Center, and can be downloaded and used free of charge. Since CACL's production in 2000, it has been successfully running at SDSC. Another factor in selecting CACL is the fact that it was designed to be employed in grid computing environments. Therefore, CACL's development team has made significant effort to speed up and simplify the creation process and use of digital certificates [30]. Moreover, CACL was employed in the previous versions of GAMA and successfully stood the tests of various real world projects. Finally, since some members of the GAMA 2.0 development team were already familiar with CACL, using CACL to provide an implementation of a certificate authority was a reasonable choice.

The Online Credential Repository

Standard web security protocols employed between a web client and web server are not sufficient to address many security challenges in distributed systems like grids. For instance, they do not support delegation of access rights. GAMA has employed the MyProxy [36] tool to bridge the incompatibility gap between web and grid security. MyProxy is an online credential repository designed to enable grid portals to use the protected resources in a secure manner [36]. It is used in GAMA because of the several important advantages that it offers. Extensive online documentation and resources, free downloading and easy configuration among other reasons have encouraged the GAMA team to use the MyProxy tool in the new version of GAMA. Moreover, MyProxy's unique features such as scalability, delegation support, and credential renewal can greatly improve a user's interface to grid security and reduce the complexity of grids [36]. Another reason to select MyProxy is its wide-acceptance in the grid portal community. Several projects such as National Computational Science Alliance, NPACI, NASA Information Power Grid, and the previous version of GAMA have employed MyProxy. The feedback from these projects has been positive suggesting that Myproxy can be successfully deployed in real-world projects.

5.7 GAMA Use Cases

This section presents key use cases that show essential interactions between end users and the GAMA system. In particular, the use cases are presented in sequence-diagram format describing user authentication and resource authorization. Furthermore, these use cases describe user registration process as well as the administrators' usage of the system.

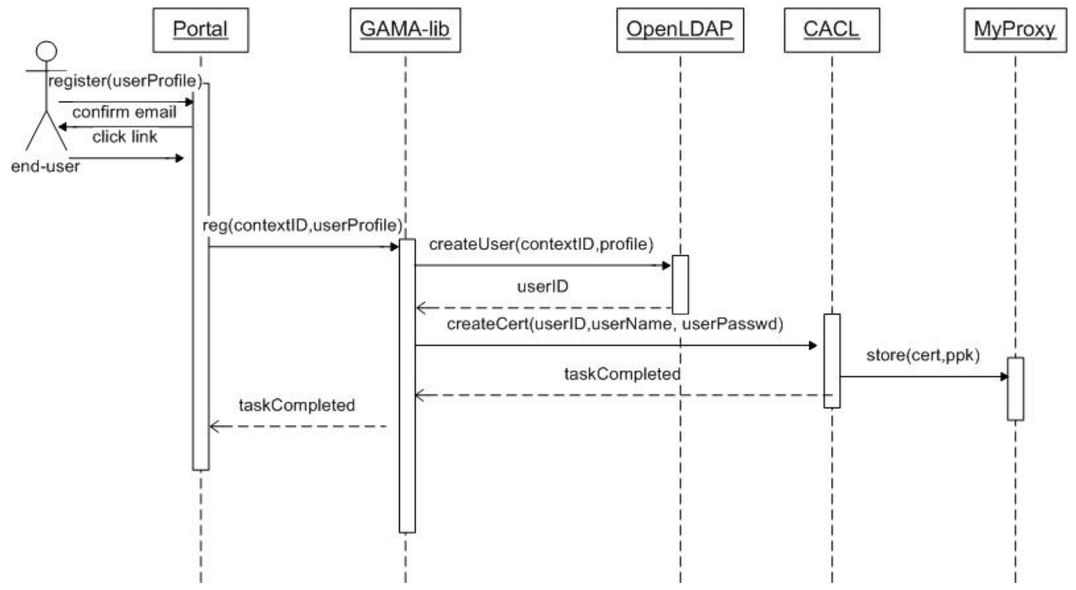


Figure 5.5: User Registration Use Case

User Registration

To register, users access the registration web portal to complete their profile as it is suggested by Figure 5.5. A user profile typically consists of several user-related information such as user name, last name, email, login name, password and more. Once the front-end web portal validates a user's email address, the user's profile along with the user's site id are passed to the GAMA-lib. The GAMA-lib stores the user's profile in LDAP tree, and then calls CACL to generate a long-lived certificate for that user. In addition to making a public-private key pair and a long-lived user certificate, CACL encrypts the user's private key with the supplied user password for authentication purposes (see user authentication use case). Furthermore, CACL sends the user's certificate and public-private keys to MyProxy for future user authentication and short-lived certificate generation.

User Authentication

Figure 5.6 captures interactions made for user authentication. To login, a user accesses the login web portal to input his login name and password. Then the GAMA-

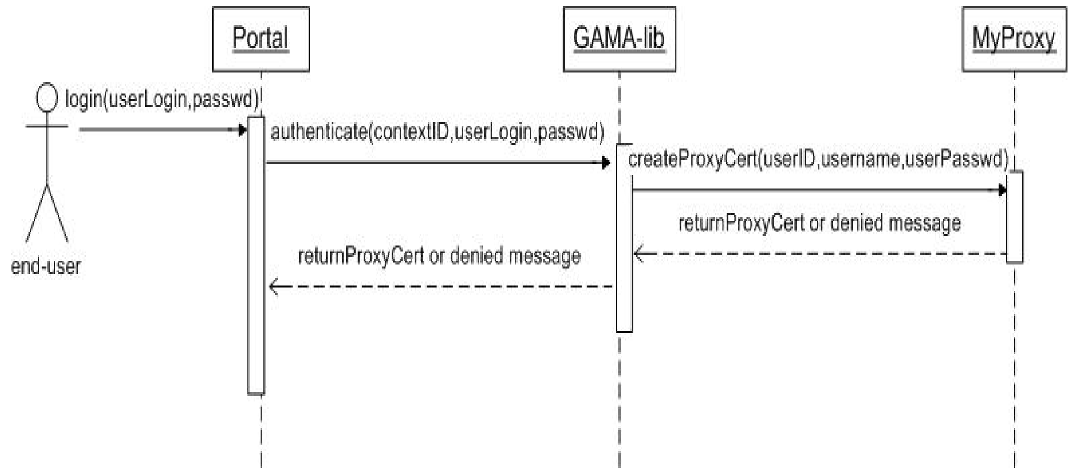

Portal
GAMA-lib
MyProxy
login(userLogin,passwd)
authenticate(contextID,userLogin,passwd)
createProxyCert(userID,username,userPasswd)
returnProxyCert or denied message
returnProxyCert or denied message
end-user

Figure 5.6: User Authentication Use Case

lib's user authentication service is called to determine if the user is authenticated to the site that he is trying to login. Subsequently, the user's credentials such as user name and password are sent to MyProxy to generate a short-lived certificate. Before issuing the proxy certificate, MyProxy attempts to decrypt the user's private key with the user's password. If successful, short-lived proxy certificate is issued and returned to the portal. Otherwise, a denial error message is sent back.

Resource Authorization

Users usually access a resource to perform an action such as read, write, execute, etc. As the Figure 5.7 suggests, the user's proxy certificate is presented to a resource. Then the resource calls the check permission service of the GAMA-lib which is available via web-services. It supplies the service with userID, resourceID, and capabilityID (identifying the user's intended action). With the help of OpenLDAP, the GAMA-lib identifies user's roles and capabilities, and determines if the user can perform the intended action. The result is sent back to the portal.

Administrator Usage

Figure 5.8 shows how administrators interact with GAMA. An administrator uses the admin web-application to interact with the system. After a successful login into

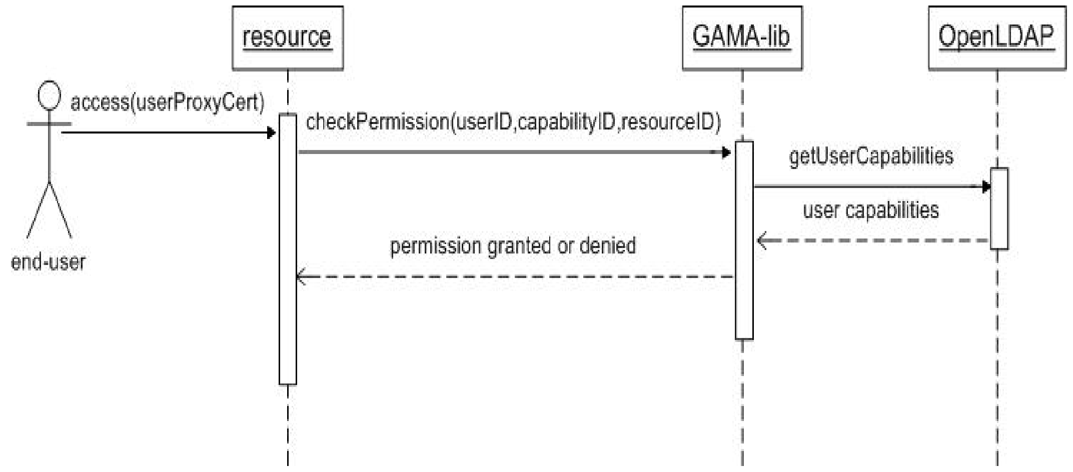


Figure 5.7: Resource Authorization Use Case

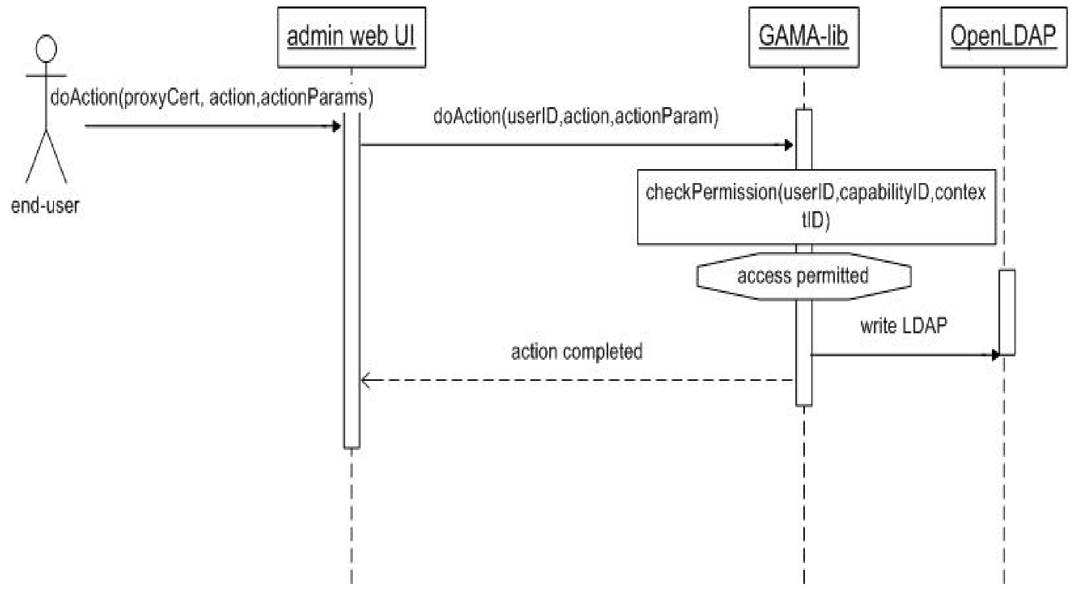

admin web UI
GAMA-lib
OpenLDAP
doAction(proxyCert, action,actionParams)
doAction(userID,action,actionParam)
checkPermission(userID,capabilityID,contextID)
access permitted
write LDAP
action completed
end-user

Figure 5.8: Administrator Usage Use Case

the system, an administrator can perform several actions like add users, add sites, delete resources, etc. The administrator's doAction request which contains the administrator's ID, action type (like addResource), and the action parameters (such as resourceName) is passed to GAMA-lib where it can be processed. The GAMA-lib, first, ensures that the administrator is “authorized” (i.e. has the capability) to perform the intended action. Then it executes the action which usually results in updating the LDAP directory.

Chapter 6

GAMA 2.0 Implementation

6.1 Introduction

This chapter discusses the implementation of the GAMA's internal alpha release. The author of this document had a significant contribution in the implementation phase. This contribution includes installing and configuring many relevant tools such as OpenLDAP. In addition, the author implemented the majority of the gamaLib services using many well-known programming techniques and patterns. Then he applied various testing techniques such as unit testing and scenario testing to examine the gamaLib source code. Finally, the author of this thesis designed and implemented a detailed exception handling solution to adequately report unrecoverable errors to end-users. While he was not involved in designing and implementing the user interface, a summary of the front-end implementation is presented in this chapter.

The rest of the chapter is organized as follows: first, we present the related tools and technologies that were used in implementing GAMA such as testing framework. Then we discuss the organization and structure of the source code. This is followed by a section describing the implementation details of the alpha release. We supplement this with a section on exception handling suggesting how exceptions are processed in GAMA. Then we discuss GAMA's testing strategies. Finally, we end this chapter by briefly talking about implementation security as well as discussing some open issues related to the alpha release.

6.2 Technology Selection

Development Languages

The GAMA 2.0 development team selected Java as the primary programming language to implement the architecture described in the previous chapter. Java offers many key features that are valuable in developing GAMA 2.0. For instance, it is designed to make distributed computing easy with networking capabilities that are inherently integrated into it [13]. In addition, security plays an important role in its design as it promotes writing extendable and easy-to-maintain code. Because the underlying technologies such as LDAP and MyProxy are Java based, and the development team has extensive Java experience, Java has been selected to implement the back-end of the GAMA 2.0 architecture. The GAMA front-end, however, is implemented in Groovy. Groovy is a scripting language employed in the GAMA 2.0 front-end implementation since it integrates well with Java, while it offers many additional benefits such as closures and dynamic typing [7].

Build Tools

Ant and Maven are two most popular build tools, and both are used in developing the GAMA 2.0. Since the author of this work was the primary developer implementing the GAMA 2.0 back-end and he was already familiar with Ant, he decided to use Ant to compile and run the back-end code. Maven, however, has been employed in developing the front-end and integrating it to the back-end code. Maven provides the key features of Ant, while it makes project management tasks easy, which is crucial in the integration process.

Version Control System

Since GAMA 2.0 is a collaborative project, a version control system is required to manage and monitor the code development process. We have selected SVN as our version control system since it has been successfully used in many projects and is supported by extensive online resources and documentations. In addition, SVN is open source and in comparison to other subversion alternatives like CVS, SVN offers great improvements such as efficient branching and tagging as well as detailed versioning.

Development Environment

We were provided with a Linux-based server for development purposes. The GAMA development team was excited about the server because of two main reasons. First, it was a spare server, so the development team could freely change its configuration, install or remove various tools, and add or delete users. In addition, a few key components of GAMA such as MyProxy were already installed on the server.

To expedite the development process, the author decided to use Eclipse framework to implement the GAMA 2.0 back-end. Eclipse offers several key features such as automatic indentation, powerful refactoring, and static analysis tools. Moreover, it has a flexible debugger that speeds up the debugging and testing process.

Deployment Framework

The back-end GAMA server requires various services and software packages such as CACL, MyProxy, web-service libraries, and more. These technologies are usually difficult to install and a few of them require a significant amount of configuration after installation. To alleviate the installation and deployment burden on the administrator, we have employed the Rocks clusters management tool [37]. Rocks installs various GAMA components with minimum system administrator intervention. In addition, it performs many operations such as configuring various tools, deploying web-services, and starting the necessary services using startup scripts. In short, the Rocks clusters tool is employed to provide a fully functional GAMA back-end server running out of the box with minimal effort on the part of the system administrator [18].

Logging Services

Log4j is a popular Java logging utility which is employed in implementing GAMA 2.0. Perhaps, the main advantage of Log4j is that logging behavior can be controlled by editing a configuration file, without modifying the application code. Another distinctive feature of Log4j is the notion of inheritance that allows inserting logging statements at arbitrary granularity. These advantages, augmented with Log4j's easy installation and usage, encouraged employing Log4j as the logging service.

Testing Framework

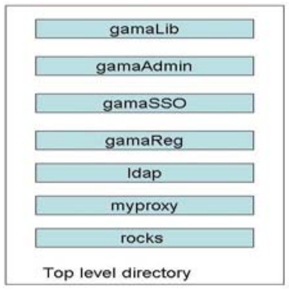
We strive to make GAMA 2.0 defect free to the maximum extent possible by testing it under various conditions. JUnit, the de facto standard unit testing library for Java, has been chosen as the testing framework in GAMA 2.0. In particular, JUnit 4 promises to simplify testing by exploiting Java 5's annotation feature to identify tests [8]. In GAMA, we have employed various testing techniques such as unit, functional, and scenario testing using the JUnit 4 framework.

6.3 Code Organization

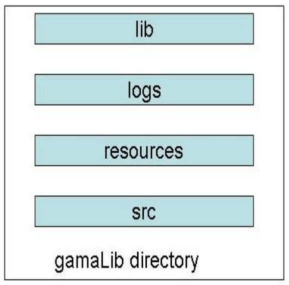
This section describes the organization of GAMA 2.0 code. It shows how various GAMA 2.0 packages are structured based upon the multiple tiers of GAMA 2.0's architecture. Moreover, it focuses on the back-end code structure which is one of the author's main contributions.

Top Level Directory

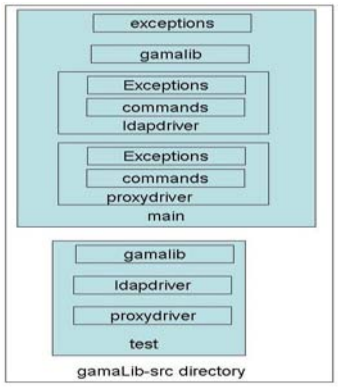
Figure 6.1(a) depicts the top level directory which contains the following main components:



(a) Top Level Directory



(b) gamaLib Directory



(c) gamaLib-src Directory

Figure 6.1: GAMA 2.0 Code Organization

- GamaLib: a Java API for accessing and manipulating the account data. The core back-end implementation resides here.
- GamaAdmin: web-application for administering the gama appliance. It contains the code for administrator interfaces.
- GamaSSO: web-application for single sign-on. The GamaSSO is a web-application that provides login component.
- GamaReg: web-application for user registration.
- Ldap: various files related to LDAP configuration.
- Myproxy: various files related to myproxy configuration.
- Rocks: GAMAv2 appliance as a rocks roll. It bundles all GAMA technologies to provide an out-of-the-box solution.

GamaLib Directory

The author's main contribution was implementing the gamalib which has the following main directories as it is suggested in Figure 6.1(b):

- Lib: contains a wide range of libraries that are used in developing the back-end such as log4j, junit, and LDAP libraries.
- Logs: contains log statements saved as files.
- resources: contains various configuration files such as the log4j configuration file.
- src: contains the back-end implementation code.

Gamalib-src Directory

Figure 6.1(c) describes the gamaLib-src directory which is made of two sub-directories: main and test. The following directories reside in the main directory:

- Exceptions: contains all exceptions that gamaLib may throw.
- Gamalib: has the implementation of a wide range of GAMA services such as resource and user management.
- Ldapdriver: Contains driver code that connects to the LDAP server and performs relevant operations such as add, delete, and more. It is made of two sub-directories. Command directory contains the LDAP-related commands and the exception directory has GAMA LDAP exceptions.
- Proxydriver: the structure of the proxydriver directory is similar to the ldapdriver directory. Namely, there are two important sub-directories, the command directory, which has the myproxy-related command and the exception directory, which contains the myproxy-related exceptions.

Test sub-directory is another important part of the gamalib. It contains the following:

- Gamalib: test cases that target gamalib services.

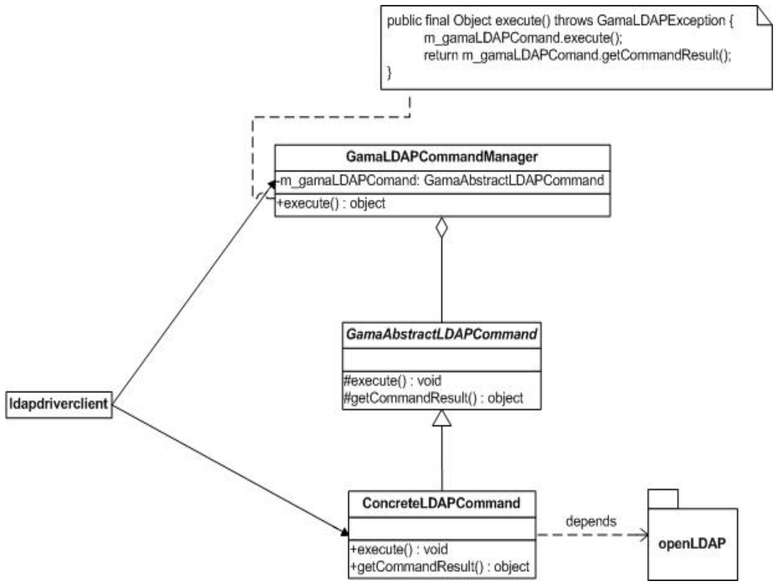
- Ldapdriver: test cases that examine the ldapdriver implementation.
- Proxydriver: proxydriver test cases.

6.4 Code Implementation

6.4.1 Drivers

In designing and implementing GAMA 2.0, we have strived to make it pluggable and extendable. As a partial fulfillment of this requirement, we designed and implemented two drivers; ldapdriver and proxydriver. The ldapdriver hides the underlying repository technology, OpenLDAP, which is employed to store user and resource information from the gamalib. At the same time, the proxydriver shields the gamalib from the online credential management service that GAMA 2.0 uses: MyProxy utility. As core technologies change rapidly, it is pivotal to make gamalib independent from them to the maximum extend possible. By hiding gamalib from low-level components, drivers promote weak coupling which results in reducing the complexity of the system.

The drivers connect to the intended servers (i.e. OpenLDAP and MyProxy servers) and perform specific operations. By considering various operations like addUser, addResource as commands, the Command design pattern is employed to implement the drivers. The Command pattern provides well-desired loose coupling since it decouples the object that invokes the operation from the one that knows how to perform it. This allows gamalib to treats commands as “black box” without knowing how they are implemented. This provides a logical separation of duties and makes code easier to maintain. Furthermore, the Command pattern offers other benefits. For instance, it expedites debugging as logging commands in execution suggests the state in which an error has occurred. Moreover, by considering commands as first class objects, the Command design pattern allows commands to be manipulated and extended like any other object. At the same time, adding new commands is easy since it does not require modifying the existing commands.



public final Object execute() throws GamaLDAPException {
m_gamaLDAPComand.execute();
return m_gamaLDAPComand.getCommandResult();
}
GamaLDAPCommandManager
-m_gamaLDAPComand: GamaAbstractLDAPCommand
+execute() : object
GamaAbstractLDAPCommand
#execute() : void
#getCommandResult() : object
ldapdriverclient
ConcreteLDAPCommand
+execute() : void
+getCommandResult() : object
depends
openLDAP

Figure 6.2: ldapdriver Command pattern

Ldapdriver

Figure 6.2 shows the high-level structure of the Command pattern in ldapdriver. A typical usage scenario is as follows: an ldapdriver client like gamalib instantiates a concrete LDAP command such as addUser. Then it calls the command manager with the command. The command manager or the invoker is responsible for executing the command and subsequently returning the result to the client, gamalib. The benefits of the Command pattern, discussed above, are present here. For instance, as seen in Figure 6.2, gamalib is not aware of OpenLDAP which is the underlying data repository.

Ldapdriver Commands

While the previous subsection discussed the overview of the Command pattern in ldapdriver, here we present many ldapdriver commands and discuss how they are implemented. Figure 6.3 shows a simplified snapshot of how commands are structured in ldapdriver. In a nutshell, we have divided the commands into two categories, GamaEntryCommand and GamaAttributeCommand. Commands that target entries in the LDAP directory are in the GamaEntryCommand group, while commands that mod-

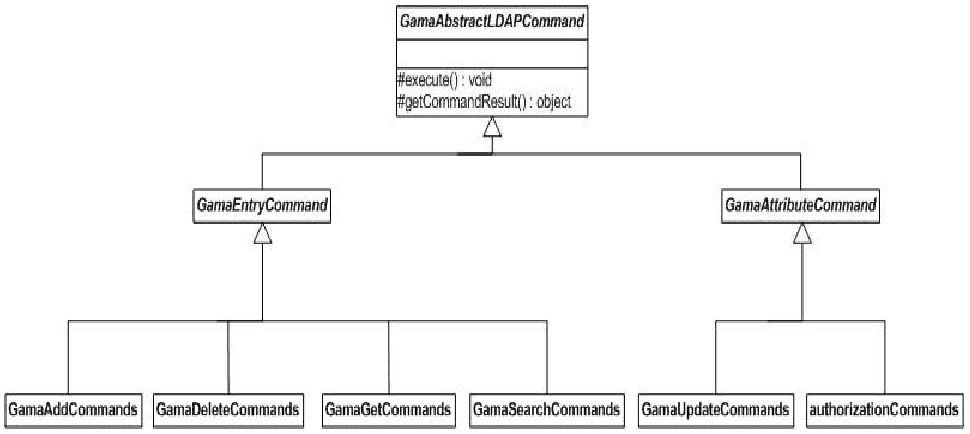


Figure 6.3: ldapdriver commands

ify attributes of a particular entry belong to the GamaAttributeCommand category. addUser, deleteResource, and searchSite are among many commands that target specific entries in the LDAP directory. For instance, addUser creates a new entry corresponding to the new user, while deleteResource deletes a resource entry from the directory tree. Nevertheless, commands like updateUserAttribute and addGroupMember modify attributes of existing entries; therefore, they reside in the GamaAttributeCommand group. Because commands are structured in a hierarchical fashion, it is easily possible to maintain and extend existing commands as well as add new ones when desired.

GamaEntryCommand

Here, we discuss a few crucial entry-related commands.

- GamaAddCommands consist of several commands that add entries to the LDAP tree. For example, GamaAddResourceEntryCommand adds a resource along with its associated attributes to the LDAP directory. Each GamaAddCommands requires a siteDN as parameter which uniquely identifies the parent site in which the entity will be added.
- GamaDeleteCommands consist of several commands that delete entries from the LDAP tree. For example, GamaDeleteResourceEntryCommand deletes a

resource along with its associated attributes from the LDAP directory. Each GamaDeleteCommands requires an entryDN as parameter which uniquely identifies the entry that will be deleted.

- GamaGetCommands consist of several commands that return all attributes of an entry. For example, GamaGetResourceAttributesCommand returns all resource attributes. Each GamaGetCommands requires an entryDN as a parameter, which uniquely identifies the entry that its attributes will be returned as a map of attribute names to values.

- GamaSearchCommands consist of several commands that search the LDAP tree. For example, GamaGetSiteResourcesCommand takes a siteDN as a parameter, searches for resources in the site, and returns them as a list of entries.

GamaAttributeCommand

Several commands that modify attributes of existing entries are presented below:

- GamaUpdateCommands consist of commands that update some of the attributes of an entry. A map structure is used to hold new attribute values and the map is passed to commands as a parameter. Also, each GamaUpdateCommands requires an entryDN as a parameter which uniquely identifies the entry that its attributes will modify. GamaUpdateUserAttributesCommand, for instance, might update a user's personal information like his/her telephone number or address.

Authorization Commands

Ldapdriver has many commands that collectively implement the GAMA security policy mentioned in chapter 5. Although describing all authorization-related commands is beyond the scope of this report, a summary of key commands is discussed here.

- GamaAdd/RemoveCommands: ldapdriver has commands to add/remove a capability to/from a role. For example, GamaAddCapabilityToRoleCommand takes two parameters (capabilityDN, roleDN), and adds the new capability to the role's capability list. Using this command, for example, we can add deleteCapability to managerRole. GamaDeleteCapabilityFromRoleCommand, on the other hand, removes a capability form the role's capability list. Similar to these commands, GamaAddUserToGroup and GamaDeleteUserFromGroup adds/deletes a user to/from a group respectively.

- GamaGrant/RevokeCommands: GamaGrantCommands grant a role or capability to a user in a particular context (scope). For instance, GamaGranCapability-Command may grant executeCapability to a user in the scope of a resource. Similarly, GamaGrantRoleCommand may grant managerRole to a user in the context of a site. In addition, GamaRevokeCommands revoke a role or capability from a user in a context. GamaRevokeCapabilityCommand can revoke executeCapability from a user in the scope of a resource, while GamaRevokeRoleCommand may revoke mangerRole from a user in a context like site.

- GamaCheckPermissionCommand: this perhaps is one of the critical commands in ldapdriver that implements resource authorization. It takes three arguments (capabilityDN, contextDN, userDN) and it determines if the user carries the desired capability in a given context. For instance, upon a write request, a database calls this command using its own DN, writeDN, and e userDN. Then GamaCheckPermissionCommand searches the LDAP tree to identify the user's group memberships, roles, capabilities in the appropriate scope. Using this information, GamaCheck-PermissionCommand derives a list of user capabilities and determines if the desired capability is in this list.

Proxydriver

The author of this thesis was not involved in implementing the proxydriver, although similar to the ldapdriver, it uses the Command design pattern. More specifically, GamaProxyCommandManager is responsible for executing MyProxy commands such as createUserCertificate or authenticateUser.

Gamalib

Section 5.4 presents a set of services that GAMA 2.0 provides to public. GamaLib implements many services using popular programming principles such as inheritance and polymorphism. While the previous section discussed the significance as well as the usage of drivers in gamaLib, here we focus on the gamaLib implementation. GamaLib is composed of four interfaces that provide GAMA 2.0 services as well as many additional classes that help implement them. Before discussing the interfaces and their implementation, we summarize two most important components of gamalib: the GAMA entity identification system and the entity attribute management system. These two components are frequently employed in implementing the services.

GAMA Entity Identification System

Figure 6.4 depicts the gamaLib's entity identification system. In gamaLib, entities such as users, resources, roles are uniquely identified by their ID. As Figure 6.4 suggests, a context can be either a site or a resource, while a user entity object can be a user or a group.

The entity identification system simply reflects the naming mechanism of the underlying data repository without explicitly relying on its schema. Therefore, changing the back-end's repository or its schema will not affect the gamaLib identification system. Since OpenLDAP is used as the back-end repository, an entity ID in gamaLib corresponds to an entry's distinguished name (DN) in the OpenLDAP directory tree.

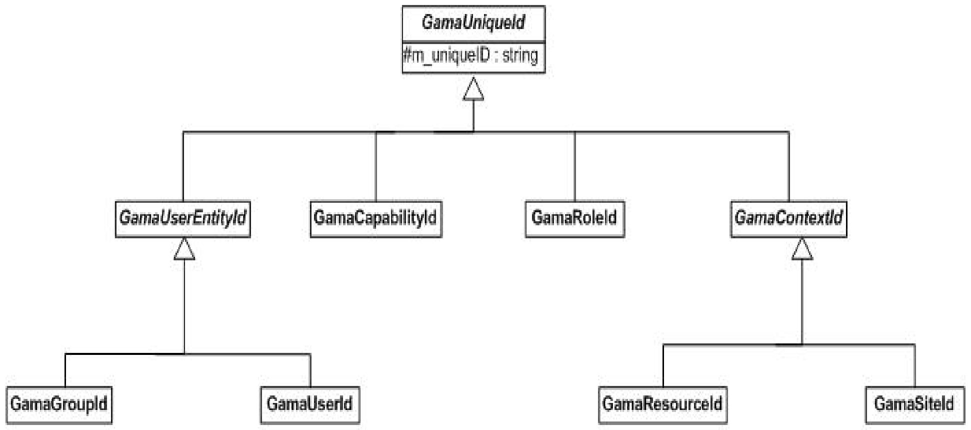


Figure 6.4: GAMA Entity Identification System

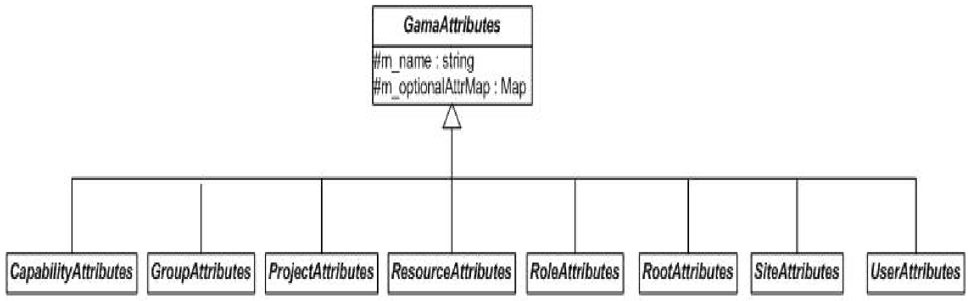


Figure 6.5: Entity Attribute Management System

Entity Attribute Management System

While the entity identification system uniquely identifies gamaLib entities, the entity attribute management system, simplified and shown in Figure 6.5, represents attributes associated with an entity. It also provides a mapping from gamaLib entity attributes to OpenLDAP entry attributes. For example, GamaUserId uniquely identifies a user entity in gamaLib, while GamaUserAttributes contains the user's attributes such as user name, site, and address. Moreover, GamaUserAttributes relates these attributes such as "site" to LDAP attributes like "OU" (organizational unit).

GamaLib Services

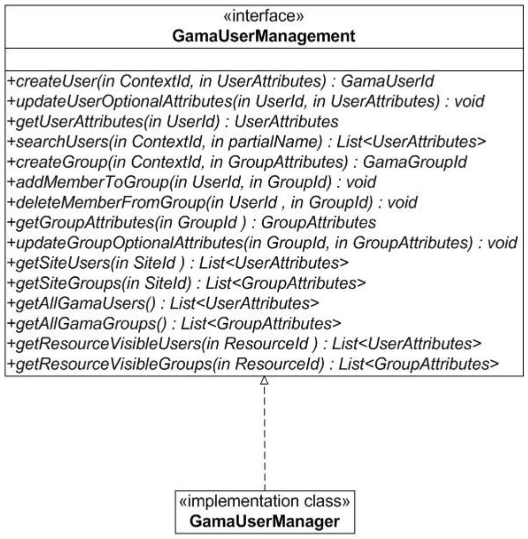
There are four interfaces that collectively list all gamaLib services that have been implemented so far: GamaUserManagement, GamaResourceManagement, GamaAuthorizationManagement, and GamaAuthenticationManagement. In this section, we summarize these interfaces and discuss what kind of services each of them offers.

- GamaUserManagement: as the name suggests, this interface presents user-related services. Figure 6.6(a) shows the services that have been successfully implemented. For instance, createUser takes contextID and userAttributes as parameters then calls proxydriver to generate a user certificate and ldapdriver to store user attributes. It returns the userID, which subsequently can be used in other services like getUserAttributes.

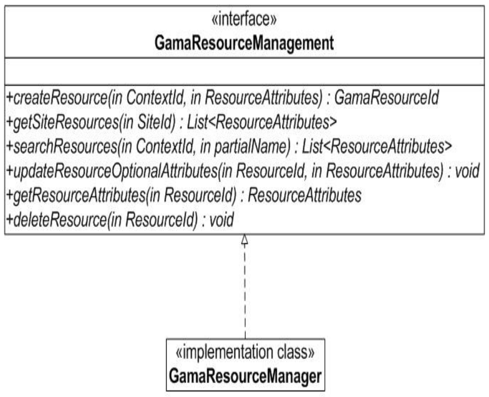
- GamaResourceManagement: there are several resource-related services. GetSiteResources, for example, takes a siteID as the parameter and returns a list of resources that belong to the site. Figure 6.6(b) depicts the resource-related services.

- GamaAuthorizationManagement: a wide variety of authorization services are listed in Figure 6.6(c). There are services for managing authorization entities like createRole and deleteCapability. In addition, there are services for relating these entities to users and groups like grantCalability, revokeRoleFromUser. Finally, there are services for determining users' capabilities like getUserCapabilities and checkPermission.

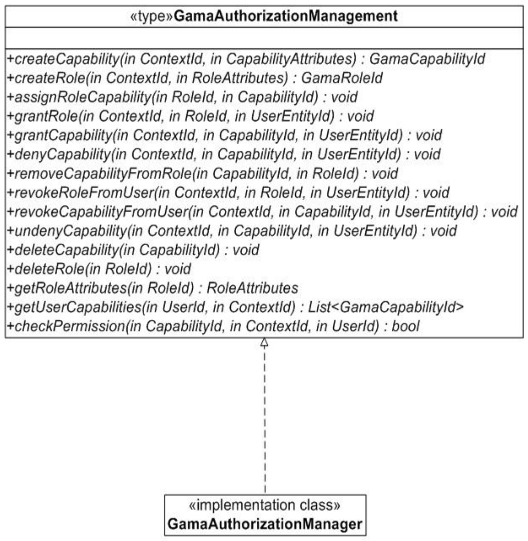
- GamaAuthenticationManagement: Figure 6.6(d) shows the authentication services. As of now, there is only one authentication service, authenticateUser, which calls proxydirver with user's login and password and returns a short-lived proxy certificate.


«interface»
GamaUserManagement
+createUser(in ContextId, in UserAttributes) : GamaUserId
+updateUserOptionalAttributes(in UserId, in UserAttributes) : void
+getUserAttributes(in UserId) : UserAttributes
+searchUsers(in ContextId, in partialName) : List<UserAttributes>
+createGroup(in ContextId, in GroupAttributes) : GamaGroupId
+addMemberToGroup(in UserId, in GroupId) : void
+deleteMemberFromGroup(in UserId , in GroupId) : void
+getGroupAttributes(in GroupId) : GroupAttributes
+updateGroupOptionalAttributes(in GroupId, in GroupAttributes) : void
+getSiteUsers(in SiteId) : List<UserAttributes>
+getSiteGroups(in SiteId) : List<GroupAttributes>
+getAllGamaUsers() : List<UserAttributes>
+getAllGamaGroups() : List<GroupAttributes>
+getResourceVisibleUsers(in ResourceId) : List<UserAttributes>
+getResourceVisibleGroups(in ResourceId) : List<GroupAttributes>
«implementation class»
GamaUserManager

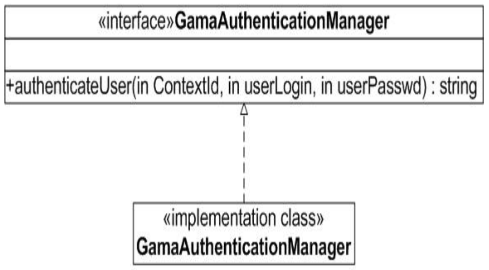
(a) gamalib user-related services


«interface»
GamaResourceManagement
+createResource(in ContextId, in ResourceAttributes) : GamaResourceId
+getSiteResources(in SiteId) : List<ResourceAttributes>
+searchResources(in ContextId, in partialName) : List<ResourceAttributes>
+updateResourceOptionalAttributes(in ResourceId, in ResourceAttributes) : void
+getResourceAttributes(in ResourceId) : ResourceAttributes
+deleteResource(in ResourceId) : void
«implementation class»
GamaResourceManager

(b) gamalib resource-related services


«type»GamaAuthorizationManagement
+createCapability(in ContextId, in CapabilityAttributes) : GamaCapabilityId
+createRole(in ContextId, in RoleAttributes) : GamaRoleId
+assignRoleCapability(in RoleId, in CapabilityId) : void
+grantRole(in ContextId, in RoleId, in UserEntityId) : void
+grantCapability(in ContextId, in CapabilityId, in UserEntityId) : void
+denyCapability(in ContextId, in CapabilityId, in UserEntityId) : void
+removeCapabilityFromRole(in CapabilityId, in RoleId) : void
+revokeRoleFromUser(in ContextId, in RoleId, in UserEntityId) : void
+revokeCapabilityFromUser(in ContextId, in CapabilityId, in UserEntityId) : void
+undenyCapability(in ContextId, in CapabilityId, in UserEntityId) : void
+deleteCapability(in CapabilityId) : void
+deleteRole(in RoleId) : void
+getRoleAttributes(in RoleId) : RoleAttributes
+getUserCapabilities(in UserId, in ContextId) : List<GamaCapabilityId>
+checkPermission(in CapabilityId, in ContextId, in UserId) : bool
«implementation class»
GamaAuthorizationManager

(c) gamalib authorization-related services


«interface»GamaAuthenticationManager
+authenticateUser(in ContextId, in userLogin, in userPasswd) : string
«implementation class»
GamaAuthenticationManager

(d) gamalib authentication-related services

Figure 6.6: gamaLib Services

Front-end Web-applications and Web-services

Because the author did not contribute in designing and implementing the front-end interfaces, this section only presents a short description of them. The GAMA front-end consists of several web-applications that allow end-users to access GAMA 2.0's services. Specifically, management web-application provides a complete administration of the GAMA appliance which includes managing users, resources, sites etc. At the same time, user registration web-application allows users to request account on the associated GAMA appliance. Account requests are held until an administrator approves them. When a request is approved, an email is sent to the user with an activation link. Once the user clicks the link, verifies his email and selects a password, the web-application creates the account for the user by calling appropriate gamaLib services. Another major component of GAMA frond-end is the Single Sign-On web-application that allows users to authenticate once to access their resources.

Front-end web-applications and gamaLib are installed on the same machine. This allows the web-applications to easily access various gamaLib services. However, user authentication and resource authorization services, two essential gamaLib services, are exposed to public via web-services calls. This is an advantage as they can be called from environments and portals that run on different machines.

6.5 GAMA Exception Handling

During the GAMA 2.0 development process, we dedicated a significant amount of time designing and implementing a robust exception handling component. Realizing that it is important to adequately report unrecoverable errors to end-users as well as to resolve recoverable conditions without user's intervention, GAMA 2.0 offers a detailed, yet flexible, exception handling solution. Figure 6.7 shows a simplified, high level overview of GAMA's exception handling component. It strives to achieve two pivotal goals of GAMA 2.0 implementation, namely, technology independence of gamaLib as well as extendability. It achieves the first goal by wrapping technology-specific exceptions in more generic exceptions. More specifically, GAMALDAPExceptions abstract away LDAP exceptions, while GAMAProxyExceptions hide MyProxy exceptions. GamaLib uses these two types of exceptions to throw higher-level exceptions that are independent from underlying tools. The GAMA front-end, subse-

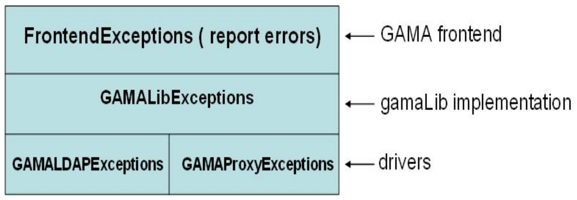


Figure 6.7: GAMA Exception Handling

quently, uses the gamaLib exceptions to report appropriate error messages to users. For example, when adding a user who already exists, ldapdriver throws GamaLDAPEntryAlreadyExistsException. GamaLib implementation, then, catches this exception and throws GamaEntryAlreadyExistsException which shields the fact that LDAP is the underlying data repository. The later exception can be used in the GAMA front-end to notify that the user already exists in GAMA. It is important to note that in addition to wrapping lower-level exceptions, gamaLib throws library-specific exceptions as well. Revisiting the user creation example, gamalib throws an exception if user's login or password is missing before the drivers are called.

We employ programming techniques such as the Factory pattern and polymorphism in developing the exception handling unit. Moreover, the exception handling solution is multi-layered corresponding to the levels of abstraction like drivers, gamaLib. Therefore, it is easily possible to extend, modify, and delete the existing exceptions as well as to add new ones.

6.6 GAMA Testing

The GAMA development team designed and implemented a comprehensive testing solution. Similar to the exception handling component, it is multi-layered corresponding to the levels of abstraction such as drivers and gamaLib. GamaLib test component examines the gamaLib code. Each testing module such as ldapdriverTests consists of many unit tests that individually examine small parts of source code. These unit tests collectively constitute scenarios that model the real use of the GAMA system. For example, the gamaLib test-component consists of several unit tests that target various gamaLib services such as addSite, addUser, checkPermission. Using the JUnit's testsuite feature, we have orderly bundled these test cases to create tests that examine gamaLib code based on hypothetical scenarios.

6.7 Implementation Security Discussion

Performing a comprehensive security analysis on GAMA 2.0 implementation is beyond the scope of this document. This section, nevertheless, briefly describes a few steps that mitigate the risk of malicious attacks.

Front-end web-applications, gamaLib services and drivers, as well as underlying repositories reside on a protected, locked down machine with no end-user access. Consequently, drivers (ldapdriver and proxydriver) can communicate with corresponding repositories in clear-text. Moreover, web portals can easily invoke many gamaLib services and these services do not need protection as they are not exposed to public (i.e. only portals have access to them).

Unlike other gamaLib services, user authentication and resource authorization services are exposed to the public as they are solidly accessed through web-services calls. To protect these services from malicious attacks such as snooping user's password, they run over a secure communication channel, https.

6.8 Alpha Release Open Issues

In this chapter, we presented a high-level overview of GAMA 2.0 implementation. It is important to bear in mind that the discussion given in this chapter corresponds to the internal alpha release. Although many core features of GAMA have been successfully implemented, there are still many open issues that need to be addressed before future releases. Here we discuss some of these issues:

- "Acquire" feature: we spent a tremendous amount of time designing and implementing the security policy and resource authorization. As mentioned before, there is a scope associated with any security-related assignment such as granting a role to a user. The scope is a sub-tree rooted at the context, where the assignment is created. Therefore, contexts inherit their parent's security policies. However, it would be nice to allow contexts to define new policies without "acquiring" any security policies from their parent contexts. This feature, although not implemented, promotes a more powerful and flexible security solution. This feature will be implemented as part of the beta release since the development team has decided to focus on testing, debugging, and refactoring the current implementation for the alpha release.

- GamaLib services access control: currently, gamaLib does not ensure that its clients have the adequate access rights to call the library's various services. It solidly relies on the front-end portals to call appropriate gamaLib's services. It might be beneficial to expose management services via web-services calls so they can be accessed from portals running on remote servers. In this case, only clients with sufficient access rights should be able to execute these services. One appealing solution to ensure that a service caller has the desired access right to call the service is to internally employ the GAMA's authorization mechanism. This can be done by obtaining DN of the caller from the current thread. Then using the caller's DN along with additional information, we can call GAMA's checkPermission service to ensure that the caller has appropriate capability to perform the indented action. Taking the addSite service as an example, checkPermission service can be called to make sure that the caller has the addSite capability before adding a site.

- Ldap-based authentication: LDAP is employed for authentication purposes in many projects. To make GAMA system appealing to as many projects as possible, we support LDAP-based authentication in GAMA as shown in Figure 5.4 As it can be seen from the Figure, OpenLDAP is connected to MyProxy via the SASL mechanism. This implies that LDAP-based projects can employ GAMA, even though MyProxy is ultimately used for authentication purposes. For the alpha release, the link between OpenLDAP and MyProxy is not implemented, so MyProxy is called directly for authentication-related requests. For the beta release, we plan to implement the link and fully support LDAP-based authentication.

Chapter 7

GAMA 2.0 Evaluation and Analysis

7.1 Introduction

This chapter evaluates design and implementation of GAMA 2.0. First, the author depicts a few advantages of GAMA 2.0 over previous versions of GAMA. GAMA 2.0 offers several new features and enhancements that were not included in GAMA 1.X such as resource authorization support. Then the author compares GAMA 2.0 to a couple of popular identity management solutions that were originally described in section 1.5. Moreover, performance of gamaLib has been evaluated in terms of execution time of various gamaLib services. The author describes the evaluation process and optimization techniques that have successfully reduced the execution time of many inefficient gamaLib services. Table 7.1 summarizes the advantages of GAMA 2.0 over GAMA 1.X and other related identity management solutions, which include support for resource authorization and LDAP-based user authentication. Moreover, unlike other solutions, GAMA 2.0 offers a service-oriented and pluggable architecture.

Table 7.1: GAMA 2.0 vs Related Identity Management Systems

	GAMA 2.0	GAMA 1.X	PURSE	GridAuth	FusionGrid
Service-oridented Architecture	YES	YES	NO	NO	NO
Pluggable Architecture	YES	NO	NO	YES	NO
Resource Authorization	YES	NO	NO	NO	YES
LDAP-based Authentication	YES	NO	NO	NO	NO
Simple User Interfaces	YES	YES	YES	YES	YES
Easy Deployment	YES	YES	NO	NO	NO

7.2 GAMA 2.0 vs GAMA 1.X

7.2.1 Authorization Support

GAMA 1.X has an option to use CAS [38] for resource authorization. CAS explicitly adds user permissions to a retrieved proxy which can subsequently be used for authorization purposes. It is up to the resources to use the user's proxy certificate to determine if he is authorized to perform intended operation based on internal policies. Although GAMA supports CAS, it was never proven to be useful and stable because not many end resources know how to use CAS authorization mechanism [18]. Therefore, communities that employ GAMA seek other solutions to fulfill their authorization requirements. CAMERA[3] project, for example, attempts to resolve GAMA 1.X's lack of authorization by granting the same access rights to all authenticated users. Therefore, after successful authentication, all users carry the same access rights and can perform similar operations. Because many communities have many users with different access rights, simply assigning equal authorization permissions to all community users is not sufficient. Often, communities require fine-grained authorization mechanisms that can support various users with different access rights. Consequently, these projects need to complement the GAMA authentication solution with a third party authorization mechanism.

To address the challenges due to lack of useful authorization support in

GAMA1.X, a flexible resource authorization solution has been designed and implemented in GAMA 2.0. The fine-grained authorization mechanism is based on RBAC models which provides a powerful access control solution based on users, groups, roles, and capabilities. It is flexible enough to be employed in many communities with various security requirements. GAMA 2.0 enforces push-mode authorization approach where upon authorization requires, a resource can easily contact the GAMA server via web-services to obtain user's access rights. Also, since the authorization solution is part of the GAMA 2.0 package, it can be easily configured and used without spending significant time and effort.

7.2.2 LDAP-based Authentication Support

GAMA 1.X augments CACL certificate authority with MyProxy certificate repository to provide a flexible user authentication solution. During user registration, CACL encrypts the user's private key with the supplied user password and sends it to MyProxy for future user authentication and short-lived certificate retrieval. Upon authentication, MyProxy attempts to decrypt the user's private key with the user's password. If successful, short-lived proxy certificate is issued and returned to the portal. Otherwise, a denied error message is sent back. One of the great advantages of GAMA 1.X's user authentication is that user's password is never stored in the back-end server. Consequently, it is extremely difficult for unwanted third parties to discover and alter passwords.

GAMA 1.X has been successfully employed and tested in various projects such as CAMERA. GAMA 2.0 follows the same authentication strategy as GAMA 1.X. However, since GAMA 1.X explicitly relies on CACL and MyProxy, many communities that desire alternative types of authentication are hesitant to employ it. For example, a wide variety of projects would like to use LDAP for authentication purposes. Ldap's powerful yet flexible authentication mechanisms, augmented with its wide acceptance as the main directory access method, encourage these projects to employ a LDAP-based authentication solution. To attract the projects that desire to use LDAP server for authentication, GAMA 2.0 is designed to support LDAP-based user authentication. The deployment architecture, Figure 5.4, in the architecture chapter shows that GAMA supports LDAP-based authentication. As seen in the Figure, OpenLDAP is connected to MyProxy via SASL mechanism. Therefore, OpenLDAP delegates authentication requests to MyProxy where they can be properly processed. This implies that LDAP-based projects can employ GAMA 2.0 for authentication, although ultimately MyProxy is used for authentication purposes. It is important to note that while the GAMA 2.0 architecture supports LDAP-base authentication, this feature has not been implemented in the alpha release.

7.2.3 Pluggable Architecture and Implementation

The previous versions of GAMA had explicit reliance on the employed technologies such as MyProxy and CACL. As the grid security technologies are rapidly changing and improving, it is very difficult to incorporate new relevant technologies into the old versions of GAMA. In addition, there are many projects that want to use their own existing technologies. For example, they may already have a customized certificate authority in place, or they already have a repository that they would like to use to store user certificates. It is extremely difficult to use these project-specific tools in GAMA 1.X as it is tightly coupled to very explicit technologies. Also, GAMA 1.X does not support a complete exception handling component to properly report errors to end-users. In terms of testing the implementation, GAMA 1.X lacks a comprehensive and flexible testing solution that examines various parts of code.

GAMA 2.0 was designed to resolve many GAMA 1.X's limitations like the ones mentioned above. Special effort has been made to design a multi-tier, pluggable, and service-oriented architecture that can cope with rapidly evolving relevant technologies. GAMA 2.0 has removed all hard coding of very specific technologies that were implemented in GAMA 1.X and replaced them with a plug-in system. In particular, it introduces various resource adapters such as MyProxy adaptor to hide the underlying technologies such as MyProxy certificate repository. These adapters (drivers) have been successfully implemented to shield the gamaLib from low-level components. Moreover, unlike GAMA 1.X, the 2.0 version has employed well-known programming techniques like the Factory pattern and polymorphism to implement a robust exception handling component. Realizing that it is important to adequately report unrecoverable errors to end-users as well as to resolve my recoverable conditions without user's intervention, GAMA 2.0 offers a detailed yet flexible exception handling solution. At the same time, GAMA 2.0 employs various testing techniques such as unit testing and scenario testing to exam different parts of the system. The GAMA test-component is extendable and the test cases can be easily compiled and run via build scripts. Therefore, as the team is implements the rest of GAMA 2.0's services and functionalities, new test cases can be easily introduced to examine new added functionalities.

7.3 GAMA 2.0 vs Other Identity Management Systems

Section 1.5 described a few current identity management systems; namely, PURSE, GridAuth, and FusionGrid. Here, we discuss advantages and disadvantages of these systems with respect to GAMA. However, instead of individually comparing these solutions to GAMA 2.0, we collectively discuss them in a few relevant contexts. For example, we analyze the communication mechanisms employed in these solutions. Then we report how they address resource authorization issues and challenges. Furthermore, we compare their architecture design and implementation.

7.3.1 Communication Mechanisms

All discussed solutions share a common goal which is to simplify identity management tasks for end-users. To achieve this goal, they propose security systems that consist of two major components: a back-end server which manages user credentials and a set of front-end portals that provide interfaces for end-users. Nevertheless, these systems employ different communication mechanisms. In GAMA 2.0, for instance, essential back-end services such as user authentication and resource authorization services are exposed to public via SOAP-based web-services calls. Using web-services to provide remote procedure calls offers several benefits. For example, user login portlets can be installed on a remote portal that communicates with the GAMA server via web-services. This promotes transparent and automatic proxy retrieval step in the environment that a user tries to log into. Portal developers can benefit from this as they have access to a user's short-lived proxy certificate without explicit knowledge of MyProxy or GAMA. The PURSE portal, in contrast, runs on the same machine that CA and MyProxy are installed. Therefore, when a user logs into a portal, the portal environment somehow needs to retrieve the user proxy certificate from the MyProxy server.

GridAuth provides client APIs that use standard HTTP over SSL to communicate with a back-end server. Because these APIs are developed for specific languages, they can only be integrated into external applications that are implemented in the same programming languages as the client APIs. With respect to GridAuth, GAMA offers some pivotal advantages as its web-services are not based on a programming language or data model. Also, web-services promote much easier integration process by formalizing communication parameters, and requiring less memory and custom-code.

7.3.2 Authorization Support

PURSE and GridAuth, similar to GAMA 1.X, only provide user authentication solutions without explicitly managing resource authorization. Therefore, to have a complete security system, communities that employ PURSE or GridAuth need to separately resolve their resource authorization concerns. Through grid-mapfile mechanism, for example, authorization decisions are made downstream at the resource level. Nevertheless, grid-mapfiles has several major shortcomings such as lack of scalability and expressiveness. A distinct advantage of GAMA 2.0 system over PURSE and GridAuth is its role-based authorization component. It is scalable as it stores authorization-related data in a centralized repository, and it does not require change in each participating site when user's policy changes. Moreover, GAMA 2.0 authorization system employs groups, roles, and capabilities to support broad range of policies and permission types that native operating systems do not support.

In contrast to PURSE and GridAuth systems, FusionGrid supports resource authorization. Its authorization system is called ROAM, and similar to GAMA 2.0 authorization component, it provides an easy-to-manage and comprehensive system for ensuring that users are allowed to access intended resources. ROAM and GAMA 2.0 authorization component share many similarities. For example, they both avoid the push model of authorization and use the pull mode. So upon authorization request, a resource consults back-end server to determine if the connecting user is authorized. Using the pull mode, the authorization path is completely transparent to the user. Moreover, both of the solutions store authorization entities in a centralized repository to ease the burden of maintaining and managing authorization policies. By allowing various resource and permission types, FusionGrid and GAMA 2.0 provide flexible yet powerful authorization solutions. They both, for instance, consider an entire site as resource and support relevant permissions such as access.

Despite the shared similarities between FusionGrid and GAMA 2.0 authorization systems, they differ in one crucial front. FusionGrid authorization has been designed and implemented in a very specific environment to meet the needs of a certain group of users. It is a custom authorization system that fulfills GridAuth community requirements. Therefore, the end result appears to be closely coupled to FusionGrid environment and users. GAMA 2.0 authorization solution, on the other hand, can be employed in many projects. Although GAMA 2.0 was developed to meet the requirements of certain communities, significant effort has been made to make a product that can be shared and used in a wide range of non-crossing communities. By supporting roles and groups, GAMA offers a more advance and flexible authorization solution than FusionGrid. Moreover, while FusionGrid ROAM system stores users, resources, and authorization data in a relational database, GAMA 2.0 stores them in a LDAP directory. GAMA 2.0 has an advantage in this regard since many projects and applications are LDAP-aware, and GAMA 2.0 can be easily integrated into them.

7.3.3 Architecture Design and Implementation

One clear advantage of GAMA 2.0 over other discussed systems is its packaging and deployment. To alleviate the installation and deployment burden on administrators, GAMA employs the Rocks clusters management tool. The rocks installs all GAMA components and performs post-install configuration to provide a fully functional GAMA back-end server “out-of-the-box.” To install other systems, however, the administrator needs to spend significant time and effort downloading and configuring many security-related technologies that are not designed for interoperability. To install PURSE, for example, the administrator has to download tools such as SimpleCA, MyProxy, and MySQL database. Then, he needs to configure these technologies and make sure that they are properly installed and can communicate to one another. These tasks can be extremely time-consuming and tiresome.

All security systems that we discussed share one common weakness. They centrally manage authentication and authorization services which create a single point of failure. While these systems realize the importance of this risk, yet only a few have taken steps to mitigate it. FusionGrid, for instance, attempted to address this weakness by implementing primary and secondary servers. It copies relevant contents of the primary server to the secondary server on a daily basis. Therefore, in the event of a failure, client interfaces automatically switch over to the backup server and users continue interacting with the system. Although, GAMA 2.0 has not addressed the single point of failure weakness, its developers plan to provide a fault tolerance system in the near future. One appealing solution is to follow in FusionGrid footsteps, and implement primary and secondary servers.

Out of the mentioned systems, it seems that only GAMA 2.0 and GridAuth are based on pluggable architecture. They have an edge over other discussed systems as they do not explicitly rely on any technology or tool. Nevertheless, these two systems have taken a different approach to provide technology independent solutions. GAMA 2.0 introduces various resource adapters such as MyProxy adaptor to hide the underlying technologies like MyProxy certificate repository. GridAuth system, on the other hand, includes a plug-in stack made of existing plug-ins each one implementing required interface functions such as login. It appears that FusionGrid, and to smaller extent PURSE, are hard coded to very specific employed technologies, and it is extremely difficult to incorporate new relevant technologies into these systems.

7.4 GamaLib Performance Evaluation

GamaLib is the core part of GAMA 2.0 which provides management, authentication, and authorization services. In designing and implementing gamaLib's services, considerable effort has been made to ensure that they are technology-neutral, easy to maintain, and easy to extend. Although performance has not been a driving factor, many communities would desire efficient implementation of GAMA 2.0 that responds to requests in a timely manner.

This section discusses gamaLib performance evaluation process. It uses sequence of benchmarks to determine the efficiency of gamaLib services. Moreover, it presents a few optimization steps that have effectively reduced the running time of these services.

7.4.1 GamaLib Benchmarks

To measure execution time of essential gamaLib services, several benchmarks have been developed and tested. A description of these benchmarks is listed below.

createSitesTest

1. Invokes createGamaRoot service to create gama root which, in turn, creates system-defined entities such as system defined capabilities and roles.

2. Invokes createNewProject service to create a project under root.

3. Invokes createNewSite service to create two sites under project site (siteA and siteB).

4. Invokes createNewSite service to create two sites under siteA (siteA1 and siteA2).

getGamaSitesTest

1. Invokes getSites service to return sites belonging to siteA (i.e. siteA1 and siteA2).
2. Invokes getAllGamaSites service to return all gama sites (i.e. siteA, siteB, siteA1, and siteA2).

getSiteAttributeTest

1. Invokes getSiteAttributes service to return siteA's attributes (i.e. name, telephone, address, etc.).
2. Invokes getProjectAttributes service to return project's attributes.

updateSiteAttributeTest

1. Invokes updateProjectOptionalAttributes service to update the description attribute of project.
2. Invokes updateSiteOptionalAttributes service to update siteA's telephone number.

searchSitesTest

1. Invokes searchSites service to return all sites with the partial name "site" that belong to siteA (i.e. siteA1 and siteA2).
2. Invokes searchSites service to return all sites with the partial name "siteA2" that belong to siteA (i.e. siteA2).

createResourceTest

1. Invokes createResource service to create two resources in siteA1 (siteA1Resource1 and siteA1Resource2).
2. Invokes createResource service to create two resources in siteA2 (siteA2Resource1 and siteA2Resource2).

searchResourcesTest

1. Invokes searchResources service to return all resources with the partial name “siteA1Resource1” that belong to siteA1 (i.e. siteA1ResourceA1).
2. Invokes searchResources service to return all resources with the partial name “resource” that belong to siteA1 (i.e. siteA1ResourceA1 and siteA1ResourceA2).
3. Invokes searchResources service to return all resources with the partial name “siteA2Resource1” that belong to siteA1 (i.e. result set is empty).

createUserTest

1. Invokes createUser service to create a user at the root level (rootUser1).
2. Invokes createUser service to create two project users (projectUser1 and projectUser2).
3. Invokes createUser service to create two siteA1 users (siteA1User1 and siteA1User2).
4. Invokes createUser service to create two siteA2 users (siteA2User1 and siteA2User2).

createGroupTest

1. Invokes createGroup service to create a group called siteA1Group1 in siteA1 with two members siteA1user1 and siteA1user2.
2. Invokes createGroup service to create a group called siteA2Group1 in siteA2 with two members siteA2user1 and siteA2user2.

getUserAttributesTest

1. Invokes getUserAttributes service to retrieve rootUser1 attributes.
2. Invokes getUserAttributes service to retrieve siteA1User1 attributes.

updateUserAttributesTest

1. Invokes updateUserOptionalAttributes service to update email and postal address of siteA1User1.

getGamaUsersTest

1. invokes getAllGamaUsers service to retrieve all gama users (i.e. rootUser1, projectUser1, projectUser2, siteA1User1, siteA1User2, siteA2User1, and siteA2User2).

getResVisibleUsersTest

1. Invokes getResourceVisibleUsers service to return all users who can access siteA1Resource1 (i.e. rootUser1, projectUser1, projectUser2, siteA1User1, siteA1User2).

createRolesTest

1. Invokes createRole service to create a role called anonymousUser with read capability. This role is created in the root site.

grantRolesTest

1. Invokes grantRole service to grant anonymousUser role to anonymous user in the root site.
2. Invokes grantRole service to make rootUser1 the gamaAdmin (i.e. grant gamaAdmin role to rootUser1 in the root site).
3. Invokes grantRole service to make projectUser1 projectAdmin (i.e. grant projectAdmin role to projectUser1 in the project site).
4. Invokes grantRole service to make projectUser2 admin of siteA1 (i.e. grant siteAdmin role to projectUser2 in siteA1).

5. Invokes grantRole service to make members of siteA1Group1 admin of siteA1 resources (i.e. grant resourceAdmin role to siteA1Group1 in siteA1).

getUserCapsTest

1. Invokes grantCapability service to grant addUser capability to siteA1User2 in site1A2.
2. Invokes denyCapability service to deny siteA1User2 the executeResource capability in site1A2.
3. Invokes getUserCapabilities to retrieve siteA1User2 capabilities in siteA1.
4. Invokes getUserCapabilities to retrieve siteA1User2 capabilities in siteA1.

siteAdminPermissionTest

1. Invokes checkPermission service to determine if projectUser2 has the addUser capability in siteA1.
2. Invokes checkPermission service to determine if projectUser2 has the addSite capability in siteA2.

resAdminPermissionTest

1. Invokes checkPermission service to determine if siteA1User2 has the addResource capability in siteA1.
2. Invokes checkPermission service to determine if siteA1User2 has the addUser capability in siteA1.

authenUserPermissionTest

1. Invokes checkPermission service to determine if siteA1User2 has the writeResource capability in siteA1Resource1.
2. Invokes checkPermission service to determine if siteA1User2 has the readResource capability in siteA1Resource1.

Table 7.2: execution time and LDAP access frequency of each benchmark (before optimization)

Benchmark Name	Execution Time in sec	#of LDAP Accesses	#of LDAP Reads	#of LDAP Writes
createSitesTest	2.178	87	55	32
getGamaSitesTest	0.328	23	23	0
getSiteAttributeTest	.045	4	4	0
updateSiteAttributeTest	.030	2	0	2
searchSitesTest	0.16	8	8	0
createResourceTest	0.188	16	10	6
searchResourcesTest	0.153	9	9	0
createUserTest	0.346	27	17	10
createGroupTest	0.105	10	8	2
getUserAttributesTest	0.070	4	4	0
updateUserAttributesTest	.051	2	0	2
getGamaUsersTest	0.410	30	30	0
getResVisibleUsersTest	0.325	25	25	0
createRolesTest	0.055	3	2	1
grantRolesTest	0.465	39	27	12
getUserCapsTest	0.470	35	32	3
siteAdminPermissionTest	0.356	30	30	0
resAdminPermissionTest	0.570	46	46	0
authenUserPermissionTest	0.939	72	72	0

3. Invokes checkPermission service to determine if siteA1User2 has the addSite capability in siteA1Resource1.

7.4.2 Evaluation Procedure and Result

The gamaLib source code, the benchmarks, and an OpenLDAP installation all reside in a Linux-based virtual machine. A test suit runs all the benchmarks in the order that they are described in the previous section. The test suit, which deletes all LDAP entries before and after running the benchmarks, can be invoked via an Ant script.

Table 7.2 displays the execution time and LDAP access frequency of each benchmark. The benchmarks are relatively simple with fast execution time and considered gamaLib services communicate with an OpenLDAP server. Therefore, LDAP access data is extremely useful in identifying inefficient benchmarks and services.

Optimization Steps

Considering the number of LDAP accesses (i.e. LDAP read or write) as a strong performance evaluation metric, Table 7.2 suggests many gamaLib services have performance problem. To reduce the execution time of various gamaLib services, the author has considered and implemented a few optimization steps. Because LDAP is the major backend repository, these steps focus on minimizing LDAP accesses and applying relevant LDAP tuning techniques such a indexing. A summary of the optimization approaches is described below.

Refining LDAP Search Queries

Some benchmarks such as createSiteTest are expected to have a large number of LDAP accesses due to creation of system defined capabilities, roles, users, and more. However, other benchmarks such as getGamaUsersTest should require much fewer LDAP accesses than corresponding values reported in Table 7.2. Revisiting gamaLib's LDAP search queries revealed the root cause of the problem: gamaLib services were making many "simple" LDAP queries to obtain the needed data. Therefore, an optimization step would be to merge these simple queries to construct fewer but more "complex" search queries. For instance, getUserAttributes service performs several LDAP searches to obtain information such as user's unique identifier, creation time, and attributes. As LDAP offers flexible and powerful filtering mechanism, all user information can be obtained via one search query. In essence, we can achieve significant performance gain by reducing the number of LDAP accesses. The latter can be achieved by exhausting the LDAP filtering mechanism to create complex search queries that would return the desired data in a few steps.

Performance Tuning the LDAP directory

Another optimization step is to exploit several LDAP tuning mechanisms. In particular, indexing is a popular technique that if employed correctly, it can significantly improve search performance. As applying the previous optimization technique results in complex search queries with several filter terms, it seems that indexing can reduce the running time of many read-based gamLib services. Therefore, we have created indices to match the popular filter components used in many gamaLib search queries such as objectClass and Organizational Unit (OU). Other tuning techniques such as increasing cache size, though not implemented in this work, might result in further performance improvement. The development team will consider these techniques in the future if necessary.

Evaluation Results After Optimization

Table 7.3 displays benchmarks' execution time and LDAP access data after applying the mentioned optimization techniques. It is clear that these optimization steps were successful in improving the performance of various gamaLib services. Refining LDAP search queries technique, in particular, significantly reduced the benchmarks' execution time by decreasing their total number of LDAP accesses. However, createResourceTest and createSiteTest have witnessed an increase in the number of LDAP writes. This is due to creation of various internal "entity containers" such as user container and capability container which ultimately stores entities like users and capabilities as they are created. Creating these containers at the time of creating a site or a resource potentially decreases the performance of the corresponding services (i.e. createSite service and createResource service). For instance, it is evident from the tables that createResourceTest benchmark did not benefit from this optimization. Nevertheless, other services such as createUser, createCapability, and createRole greatly benefited as they can assume that containers already exist and they neither need to create them nor to check for their existence.

Taking advantage of LDAP's indexing mechanism effectively improved the performance of heavily search-based services. For example, this technique has reduced the execution time of checkPermission service. Table 7.3 suggests performance improvement in all benchmarks that invoke checkPermission service. This is very encour-

Table 7.3: execution time and LDAP access frequency of each benchmark (after optimization)

Benchmark Name	Execution Time in sec		#of LDAP Accesses	#of LDAP Reads	#of LDAP Writes
	Without Indexing	With Indexing			
createSitesTest	1.775	1.55	59	1	58
getGamaSitesTest	0.123	0.116	2	2	0
getSiteAttributeTest	0.02	0.014	2	2	0
updateSiteAttributeTest	0.049	0.047	2	0	2
searchSitesTest	0.078	0.058	2	2	0
createResourceTest	0.527	0.485	20	0	20
searchResourcesTest	0.091	0.076	3	3	0
createUserTest	0.177	0.167	7	0	7
createGroupTest	0.45	0.038	2	0	2
getUserAttributesTest	0.047	0.045	2	2	0
updateUserAttributesTest	0.065	0.050	2	0	2
getGamaUsersTest	0.085	0.073	1	1	0
getResVisibleUsersTest	0.076	0.071	2	2	0
createRolesTest	0.014	0.011	1	0	1
grantRolesTest	0.321	0.303	15	5	10
getUserCapsTest	0.253	0.202	9	6	3
siteAdminPermissionTest	0.256	0.166	7	7	0
resAdminPermissionTest	0.273	0.185	8	8	0
authenUserPermissionTest	0.453	0.281	12	12	0

aging as checkPermission is one of the essential gamaLib services which will be called frequently by communities that use GAMA 2.0 as their identity management solution.

Chapter 8

Conclusion

GSI-based security systems are usually difficult to deploy and use. GAMA 1.X employs web-services technologies and the Rocks clusters management tool to overcome these limitations. Nevertheless, its lack of usable authorization support and LDAP-based authentication makes GAMA 1.X unattractive to many communities. In addition, GAMA 1.X is tightly coupled to the employed technologies. As a result, it is difficult to replace these technologies with more recent and mature ones. GAMA 2.0 resolves these issues by providing a flexible, fine-grained resource authorization solution and support for LDAP-based user authentication. Moreover, its multi-tier architecture is pluggable to cope with the rapidly evolving relevant technologies.

The essential design goal for GAMA 2.0 is to make grid security simple for system administrators as well as for users by providing functionalities such as resource authorization and user authentication. As a result, administrators are not required to spend significant time and effort aggregating separate authentication and authorization solutions. Moreover, since GAMA 2.0 uses web-services to expose its authentication and authorization capabilities, they can be accessed remotely from portals running on different machines. To alleviate the installation and deployment burden on the administrator, GAMA 2.0 employs the Rocks clusters management tool. Rocks installs various GAMA 2.0 components with minimum system administrator intervention and performs many operations such as configuring tools, deploying web-services, and starting the necessary services through startup scripts. In addition, since grid end-users such as scientists and researchers that have little or no computer security background desire simple experience with grids, GAMA 2.0's back-end component creates and manages users' credentials on behalf of them. Also, its front-end consists of a set of portals that provide various interfaces for users.

GAMA 2.0 was developed to address security concerns in many non-crossing communities. By having frequent requirement elicitation sessions with potential GAMA 2.0 clients, the development team was able to harness requirements that many non-overlapping, academic-related projects share. Taking these requirements into account, in design and implementation phases, resulted in a flexible architecture that can be applied to many projects. For instance, significant effort has been made to develop a centralized, fine-grained resource authorization solution based on RBAC models. This solution is flexible in that it can be employed in a wide range of communities with different goals and requirements. Furthermore, GAMA 2.0's architecture supports LDAP-based authentication to attract many projects that desire this form of user authentication.

We have successfully implemented the GAMA 2.0 reference infrastructure. Realizing that grid security technologies and tools are improving quickly, the development team has strived to make the implementation technology independent and extendable. The alpha release uses adapters (drivers) to hide the underlying technologies such as OpenLDAP and MyProxy. Moreover, the Command design pattern was employed to implement the drivers. The Command pattern provides well-desired loose coupling since it decouples the object that invokes the operation from the one that knows how to perform it. Moreover, by considering commands as first class objects, the Command design pattern allows commands to be manipulated and extended in a similar way as any other object. At the same time, adding new commands is easy since it does not require modifying the existing ones. In developing the exception handling component, we have used well-known programming techniques such as the Factory pattern and polymorphism. The exception handling solution is multi-layered corresponding to the levels of abstraction such as drivers. Therefore, it is easily possible to extend, modify, and delete the existing exceptions as well as to add new ones. Furthermore, the GAMA development team designed and implemented a comprehensive testing solution. It consists of many unit tests that individually examine small parts of source code and collectively constitute scenarios that model the real use of the GAMA system. Although performance was not the driving factor in implementing GAMA 2.0, a sequence of benchmarks revealed that some GAMA 2.0 services take a long time to execute. A few optimization steps were successfully implemented that reduced the execution time of many services. These steps focused on refining search queries to minimizing LDAP accesses and applying relevant LDAP tuning techniques such as indexing.

As for future work, the development team will explore new methods to make GAMA 2.0 more service-oriented and pluggable. Although the current pluggable architecture is based on adapters, it will be intriguing to investigate alternative approaches that result in a pluggable system. For instance, GridAuth uses a plug-in stack that contains the required plug-ins (see section 7.3.3). Moreover, in the alpha release, only authorization and authentication services can be accessed through web-services calls. The development team will analyze the advantages and disadvantages of exposing other services to public via web-services technologies. For instance, accessing services that write into back-end repositories such as user registration service will make GAMA 2.0 more portable at the cost of increasing vulnerability to malicious attacks. In addition, there are still a few features that have not been implemented in the alpha release such as certificate delegation. Furthermore, LDAP-based authentication is supported by the GAMA 2.0 architecture, but currently not implemented. The development team intends to resolve these open issues among others before subsequent releases.

Bibliography

[1] Akenti Certificate Schema. http://ww-itg.lbl.gov/Akenti/docs/ AkentiCertificate.xsd. Visited on December 08, 2008.

[2] Biomedical Informatics Research Network (BIRN) Project. http://www.nbirn.net/. Visited on December 08, 2008.

[3] Community Cyberinfrastructure for Advanced Marine Microbial Ecology Research and Analysis (CAMERA) Project. http://camera.calit2.net/. Visited on December 08, 2008.

[4] DataGrid Project. http://www.edg.org/. Visited on December 08, 2008.

[5] Definition of Internet. http://www.nitrd.gov/fnc/Internet_res.html. Visited on December 08, 2008.

[6] GridAuth Credential Management System. http://www.gridauth.com. Visited on December 08, 2008.

[7] Groovy Programming Language. http://groovy.codehaus.org/. Visited on December 08, 2008.

[8] JUnit 4 Testing Library. http://www-128.ibm.com/developerworks/java/library/j-junit4.html. Visited on December 08, 2008.

[9] Network for Earthquake Engineering Simulation (NEES) Project. http://nees.ucsd.edu/. Visited on December 08, 2008.

[10] One-Time Password Authentication Solution. http://msdn.microsoft.com/en-us/magazine/cc507635.aspx. Visited on December 08, 2008.

[11] OpenLDAP Software. www.openldap.org. Visited on December 08, 2008.

[12] Security Assertion Markup Language (SAML) Specification. http://xml.coverpages.org/saml.html. Visited on December 08, 2008.

[13] The Advantages of Java. http://www.theallineed.com/webmasters/07072580.htm. Visited on December 08, 2008.

[14] WS-Resource Framework. http://www- 106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf. Visited on December 08, 2008.

[15] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Gianoli, F. Spataro, F. Bonnassieux, P. Broadfoot, G. Lowe, L. Cornwall, et al. Managing Dynamic User Communities in a Grid of Autonomous Resources. *Arxiv preprint*, 2003.

[16] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Gianoli, F. Spataro, F. Bonnassieux, P. Broadfoot, G. Lowe, L. Cornwall, et al. Managing Dynamic User Communities in a Grid of Autonomous Resources. *Arxiv preprint cs/0306004*, 2003.

[17] O. Ashford et al. Collected papers of Lewis Fry Richardson. 1993.

[18] K. Bhatia, S. Chandra, and K. Mueller. GAMA: Grid Account Management Architecture. In *1st IEEE International Conference on e-Science and Grid Computing*, 2005.

[19] J. Burruss, T. Fredian, and M. Thompson. Simplifying FusionGrid security. In *Challenges of Large Applications in Distributed Environments, 2005. CLADE 2005. Proceedings*, pages 95–103, 2005.

[20] J. Burruss, T. Fredian, and M. Thompson. ROAM: An Authorization Manager for Grids. *Journal of Grid Computing*, 4(4):413–423, 2006.

[21] R. Buyya and R. Buyya. *High Performance Cluster Computing: Programming and Applications*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1999.

[22] D. Chadwick and A. Otenko. The PERMIS X. 509 role based privilege management infrastructure. *Future Generation Computer Systems*, 19(2):277–289, 2003.

[23] D. Ferraiolo and D. Kuhn. Role Based Access Control. *15th National Computer Security Conference*, pages 554–563, 1992.

[24] I. Foster and C. Kesselman. The Grid: Blueprint for a Future Computing Infrastructure, 1999.

[25] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *Proceedings of the 5th ACM conference on Computer and communications security*, pages 83–92. ACM Press New York, NY, USA, 1998.

[26] I. Foster, V. Nefedova, M. Ahsant, R. Ananthakrishnan, L. Liming, R. Madduri, O. Mulmo, L. Pearlman, and F. Siebenlist. Streamlining Grid Operations: Definition and Deployment of a Portal-based User Registration Service. *Journal of Grid Computing*, 4(2):135–144, 2006.

[27] W. Hommel and M. Schiffers. Supporting Virtual Organization Life Cycle Management by Dynamic Federated User Provisioning. In *Submitted to 13th Workshop of the HP OpenView University Association (HP-OVUA), Nice/France*, 2006.

[28] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X. 509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2002.

[29] S. Krishnan, K. Baldridge, J. Greenberg, B. Stearn, and K. Bhatia. An End-to-End Web Services-Based Infrastructure for Biomedical Applications. In *Grid Computing, 2005. The 6th IEEE/ACM International Workshop on*, pages 77–84, 2005.

[30] W. Link. CACL, A CA System with Automated User Authentication. *San Diego Supercomputer Center*, 2003.

[31] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal. Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework. *High Performance Computing: Paradigm and Infrastructure*, 2006.

[32] M. Meisinger. GAMA 2.0 Architecture Document. Internal Document, Available Upon Request, 2008.

[33] M. Meisinger. GAMA 2.0 Requirements Document. Internal Document, Available Upon Request, 2008.

[34] G. Moore. Cramming more components onto integrated circuits (reprinted from electronics, pg 114-117, april 19, 1965). *Proceedings of the Ieee*, 86(1):82–85, 1998.

[35] P. (née Broadfoot) Hopcroft, A. Martin, P. R. Group, and O. U. C. Laboratory. *A Critical Survey of Grid Security Requirements and Technologies*. Oxford University Computing Laboratory, 2003.

[36] J. Novotny, S. Tuecke, and V. Welch. An Online Credential Repository for the Grid: MyProxy. In *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*, pages 104–111, 2001.

[37] P. Papadopoulos, M. Katz, and G. Bruno. NPACI Rocks: tools and techniques for easily deploying manageable Linux clusters. *Concurrency and Computation: Practice & Experience*, 15(7):707–725, 2003.

[38] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A community authorization service for group collaboration. In *Policies for Distributed Systems and Networks, 2002. Proceedings. Third International Workshop on*, pages 50–59, 2002.

[39] A. Pereira and V. Muppavarapu. SM Chung http://doi. ieeecomputersociety. org/10.1109 TDSC. 2006.26" Role-Based Access Control for Grid Database Services Using the Community Authorization Service. *IEEE Trans. Dependable and Secure Computing*, 3(2):156–166, 2006.

[40] F. Shuman. History of Numerical Weather Prediction at the National Meteorological Center. *Weather and Forecasting*, 4(3):286–296, 1989.

[41] J. Steiner, C. Neuman, and J. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proc. Winter USENIX Conference*. Dallas), 1988.

[42] M. Thompson, W. Johnston, S. Mudumbai, G. Hoo, K. Jackson, and A. Essiari. Certificate-based Access Control for Widely Distributed Resources. In *Proc. 8th Usenix Security Symposium*, 1999.

[43] S. Tuecke. Grid Security Infrastructure (GSI) Roadmap. *Grid Forum Security Working Group Draft*, 2001.

[44] S. Tuttle, A. Ehlenberger, R. Gorthi, J. Leiserson, R. Macbeth, N. Owen, S. Ranahandola, M. Storrs, C. Yang, I. T. S. Organization, et al. *Understanding LDAP Design and Implementation*. IBM, International Technical Support Organization, 2004.

[45] G. von Laszewski. Grid Computing: Enabling a Vision for Collaborative Research. *Lecture Notes in Computer Science*, pages 37–52, 2002.