A SERVICE-ORIENTED, SCALABLE, SECURE FRAMEWORK FOR GRID-ENABLING LEGACY SCIENTIFIC APPLICATIONS

By

VIVEKANANTHAN SANJEEPAN

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2005

To my parents

ACKNOWLEDGMENTS

I would like to take this opportunity to express my sincere gratitude to all the people who helped me achieve this milestone in my life.

I thank Prof. Fortes, Prof. Lam and Prof. Figueiredo very much for their advice and support. Working with them has been a wonderful, pleasant and learning experience. I also would like to thank all the staff and students of ACIS who have helped me in numerous ways all along.

I also thank my family members, relatives and friends who helped me personally and financially to reach my dream.

TABLE OF CONTENTS

LIST OF TABLES

Table page

LIST OF FIGURES

Figure page

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

A SERVICE-ORIENTED, SCALABLE, SECURE FRAMEWORK FOR GRID-ENABLING LEGACY SCIENTIFIC APPLICATIONS

By

Vivekananthan Sanjeepan

December 2005

Chair: José A.B. Fortes
Major Department: Electrical and Computer Engineering

This thesis describes a scalable and secure framework for Grid-enabling legacy scientific applications using a service-oriented architecture. In the context of this thesis Grid-enabling means turning an existing application, installed on a Grid resource, into a service and generating the application-specific user interfaces to use that application through a web portal. Scalability, with respect to Grid-enabling a large number of different types of applications, is achieved by providing a common abstraction for a category of applications and providing a "generic" application service to wrap those applications as services. The focus of this framework is on Grid-enabling "command-line" scientific applications. The novel aspect of the framework is that the entire process—from turning an application into a service to the user-interface generation for that application—is done automatically without requiring coding or Grid-system downtime. Technical contributions of this thesis are (1) an abstraction that is generic enough to capture the descriptions of numerous types of applications, (2) an XML

schema that captures the abstraction, (3) a generic application service to interpret the abstraction and (4) a user-interface generation technique that dynamically generates application specific user interfaces. Portlet technology is used to dynamically generate application-specific user-interfaces. Further, the framework makes it possible to customize the applications for different user groups by simplifying, restricting, extending or composing the functionalities of applications.

The security architecture of the framework supports multiple, loosely-coupled security domains using a security federation. This loosely-coupled security federation makes it possible for the resource owners who belong to a security domain to share the resources with Grid users who belong to another security domain while providing a single-sign-on capability to the users and maintaining access control over their resources without changing the existing security mechanisms. The framework is useful for organizations to Grid-enable large number of applications, to expose them as services and to aggregate and provide access to the Grid-enabled applications through portals.

CHAPTER 1
INTRODUCTION

Grid computing enables “flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions and organizations” [1, p.200]. In the recent past, significant amount of progress has been made in the core areas of Grid computing middleware. In order to make use of the enormous computing power made available by Grid computing technologies, applications are required to be written using Grid APIs. There are several projects focused mainly on making the Grid APIs simpler and easier to use [2-4]. However many users can benefit significantly from being able to run a wealth of valuable “legacy” scientific applications that can not be re-factored using Grid APIs. This thesis describes a scalable and secure framework to Grid-enable such “legacy” scientific applications using a service-oriented architecture.

Legacy applications are broadly defined as codes developed with technologies that precede in time the introduction of Grid computing, Web services and other recent computing approaches and programming best-practices. The proposed approach concentrates on the Grid-enabling of a large class of such applications, “command-line” applications.

Ideally, these applications should be re-factored using Grid APIs to Grid-enable them. However, in most cases it is not easy if not impossible. Many legacy applications do not have the source code available in the public domain. A lack of proper documentation and the complexity of the applications can make re-factoring difficult

even when the source code is available. The need to re-factor a large number of applications compounds the problem.

In the context of this thesis, Grid-enabling means turning an existing application, installed on a Grid resource, into a service and providing access to the application by generating the application-specific user interfaces to use that application through a web portal. Given the fact that there is large number of legacy scientific applications that need to be grid-enabled, scalability, with respect to Grid-enabling a large number of different types of applications, is an important requirement. Further the user community consists of a variety of people with varying skills. It is important that the proposed solution addresses their needs in a user-friendly manner.

Grid-enabling of these applications is very valuable for the following reasons:

- Applications can benefit by being able to choose the best available HPC resources at the time of execution because the Grid can provide resources otherwise not available.
- A significant percentage of the large simulations involve executing the same applications several times over a range of parameters (parameter “sweep”). By Grid-enabling them, such executions can be performed in parallel.

This thesis describes a service-oriented, scalable and secure framework to Grid-enable command-line legacy scientific applications without requiring modifications or recompilations of the original source code. Technical contributions of the thesis are: (1) an abstraction that is generic enough to capture the descriptions of numerous types of applications, (2) an XML schema that captures the abstraction, (3) a Generic Application Service (GAP Service) to interpret the abstraction and (4) a user-interface generation technique that dynamically generates application specific user interfaces. Since the foundation of the framework is the GAP service, the framework is called the GAP framework. Grid-enabling a legacy application requires only the provision of the

necessary application description XML document that will be used dynamically when that application is used. Dynamic application-specific user interfaces are generated using Java Portlet and Java Server Pages (JSP) technologies.

Grid computing systems, by definition, are decentralized and autonomous. As such, the entities of the GAP framework can be autonomous and belong to different security domain. Each entity will want to retain access control over its resources. Security architecture of such a system must cope with the above constraints while providing a single-sign-on capability to the users. This thesis proposes a security architecture that addresses the above requirements using federation. The proposed federation loosely-couples the security domains without requiring them to change their existing security mechanisms.

The thesis is organized as follows. Chapter 2 elaborates the motivations and requirements of Grid-enabling of applications. Chapter 3 and 4 provides detailed description of the functional architecture and functional implementation respectively. Chapter 5 and 6 describes the security architecture and security implementation respectively. Chapter 7 provides an evaluation. Chapter 8 discusses some of the related work. Chapter 9 concludes the thesis with an outline of the future directions.

CHAPTER 2
MOTIVATIONS AND REQUIREMENTS

The motivation for this work is to provide a framework to Grid-enable a large number of applications securely, so that these application can be run on a computing Grid, accessed ubiquitously using standard and widely accepted protocols by Grid-portals and other aggregating systems and used by end users via application specific user interfaces. This chapter identifies five specific requirements of Grid-enabling and outlines briefly how these requirements are satisfied in the proposed GAP framework. Chapters 3-6 provide detailed description of the architecture and implementation of the GAP framework.

2.1 Service-Oriented Architecture

In general, for an application to run on a computing Grid, it has to be interfaced to the Grid middleware for various reasons such as obtaining a suitable computing resource, setting-up the execution environment, setting-up data used by the application, scheduling a job, transferring execution output etc. The GAP framework provides a Generic Application (GAP) service that is dynamically configured by an application description to interface a legacy application to the Grid middleware. The GAP service is a stateful Web Services Resource Framework (WSRF) service. WSRF [5] is a set of specifications that was the result of convergence of Web services and Grid computing technologies. This service is generic because it is capable of handling all command-line applications. The application description is an XML document that is created from the information given by the application specialists. This XML document conforms to an XML schema

constructed for this project (listed in Appendix B). A pool of GAP services are managed by the GAP factory service. The GAP factory service, the GAP services and other related services are called the GAP framework-services.

The selection of Web services and service-oriented-architecture is influenced by several factors.

- *Technology Neutral*: Service endpoints are platform-independent. Computing Grids are heterogeneous in nature and technology neutrality is a significant advantage.
- *Standards-Based*: Protocols are standard-based and hence allow interoperation of different implementations.
- *Abstracted:* Service interface are independent from, and hide details of, service implementations.
- *Publishable and consumable*: A service interface, in the form of WSDL and service endpoint (for the "generic" application service to be described), can be published in, and discovered from, a UDDI directory.
- *Stateful*: By modeling services as WSRF services, it is possible to maintain and interact with state in a standard way.

2.2 Application-Specific User Interfaces

The second requirement of Grid-enabling is to provide access to the applications, via a Grid portal, using application specific user interfaces. Grid portals are increasingly becoming the access points for Grid resources. There are two general models of Grid computing portals:

- Portals that provide ubiquitous access to High Performance Computing (HPC) resources. For this type of Grid portals, users are expected to have accounts with HPC resources and are responsible for installing the necessary applications and configuring the execution environment to run those applications. Hence users are expected to have sufficient knowledge about the underlying Grid computing middleware. Although it provides flexibility to the technically savvy users, this model is not well suited to a larger community of users who are not necessarily familiar with Grid computing technologies and middleware. OGCE [6] and HotPage [7] are two examples of this type of portals.

- Portals in which administrators take the responsibility of installing the applications, configuring the execution environment, interfacing the applications and middleware and providing a user friendly interface to the applications. For this type of portals, users can transparently get their job done without the need to know the underlying middleware. These are also known as application portals. In-VIGO [8] and PUNCH [9] are two examples that follow this model.

Although the second model makes the life of portal users easier, portal administration becomes much more difficult. In this model, portal administrators, who are not necessarily experts in different scientific application domains, need to study and understand applications from many application domains in order to Grid-enable them. Additional coding overhead is incurred because of the need to interface middleware and applications. Consequently, this approach does not scale well when a large number of applications need to be made available through a Grid portal.

This thesis describes an approach that generates application specific user interfaces, similar to one provided by application portals, automatically with the help of the GAP service. Portlet and JSP technologies are used to construct the user interfaces. A set of portlets and JSP pages that perform the task of generating application specific user interfaces automatically and dynamically is hosted on a Portal server. This portal is called the GAP portal.

2.3 Scalability

Grid-enabling is not a problem if only a few applications need to be Grid-enabled. However, scalability becomes an important requirement if a large number of applications need to be Grid-enabled. The left hand side of Figure 2-1 shows the typical Grid-enabling process. In this process, scientists (application specialists) and developers work together to create an application service to encapsulate the application and then develop a portal to provide access to the application service. Application specialists come from many

different disciplines like physics, chemistry, biology etc. They can not be expected to know Grid technologies. On the other hand, developers are not familiar with various application disciplines. This makes the process of Grid-enabling an application very cumbersome. The need to Grid-enable large number of applications compounds the problem.

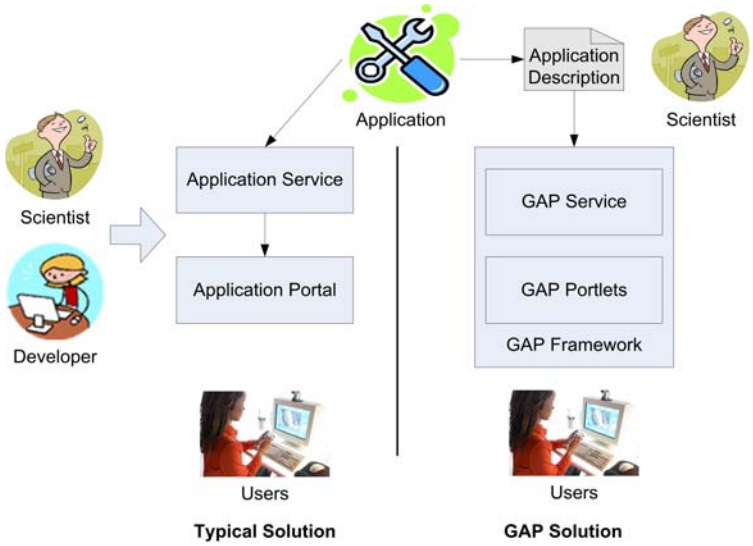

Figure 2-1. Comparison of a typical Grid-enabling solution and the GAP solution. In a typical solution a scientist and a developer have to collaborate to code an application service and an application portal. In the GAP solution only an application description needs to be created.

In the GAP framework (shown on the right hand side of Figure 2-1), only a description of the application needs to be created. Once the description is created the rest is performed automatically. The GAP framework clearly separates the roles of user, administrators and application specialists, without requiring one to know the others' skills. Scalability of this approach is achieved by providing a common abstraction for the class of command-line applications and delivering the applications through a dynamically reconfigurable GAP Service. The distinguishing feature of this approach compared to other approaches is that no coding, code generation, compilation or service/web

application deployment is involved every time a new application is Grid-enabled. Hence system downtime is not required when a new application is Grid-enabled.

2.4 Usability

One important non functional requirement often not well recognized is the usability of a proposed approach for Grid-enabling. The success of a solution ultimately depends on how comfortable the users feel in using it. The proposed approach incorporates features to enhance user experience.

One such feature is application customization. Users of the applications vary from students to researchers with varying needs and skills. Beginners like students would like to have a simplified application. Administrator would like to restrict the application capabilities given to certain user groups, for example guests, so that the compute resources are not overloaded with several long running jobs. Researcher would like to do parameter sweep (executing the same application over and over again while changing a small set of parameters systematically) in one go. The proposed approach provides a simple mechanism whereby applications can be customized to various user groups to satisfy their needs. Customization may involve simplifying, restricting or extending the application.

Another usability feature is user input validation. When an application is executed in a computing Grid, many tasks, such as obtaining resource, setting-up data, setting-up execution environment etc., are performed before actually executing the application. Sometimes, after doing all these things, applications fail because some user inputs are in error. The proposed approach avoids such frustrating failures by validating user inputs before submitting jobs to compute resources.

2.5 Security

Grid systems, by nature, consist of autonomous entities. In the context of the GAP framework, a GAP portal, provided by one organization, will want to have the ability to use the GAP services provided by another organization. The GAP services, in turn, will want to have the ability to use compute resources provided by a different organization. As can be seen, the GAP framework may consist of autonomous entities that belong to different security domains. The primary security requirements of the GAP framework are as follows.

- The users should be able to single-sign-on.
- The entities must be able to retain access control over their resources.
- The security administration within and between the security domains must be manageable.

The security architecture of the GAP framework addresses these requirements using a federation. The foundations of the federation are built using standard and widely accepted Web services security technologies. Security Assertion Markup Language (SAML) [10] is used to build the federation. The use of standard and widely accepted technologies enhances the interoperability of the GAP framework with other Grid computing systems.

CHAPTER 3
ARCHITECTURE

The GAP framework provides a scalable and secure Grid-enabling mechanism for command-line legacy applications by providing a common abstraction to all command-line applications. The novel aspect of the GAP Framework is that the process of Grid-enabling (from turning an application into a service to creating application-specific user interface to use that application) is done automatically. Its scalability, with respect to Grid-enabling a large number of different types of applications, is achieved by providing a common abstraction to all command-line applications and supporting the use of them using a generic application service. Figure 3-1 shows the high-level system architecture underlying the GAP framework. It consists of the GAP portal, the GAP framework-services and the Grid middleware (in this case, the middleware is In-VIGO). There are two processes involved in Grid-enabling legacy applications. In the application provisioning process, an application is made available to the GAP framework. This process is performed by application specialists. In the application usage process, the applications are made available to the users for execution through a portal using application specific user interfaces. The system administrators manage the GAP portal, the GAP framework-services and the Grid middleware.

Although the work has been done in the context of In-VIGO, the GAP architecture is not specific to In-VIGO Grid computing middleware. Any Grid computing middleware that provides job and file system services can be used with the GAP architecture. Further the GAP architecture treats the GAP portal, GAP framework-services and the Grid

computing middleware as three independent and autonomous components. The only requirement is the existence of trust relationships between the portal and the services and between the services and the Grid middleware. The detailed description of the security architecture of the GAP framework is given in Chapter 5.

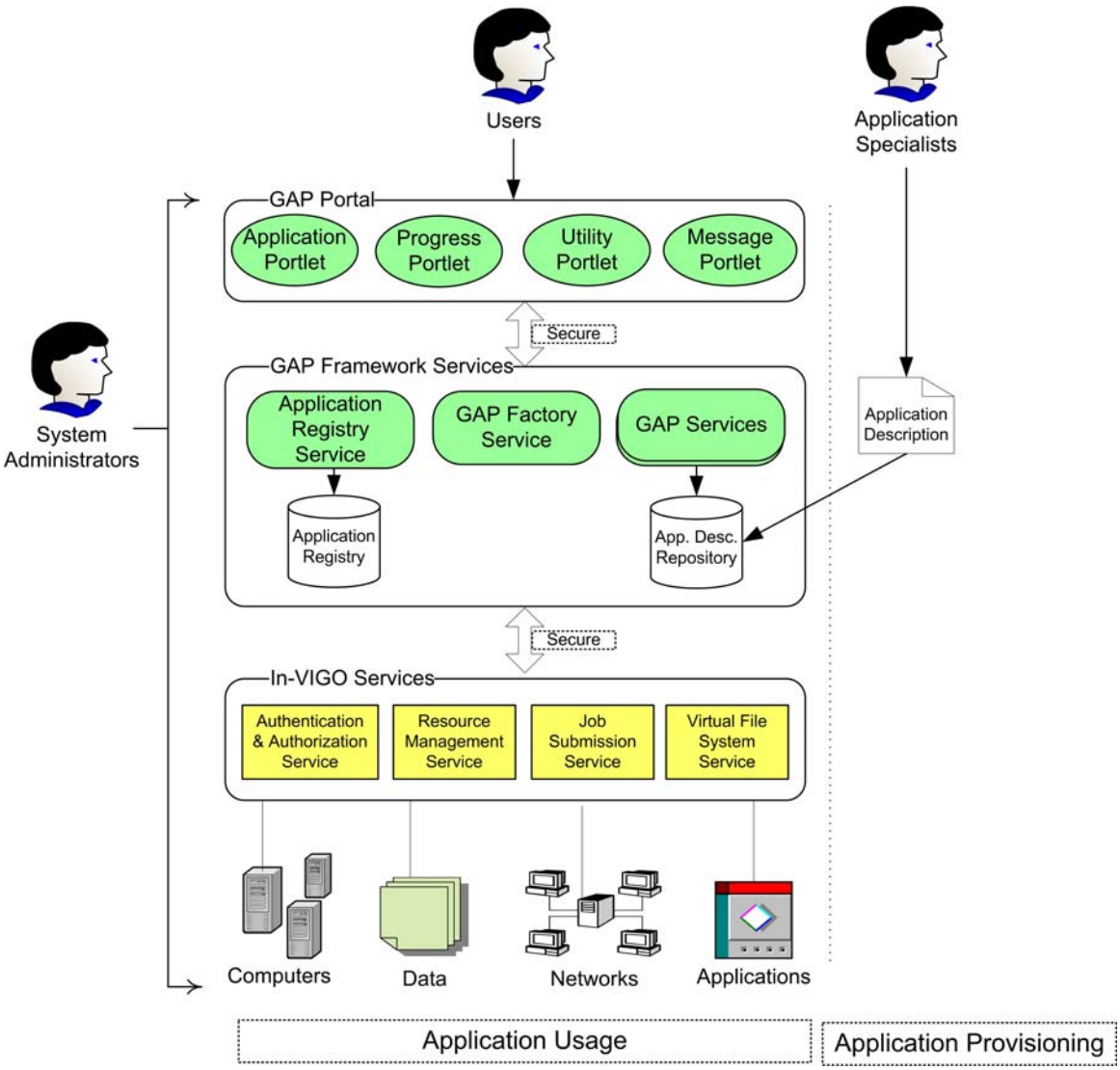

Figure 3-1. The architecture of the GAP framework. The figure shows the GAP Portal, the GAP framework-services and In-VIGO middleware. The figure shows the two processes that are part of Grid-enabling (the application provisioning shown on the right side and application usage shown on the left side). It also shows the three different actors (system administrators, users and application specialists) involved in these processes.

3.1 GAP Framework-Services

The GAP framework is built using a service-oriented architecture. The set of services that are the foundation of the GAP framework are called the GAP framework-services, the most important being the GAP service. The services are designed so that they perform distinct tasks. The clear separation of the functions of the services allows them to have cleaner and simpler interfaces. The GAP framework-services are Web services resource framework (WSRF) compliant and hosted in a WSRF container. WSRF is a set of specifications that provides means to construct stateful Web services and provides means to interact and manage state in a standard way. The current version of the GAP framework consists of three services as described in the next sections. In the future, additional services can be added to provide more value added services to the users.

3.1.1 GAP Service

The GAP service encapsulates an application and provides a service interface to that application. The service and the service interface are generic so that it can encapsulate any command-line applications. This generic abstraction provides scalability in the application provisioning process. This service is responsible for gluing an application to the Grid middleware to make it run on a Grid. It is also responsible for creating a user interface specification that is used by the portal to create dynamic user interfaces. It uses Grid middleware services to submit jobs and to maintain input and output files related to applications.

A single GAP service is capable of supporting many simultaneous application sessions. Each session state is encapsulated and isolated in a resource. However a single GAP service is not capable enough to support a large number of simultaneous sessions. This can be an application usage scalability bottleneck. In order to avoid this scalability

bottleneck, the GAP framework instantiates a pool of the GAP services. This pool of the GAP services is managed by the GAP factory service.

3.1.2 GAP Factory Service

The GAP factory service is the gateway to a pool of GAP services. This is a stateless service which only participates in setting up an application sessions. This service is responsible for managing the pool of the GAP services. All the application session requests first come to the GAP factory service which then selects a GAP service using the load balancing policies and sets up a session between the client and the selected GAP service. Before setting up the session, the GAP factory service authenticates and authorizes the session, assigns session IDs and stores session information to persistent storage. The architecture permits to have a number of the GAP factory services acting as gateway for different groups of clients. Further each GAP factory service could independently be configured with different policies related to load balancing and session management.

3.1.3 Application Registry Service

The GAP framework exposes applications as WSRF services to clients. As such, there is a need to expose the available applications so that clients can discover and choose applications. Since the GAP service interface is generic, it does not contain application-specific information. The application registry service acts as the custom application registry where clients can obtain information about the Grid-enabled applications. The application registry service provides information about the applications. The information may include a description, licensing information, cost if applicable etc. The application registry has been designed to categorize applications using a hierarchical structure. An

application may belong to a category which in turn may be a sub category of another category. The depth can be arbitrarily long as needed.

3.2 GAP Portal

The GAP portal is the front-end for the users. In addition to being the user interface to the GAP framework-services, the GAP portal may provide additional services to the users. The GAP portal is a portlet-based system. It consists of set of portlets hosted in a portal server. There are four portlets that are part of the GAP portal interface providing user interfaces to the GAP framework-services. These four portlets are layered on a single page. One of the guiding philosophies in designing the user interface has been usability and simplicity. Care has been taken to avoid unnecessary mouse clicks.

- Application portlet: This is the main portlet of the GAP portal interface. The portal interface is built around this portlet. The logic to construct the application specific user interfaces is contained in here.
- Progress portlet: The execution flow has four steps (initialization, mode selection, arguments and results). This portlet is used to show the steps involved in executing a job and to shows the current state.
- Message portlet: This portlet is used to show error, warning and informational messages produced by the GAP service.
- Utility portlet: This portlet lets users to upload files to be used by an application.

3.3 Actors

The architecture identifies three different actors participating in two different processes. The first one is the application provisioning process in which an application is made available to the GAP framework. The second process is the application usage process where a user uses the application. The distinct feature of the GAP framework is that it clearly separates roles of the system administrators, users and application provides without requiring one to know the others' skills.

Application specialists perform the application provisioning process, shown on the right hand side of Figure 3-1. They are expected to be very knowledgeable about the application to be Grid-enabled. However, they are not expected to know the technical details of the underlying Grid computing middleware. The application provisioning is a one-time process for each legacy application. In this process, an application specialist provides a description about an application. This application description includes general information, the description of inputs/outputs, the available "execution" modes for the application and information about execution environment required by the application. The application description is captured in an XML document and stored in the "Application Description" repository.

Users are clients who use of the legacy scientific applications. Users are expected to have a working knowledge of how to use the application they are invoking, but not the installation requirements of the application. Also, they are not expected to know about the underlying Grid computing middleware. Application usage (shown on the left hand side of Figure 3-1) is a recurrent process and is invoked every time a user requests an application.

System administrators are responsible for managing the Grid resources and portals. They have an in-depth knowledge of the Grid computing middleware and portal system. They are not expected to have in-depth knowledge of the applications that the Grid-computing portal provides. The architecture does not mandate a single administration of the portal, services and middleware. In fact, each of them can be managed by different autonomous administrators.

The following sections describes the processes (application provisioning and application usage) identified by the architecture.

3.4 Application Provisioning

In order to Grid-enable an application, the application first needs to be provisioned. The application provisioning process involves the following two steps illustrated in Figure 3-2.

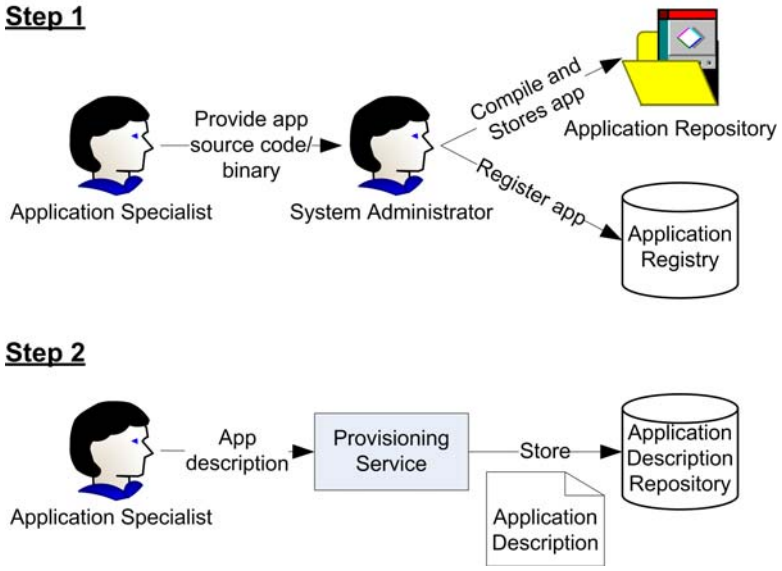

Figure 3-2. Application provisioning process. This process has two steps. In the first step, the system administrator places the application binary in the application repository and makes an entry in the application registry. In the second step, the application specialist provides the application description which then captured as an XML document and stored in the application description repository.

1. An application specialist requests the system administrator permission to Grid-enable an application. If approved, he/she then provides the application, the required libraries and samples in the form of binaries and/or source code. The system administrator then compiles the source code if necessary, tests them and stores the executable application in the application repository. Also he/she registers the application in the application registry. Compiling an application is a generic process and hence this step does not require any application specific knowledge from the system administrator.

2. The application specialist interacts with the provisioning service to provide the application description. The provisioning service captures the application

description in an XML document. This document conforms to the application description schema defined by this project (listed in Appendix B). This XML document then gets stored in the application description repository. In order to perform this step comprehensive knowledge about the application is required. However any knowledge about the Grid is not required. The system administrator does not get involved in this step. Once the application specialist tests the application, he/she requests the administrator to make the application available to the users.

Once these two steps are performed, the application is ready to be used through the portal. All the rest of the tasks required to bind the application with the Grid middleware and to create application-specific user interfaces are performed automatically by the GAP framework when the application is used.

3.5 Application Usage

A user who wants to use one of the Grid-enabled applications accesses the application through the GAP portal. Figure 3-3 shows the series of activities involved in the process of executing an application using the GAP framework. As can be seen a number of activities takes place when an application is executed on a computing Grid. However, the user does not see any of these Grid-related complexities. Hiding the complexities from the users enhances the usability of the GAP framework. The following are the activities that take place when an application is used.

1. User login: A user that wants to use an application first has to login to the GAP portal where he/she will be authenticated using the provided credentials (in this case a username/password pair).

2. Discover applications: Once a user logs in, the portal discovers the list of available applications for that user from application registry service.

3. Retrieve application information: The application registry service in turn obtains the application information from the application registry database.

4. Select an application: User the selects the application that he/she wants to execute.

5. Send selection: The portal then sends the user’s selection to the GAP factory service. The GAP factory service first makes sure that the user is authorized to use

the application. If the user is authorized, then the GAP factory service selects one of the GAP services for this application session. The selection of the GAP service can be controlled by appropriate policies.

6. Initialize the GAP service: The GAP factory service initializes the selected GAP service for the selected application.

7. Retrieve application description: As part of the initialization process, the GAP service obtains the application description from the repository. This description is the output of the application provisioning process. In addition, the GAP service creates a WSRF-resource to hold the session state. This WSRF-resource will be active till the end of the session.

8. Obtain interface specification: After the GAP service initialization, the end point address of the GAP service and the WSRF resource created to hold session context is send to the portal. Portal contacts the GAP service using the end point address to obtain the user interface specification for a customized version of the application. Using this user interface specification, the portal dynamically constructs the application user interface.

9. Provide application inputs: The user provides the inputs to the application using the user interface created by the portal.

10. Send application inputs: The user provided inputs are sent to the GAP service. The GAP service validates the user inputs, merges with default values as necessary and constructs the command to be submitted to the Grid middleware to execute the application.

11. Submit job: The GAP service contacts In-VIGO resource manager to submit the job.

12. Obtain results: Once the job is complete, the GAP service uses In-VIGO virtual file system to obtain the results of the job. This result is send to the portal to be displayed to the user.

In order to make the explanation easier, the above process ignores the existence of many customized versions of the application. When there are more than one customized versions of the application available to the user, there are some additional steps after the selection of application and before the interface generation. The user will be required to select one of the available customized modes of the application. Then a user interface for that customized mode will be generated for the user to provide application inputs.

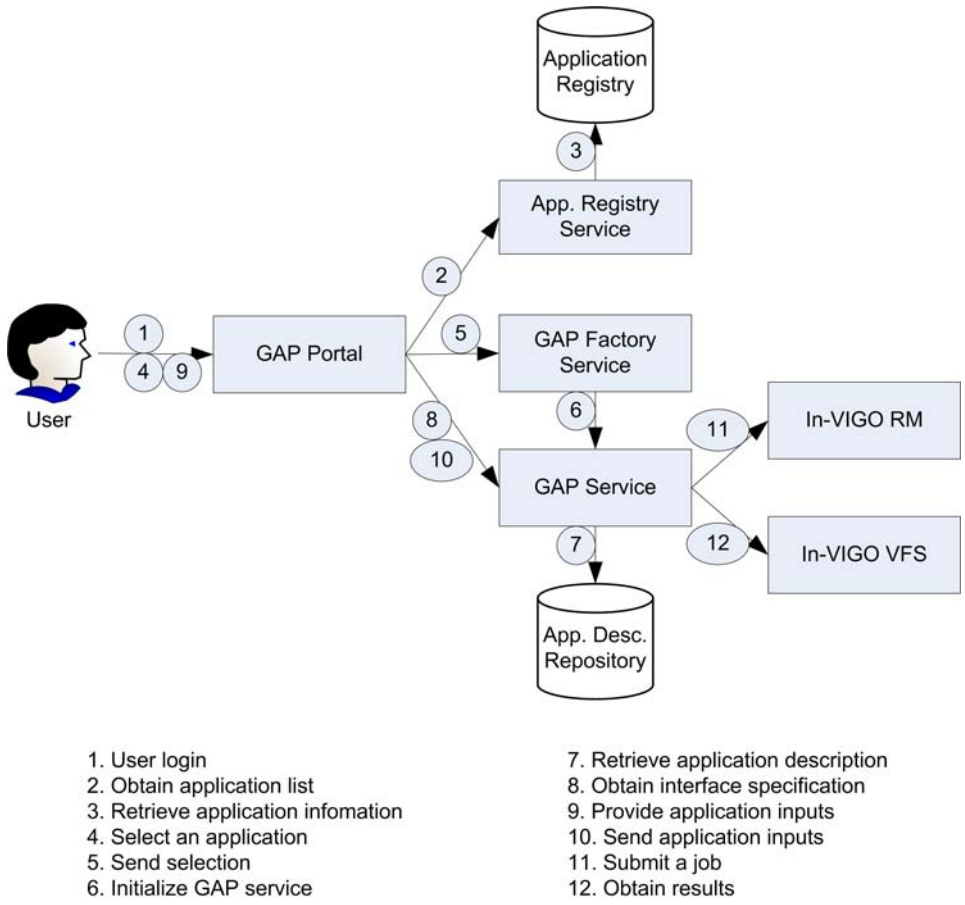

Figure 3-3. Application usage process. This diagram shows the sequence of activities that takes place and the components involved in the process of executing an application by a user. Although there are several activities involved, the user does not see them.

CHAPTER 4
IMPLEMENTATION

Chapter 3 describes the architecture of the GAP framework from the functional point-of-view. This chapter provides the implementation details of a prototype system. As in Chapter 3, this chapter will focus on the functional part of the implementation. The security architecture and implementation is described in Chapters 5 and 6 respectively. The following sections provide implementation details of the GAP framework-services and the GAP portal.

4.1 GAP Framework-Services

4.1.1 Generic Application Service (GAP Service)

As described in Chapter 3, a legacy application is invoked through the Generic Application Service (GAP Service). The GAP Service is generic because the same service is used for all command-line applications. When it is invoked for a particular application, it retrieves the corresponding application description from the repository (in the form of an XML document). The GAP Service is configured dynamically by the application description. Table 4-1 describes the GAP Service interface.

The sequence diagram shown in Figure 4-1 shows the interaction of the GAP service with other components of the architecture. When a client, in this case an application Portlet, wants to execute an application, it contacts the GAP factory service to obtain an end point of a GAP service. The GAP factory service selects and initializes a GAP Service (by calling the *createVA* method). GAPService.*createVA* method locates and parses the corresponding application description XML document for the selected

application. It also creates WSRF resource to encapsulate the session state. At this point the end point address of the GAP service combined with the resource is sent back to the application portlet.

The application portlet then interacts with the GAP service directly. From the description, the service obtains defined application modes and provides the modes to the client through a *getModes* call. The user, through the client, then selects one mode and the client asks for the arguments for that mode (*getArguments*). The service looks up the mode definition and selects relevant arguments and sends them to the client. The arguments contain interface specification for the selected mode of the application. This interface specification is used by the application portlet to construct a user interface.

The user provides appropriate values for the arguments and invokes the application by calling the *execute* method. The *execute* method validates the arguments and, if valid, creates appropriate directory structure in the virtual file system (VFS), constructs a command and schedules the job to be run using In-VIGO Resource Manager.

As soon as the execution is scheduled, the *execute* method will return. At this point, the user could continue to do other things. Once execution is complete, the client will be notified and it may get the results by invoking the *getResults* method. When *getResults* method is invoked, the GAP service obtains the *stdout*, produced by the execution of the application, using VFS.

At this point the user may terminate the session. When the session is terminated, the WSRF resource created to encapsulate the session is destroyed. If the user does not choose to terminate the session, the session can be destroyed when there are no activities for a predefined period of time.

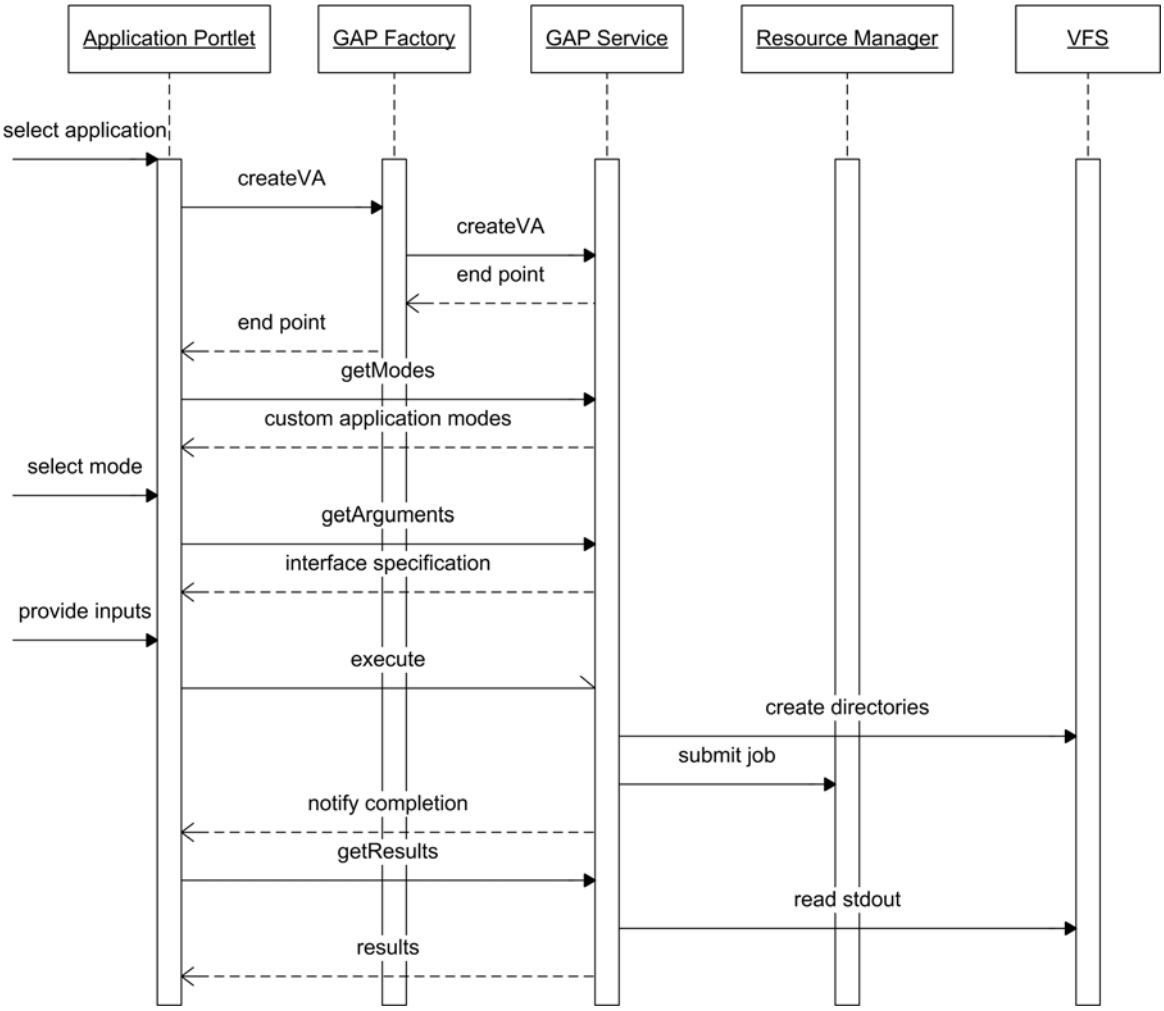

Figure 4-1. Interaction of the GAP service with other components of the architecture

Table 4-1. Generic application service interface

createVA()	Initializes the GAP Service with a specified application. It involves creating a WSRF resource to encapsulate the session state and obtaining and parsing appropriate application description.
getModes()	Returns all the defined modes for the application
getArguments()	Returns the argument description of all arguments of the selected mode. This is the interface specification for the selected mode.
execute()	Creates appropriate directory structure in the user's VFS account and executing the job through the resource manager.
getResults()	Provides the results of the execution. The result in this case means the stdout. If the application generates additional files, that can be accessed via the file manager.

The GAP service is implemented as a WSRF service. It is capable of handling multiple application session simultaneously. For each application session it creates a resource to encapsulate the session state. Figure 4-2 shows the component architecture of the GAP service. The application description parser module is an XML parser. In addition to parsing the application description XML document, it performs translation of keywords. These keywords are used to mask Grid specific information such as paths from the application description. The validation module validates user inputs using Java regular expressions. The job module interacts with In-VIGO Resource Manager to submit a job to a remote compute resource. The file system module interacts with In-VIGO virtual file system service to provide input files for applications and to read output files.

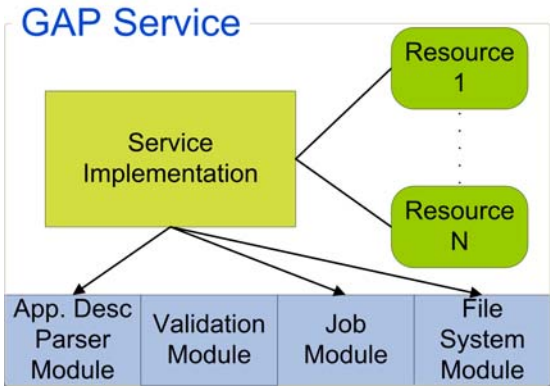

Figure 4-2. GAP service architecture

4.1.2 Application Description

The GAP Service is driven by an application description. The application description is an XML document conforming to an XML schema described in more detail in this section. The complete schema is listed in Appendix B.

The application description is categorized into three categories, as shown in Table 4-2: general information, input/output description and execution environment-related

information. The general information includes name, description, application version etc. The input/output description is the main part of application description. This is organized as a set of arguments, each of which is a collection of parts that are logically related and hence have to appear together when executing the application. A part is a description of a piece of user input and associated flag if there is one. Arguments and the corresponding parts are used to construct the command that is issued to execute the application. They also include information to validate user inputs and provide default values. The execution environment information describes the requirement on the execution environment. This information is used to select or construct the execution environment at run time.

Table 4-2. Description of applications

General information about the application	Application Name
	Description
	Version
Description of inputs and outputs, organized as set of arguments, each of which contains one or more parts	Flags
	Input data types
	Cardinality
	Descriptive names
	Validation Information
	Default Values
	Dependency
Execution Environment	Operating System
	Required libraries
	Directory structure
	Memory
	Bandwidth
	Environment Variables

An XML schema (listed in Appendix B) has been defined to accommodate the above-mentioned application description. Each application is specified by one XML document conforming to the schema definition. The schema is constructed in such a way that it is easy to define simple applications and, at the same time, powerful enough to specify complex applications. Note that the application description does not contain any

Grid-specific information. Where they are needed (for example paths), key words are used to mask such information. These keywords are mapped to actual values by the GAP service when an application is used.

Using the application description, the GAP service is able to perform the following validations:

- Validation of input values before an execution starts to increase the probability of successful completion of the execution. This is important for long-running applications or for applications that form a part of a workflow. A method based on regular expressions is used to validate the input values.
- Validation of the input dependencies by grouping them together in an appropriate sequence.

Table 4-3. Excerpts from Dinero description

Executable Information
<name> Dinero IV</name> *<binary> dineroIV</binary>* *<description> Cache Simulator </description>* *<path>APP_HOME/dinero</path>* …
Application Input/output description
<argument name="kind" minOccurs="1" maxOccurs="5"> *<part flag="-l">* *<description>Level</description>* *<defaultValue>1</defaultValue>* *</part>* *<part flag="-">* *<description>Type</description>* *<defaultValue>u</defaultValue>* *</part>* *<part flag="size">* *<description>Size</description>* *<defaultValue>16k</defaultValue>* *</part>* *</argument>* …
Execution Environment
<env> *<os>Linux</os>* ... *</env>*

Table 4-3 shows excerpts from the application description for Dinero (complete document is listed in Appendix C). The executable information section shows the usage of keywords. In this case, path where the application is installed is masked by the keyword "APP_HOME". This will be translated to actual path by the GAP service when the application is executed. The application input/output description section shows an example definition of an input. As can be seen, it is a multi-part argument ("kind") containing three different parts, namely level, type and size of the cache. Without specifying level and type, the size specification is incomplete. Dependencies among inputs are enforced by grouping inputs together in the appropriate sequence.

A feature called "Execution Mode" has been defined in the application description. An execution mode of an application is a customized version of that application. Some complex applications have numerous input parameters. A user performing some simulations may not be interested in all of them. For example, suppose it is desired to Grid-enable Dinero for use by both researchers and students. A researcher would like to have the full capabilities. On the other hand, a student who is new to the application would like to have a simpler interface to make the learning process easier and faster. From the administrator's point of view, it is necessary to limit the capabilities of the application given to the students in order to avoid a large number of simultaneous compute-intensive jobs. Each user group can be given a customized version of Dinero by defining different modes. Table 4-4 shows an example mode definition for Dinero.

As can be seen in Table 4-4, a mode is a collection of arguments. These arguments are already defined in the application input/output description. Hence only the name of an argument needs to be specified in the mode definition. However, one can modify the

argument definition to a limited extent in the mode definition. For example, the *"kind"* definition in table 3 allows up to 5 cache levels (*maxOccurs = 5*). But the basic mode definition in table 4 limits the *"kind"* to only 1 level (*maxOccurs=1*). Also, it is possible to include an argument for which a user does not need to provide values. This can be done by setting the *useDefaultValue* to true. An argument with this setting will assume the default value. Other arguments will assume the user provided values.

Table 4-4. Dinero basic mode

```
<mode name="basic">
    <argument name="informat" useDefaultValue="true"/>
    <argument name="blocksize" maxOccurs="1"/>
    <argument name="kind" maxOccurs="1"/>
    <argument name="statcombine"/>
    <argument name="computecapacity" maxOccurs="1"/>
    <argument name="stdin" useDefaultValue="true"/>
    <argument name="stdout" useDefaultValue="true"/>
</mode>
```

The application description documents are stored in an application repository. In the current implementation, the application repository is directory in the VFS. The path of the application description repository is provided to the GAP service through its JNDI configuration file. In the future implementation, the repository can be an XML database.

4.1.3 GAP Factory Service

The GAP factory service is the gateway to the pool of GAP services. It is a stateless service. It only gets involved in setting up the application session. Once the application session is setup, it no longer participates in the session. Having a limited involvement in an application session makes it capable of handling a large number of sessions. Further failure of the GAP factory services does not interrupt existing sessions. Only disruption is that no new session can be created. Those characteristics are very necessary given the GAP factory's role as gateway to the GAP services. Its primary functions are session

management, load balancing and access control. For each session, the GAP service assigns session number and records the session information in a database. This information can be used in the accounting function. The GAP factory service manages a pool of GAP services. It is responsible for managing the load among the GAP services. Current implementation only supports round-robin method of load balancing. Future implementations can add other methods of load balancing. The implementation of the access control function of the GAP factory service is described in the Chapter 6. The GAP factory service interface exposes only one operation called *createVA.*

4.1.3 Application Registry Service

The application registry service is a service-oriented front-end to the application registry. The application registry of the GAP framework is implemented as a relational database. It classifies and organizes the application in a multi-level hierarchical structure. In the current implementation, the application registry is defined by a simple database schema. Future versions may extend the schema as required.

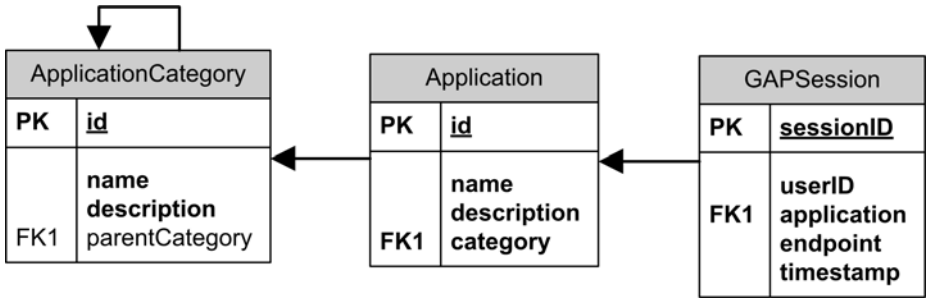

Figure 4-3. Application registry database

Figure 4-3 shows the E-R diagram of the database. *ApplicationCategory* table defines various application categories. As can be seen, an application category can be a sub category of another category. (e.g. *Cache Simulators* can be a sub category of *Computer Architecture*). Further, this structure can be used to define suite of applications

(e.g. *sim-fast* application belongs to *Simple Scalar* suite which in turn belongs to *Computer Architecture* category). *Application* table defines the applications. The *GAPSession* table defines session information.

The application registry service is also stateless and exposes only one operation called *getApplicationInfo*. This operation returns application information organized as a hierarchy.

4.1.4 Configuration of the GAP Framework-Services

The configuration parameters for the GAP framework-services are provided through JNDI configuration file. Each service has its own JNDI configuration file. In addition, global configuration parameters can also be defined through the container's configuration file.

4.2 GAP Portal

Portlet technology is used to automatically and dynamically generate user interfaces for Grid-enabled applications. Portlets are Java-based web components, managed by a Portlet container that process requests and generate dynamic content. Portlets are used by portals as pluggable user-interface components that provide a presentation layer to information systems [11, p.13].

The choice of Portlet technology for this project is influenced by many factors. Portlet specification is an open standard supported by most of the portal server vendors. Portal API attempts to create a well-defined interface and hence ensures interoperability between different portal servers. Portlets also provide better support for multiple devices. One can define different JSP pages for different devices for the same Portlet and hence support multiple devices while reusing the logic programmed in the Portlets. Portlets provide a modular structure. A web page is created by aggregating Portlets. Users have

their choice of what Portlets to aggregate and what layout to use. This enables users to have personalized user interfaces. Also, administrators have the liberty of organizing permission on per Portlet basis which gives them finer access-control granularity. The GridSphere [12] portal server is used in this project.

The novel aspect of the user interface design in this project is that the structure of the user interface is dynamic. The structure is determined by analyzing the description of the arguments in the application description for the selected mode. Then widgets like text boxes, combo boxes, labels, etc. are rendered accordingly.

The current version of the user interface consists of four Portlets: Application portlet, Message portlet, Progress portlet and Utility portlet. The Application portlet is the main portlet and is responsible for the dynamic user interface generation as described above. The Progress portlet show the steps involved in executing an application and to displays the progress of the execution. The Message is used to convey messages generated by the GAP Service. The Utility portlet provides means to upload files. This is needed because many applications require an external file. Portlets use JSP pages to render the visual components. This design conforms to the MVC (model-view-controller) design pattern. The data obtained from the GAP service represents the model. JSP pages are the views and the portlets play the role of controller. There are four states involved in executing an application.

- Init State: A user initializes an application.
- Mode State: The user selects one of the customized modes of the application.
- Argument State: The user provides the necessary inputs to the application.
- Result State: The user obtains the result of the execution of the application.

The portlet interface is designed as a state machine so that a user can go back and forth to correct any errors he/she makes. Figure 4-4 shows a dynamically created

interface for Dinero, a cache simulation application, for two different modes. The left-hand side of the figure shows application portlet for a simplified version of the Dinero application. The right-hand side of the figure shows the application portlet for the full fledged version of the Dinero application.

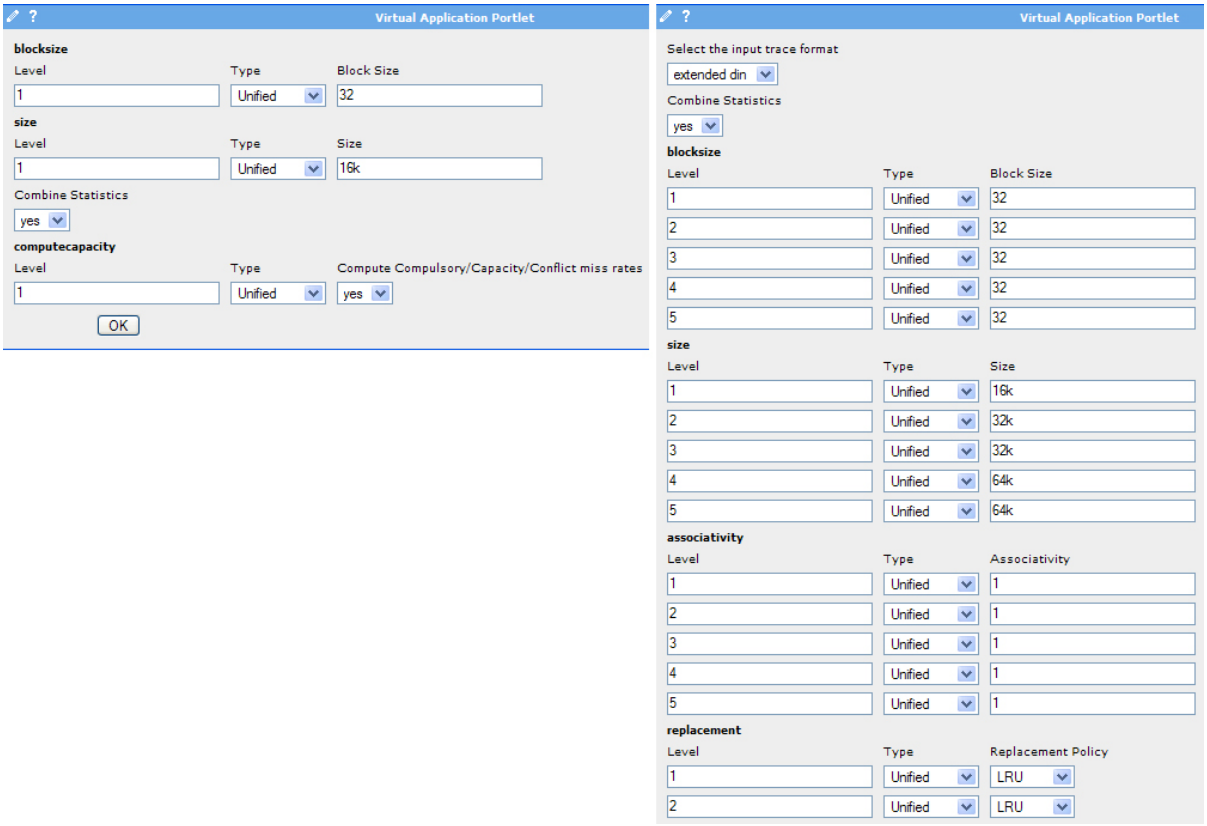

Figure 4-4. Dynamically generated interfaces of Dinero for two different modes. The left hand side shows the interface for a basic mode. The right hand side shows the interface for an advanced mode.

CHAPTER 5
SECURITY ARCHITECTURE

As in any computer system deployment, security is critical to the successful deployment of the GAP framework. The distributed nature of the GAP framework poses a number of challenges to the design of the security architecture. In addition to the typical security issues like authentication and authorization found in non distributed systems, issues related to secure communication have to be addressed in a distributed system like the GAP framework. The loose coupling of the entities of the GAP framework that belong to different security domains further complicates the problem, making it even more complex than the security architecture for a single domain distributed system. In such multi-domain systems, one has to address issues like trust, single-sign-on, security management and distributed authentication and authorization.

The GAP framework employs a service-oriented architecture. In order to fully utilize the benefits of the service-oriented architecture of the GAP framework, the security architecture must use widely accepted standard-based technologies. This chapter describes the GAP security architecture that addresses the security challenges of multi-domain system, using widely accepted standard technologies. The emphasis is on addressing the problem of having multiple security domains. The novel aspects of the proposed architecture are (1) the extended form of identity federation which incorporates the user attributes in addition to the user identity and (2) the distributed context-aware role-based access control mechanism. Liberty Alliance [13] defines specifications and architectural models for identity federation. However identity federation is used mainly to

provide a single-sign-on capability. The federation used in the GAP framework provides a basis for distributed, context aware access control in addition a single-sign-on capability.

As described earlier, the GAP architecture consists of the GAP portal, the GAP framework-services and the Grid middleware. These entities are independent and may well belong to different security domains. In addition, they may have different security infrastructures employing different technologies. For example, the GAP portal may use a username/password based mechanism. The GAP framework-services may use a public key infrastructure (PKI) based mechanism. The middleware may use a symmetric key based mechanism. Further, these entities may have to communicate with each other over a public, unsecured network. Figure 5-1 depicts the security problems that need to be addressed by the GAP framework.

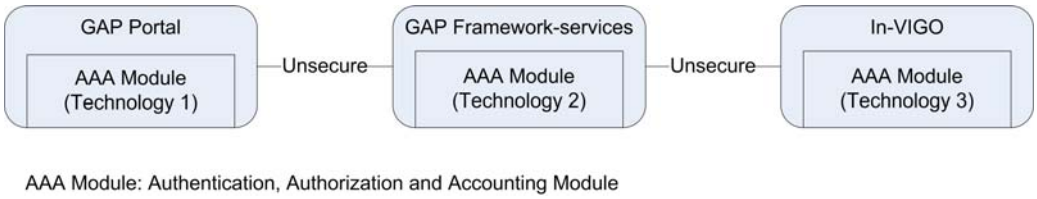

Figure 5-1. Security problem of the GAP framework. As can be seen, each entity is independent and has its own security domain. Each entity may employ different security technology to address authentication, authorization and accounting issues. The communication links between the entities are unsecured.

In order to address the above mentioned issues, this chapter describes a distributed security architecture that:

- Provides single-sign-on for the end users.
- Does not require changes to existing security mechanisms employed by individual entities.
- Lets the resource owners maintain access control over their resources.

- Simplifies security management in the sense that it does not require excessive coordination between the entities in order to perform security related tasks like adding a user, adding a resource, changing resource access policies etc.

The proposed solution extends the identity federation technique to include the user attributes in addition to the user identity to build the distributed security infrastructure. The attribute federation technique supports multiple user classes to be federated. Further, a distributed, context-aware role-based access-control mechanism is proposed to deal with the distributed authorization issues in the presence of multiple, federated user classes. This technique simplifies the security management while letting the resource owners to have access control over their resources. Although this architecture is developed in the context of the GAP framework, it has general applicability for a service-oriented system that spans multiple security domains. Appendix A describes various functions of security architecture for a distributed, multi-domain system. The following section describes how those functions are realized in the GAP security architecture. The subsequent sections discuss use-cases and advantages of the GAP security architecture.

5.1 GAP Security Architecture

The GAP framework consists of independent entities (GAP portal, GAP framework-services and Grid middleware) that communicate over an unsecured network. The GAP security architecture uses the federation technique to loosely-couple GAP portal, GAP framework-services and Grid middleware, without requiring modifications to the existing security mechanisms.

Figure 5-2 shows the GAP security architecture. The foundations of federation are secure communication and trust. Figure 5-2 shows how these primitives are enabled between the entities. Using the federation, distributed authentication and authorization

functions are realized. The following sections describe how the GAP security architecture realizes the various security functions outlined in Appendix A.

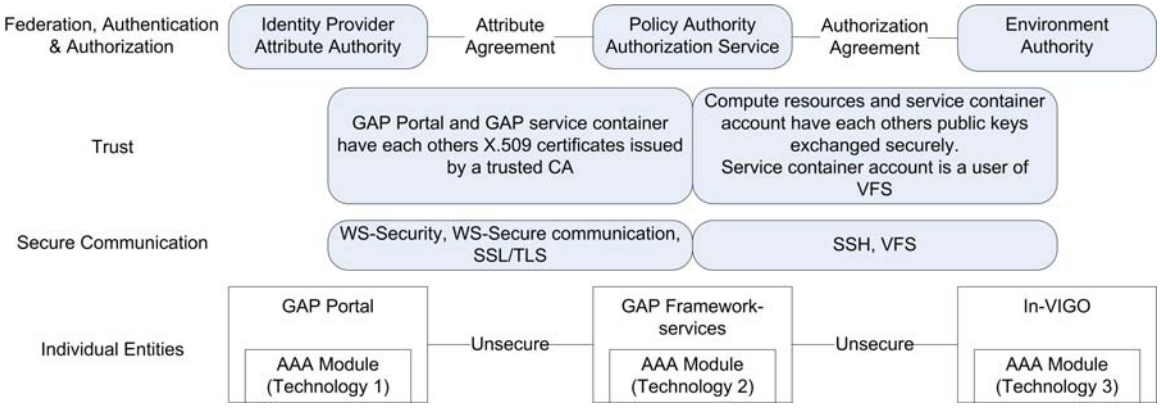

Figure 5-2. GAP security architecture. This figure shows how trust, secure communication and federation are constructed among the entities. Secure communication and trust are the foundation of federation. The federation is used in the distributed authentication and authorization functions.

5.1.1 Secure Communication

The GAP portal and the GAP framework-services communicate using SOAP messages. Therefore, secure communication mechanisms for Web services can be applied between the GAP portal and the GAP framework-services. WS-Security, WS-Secure communication and SSL/TLS are the different mechanisms that are provided. These mechanisms can be used to provide different levels of protection for the messages that are exchanged. (Messages can be encrypted, digitally signed or encrypted and digitally signed.) The choice of the security mechanism and the level of protection applied are configurable parameters. These configuration parameters are provided through security descriptors on both client and server side. The GAP framework-services and the In-VIGO Grid middleware use SSH security mechanism. This is because the In-VIGO Grid computing middleware is not yet service oriented.

5.1.2 Trust

In order to form a trust network, some trust relationships need to be preconfigured at the configuration time. In the GAP framework, trust relationships between GAP portal and GAP framework-services and between GAP framework-services and Grid middleware are preconfigured.

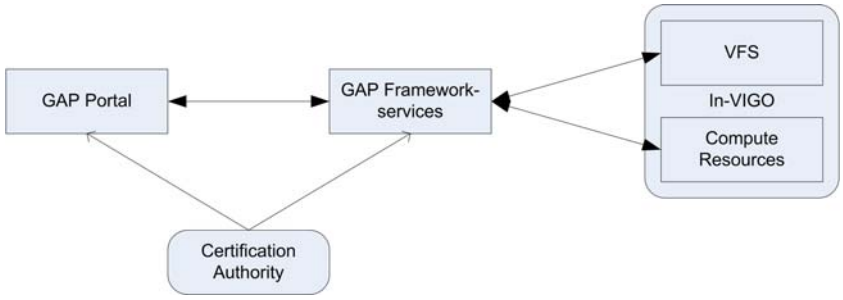

Figure 5-3. Preconfigured trust relationships in GAP framework. Trust relationships between the GAP portal and the GAP framework-services and between the GAP framework-services and the In-VIGO middleware are preconfigured.

Figure 5-3 shows the preconfigured trust relationships in the GAP framework. The GAP portal and GAP framework-services establish a trust relationship by sharing their X.509 certificates, issued by a trusted certification authority. When messages are exchanged between the GAP portal and GAP framework-services, the proof-of-possession method is used to evaluate the trust relationship. The messages are either encrypted or digitally signed according to the WS-Security specification. In this trust relationship the GAP portal takes the responsibility of faithfully asserting the identities and attributes of the user. The GAP framework-services in turn take the responsibility of honoring those assertions and let the users to execute the applications that are available to them.

The In-VIGO Grid middleware consists of two primary types of resources from the perspective of security architecture. They are the compute resources and the virtual file

system (VFS) service. The term compute resource means a single computer, a cluster or a network of computers. The GAP framework-services and the Grid middleware may establish trust relationships by two different mechanisms for each of the two types of resources. Here again the method of evaluation of trust relationship is the proof-of-possession. In this trust relationship the GAP framework-services take the responsibility to faithfully apply the access control policies and to assert authorization decision. The In-VIGO middleware in turn takes the responsibility of honoring the assertions and provides compute and VFS resources.

5.1.3 Identity Provider

The users of the GAP framework need an identity. The GAP portal acts as the identity provider for the users. User identities are provided in the form of username/password pairs. This username/password pair is local to the portal and not recognized by the other entities in the GAP framework. Attribute assertions are used to federate the users so that other entities recognize them. The portal administrator may decide the policies on how the user's credentials are ascertained. It may be based on an existing identity (GatorOne, SSN etc.) or may be based on other methods. Although the current architecture uses a portal server which uses username/password pair to provide user identities, the architecture is not limited to the use of username/password pair. In fact, the architecture can support a portal that uses advanced methods like biometric identities. The choice of the identity provision method does not, in any way, affect the other entities in the architecture.

5.1.4 Attribute Authority

The users generally have attributes. These attributes are the ones that differentiate different classes of users. A typical RBAC solution to a single domain system binds the

users to roles at the time user identities are provided based on user attributes. Thereafter the user attributes do not play a part. However in a system like the GAP framework where the identity provider and access control functions are likely located in different security domains, the early binding of identities to roles is not desirable. Each security domain may want to assign different roles to the same user, which is an internal decision of that security domain. Each security domain may be interested on a set of user attributes for role assignment that is different from the attributes requested by another security domain. Further, the entity that makes access-control decisions may not want to expose its role-hierarchy to the identity provider. Hence the GAP framework provides an attributes authority which faithfully asserts the user attributes. The other entities can use the attributes to assign roles to users dynamically. In the current architecture the GAP portal performs the function of attribute authority.

5.1.5 Authentication

The GAP portal authenticates users when they log-in to the portal by validating the username/password pair. In order to provide a single-sign-on capability to the users, once a user is authenticated, other entities must be able recognize him/her without requiring to authenticate again. This is made possible by federating identities.

5.1.6 Federation

The federation between trusted entities in the GAP framework is achieved by exchanging assertions. In a typical federated-identity management system, the issuing party issues an authentication assertion about an authenticated user. The authentication assertion typically contains the subject, authentication method, conditions under which the assertion is valid and the lifetime of the assertion. In the GAP security architecture, the identity federation is augmented to include attributes of the users. This enables the

issuing party to differentiate different classes of users. The set of attributes that need to be asserted are agreed upon by the issuing and relying parties in advance. The GAP portal, being the identity provider and attribute authority, is the issuing party. The GAP framework-services are the relying party. The attribute assertions play a part in the proposed distributed authorization mechanism.

5.1.7 Authorization

Authorization is the process of managing access control to objects (in this case applications) by subjects (in this case the users). Simply stated, the authorization function of the GAP security architecture is to decide the following question.

Is *user A permitted to use application X?*

The role-based access-control (RBAC) [14] is the most common technique used manages access control. RBAC assigns users to roles which generally reflect their position in the organization. A role is usually assigned a set of permissions by the administrator. The main advantage of using RBAC is its ability to greatly simplify the security administration.

Figure 5-4. Role based access control. The roles serve as an intermediary to bring users and permissions together.

Figure 5-4 shows a simplified RBAC model. As shown in the figure, a role is both a collection of users on one side and a collection of permissions on the other. The role serves as an intermediary to bring these collections together. In the GAP framework, a set of permissions can be defined on applications and other resources like data and the

permissions and the users can be linked by roles. Additional features of RBAC such as role hierarchies and constraints can also be used to enhance the access control model.

The GAP framework will benefit from using a context-sensitive RBAC [15-17]. Context information may be applied to the user assignment or permission assignment or both. Keeping the context information out of access control policies makes the administration easier. The following illustrates how the use of context can make the security policy administration easier. In this illustration, context is applied to the permission assignment. A security policy can be specified as:

Students who are on the *university network* can **execute applications** *during the class time* if the resources are not *heavily loaded.*

Where:

Role: Students

Permission: Execute applications

Contexts: on the university network, during the class time and heavily loaded

The above security policy will remain the same even if the IP addresses of the university network are changed or this class time is changed or the definition of heavily loaded resource is changed. The context can be classified as subject context (e.g. on the university network refers to subject), object context or environment context (e.g. heavily loaded refers to the environment). Table 5-1 shows some of the relevant context information that need to be considered when making access control decision in the GAP framework.

Table 5-1. Some relevant context information for access control

User (Subject) context	**Application (object) context**	**Environment Context**
Location	Licensing requirements	Time/Day
Authentication method		System load

Even the context-sensitive RBAC does not completely solve the access-control problem of the GAP framework. The multi-domain nature of the GAP framework introduces additional problems.

- The users are defined by the GAP portal. The execution environment is managed by the In-VIGO middleware. The applications are managed by the GAP framework-services. The authorization decision is made by the GAP framework-services that do not know the users. If the GAP portal were to expose the user information to the GAP framework-services then there would be a tight coupling between the GAP portal and the GAP framework. For every user addition, deletion and modification, the GAP portal and the GAP framework-services have to collaborate.

- There will be many different classes of users that will use the applications. Since the GAP framework-services that make the access control decision do not know the users, by extension they do not know user classes as well. If the user class information is not incorporated into the access control decision, all the users who come through the GAP portal will be treated as having same roles and hence same access rights. That is not flexible. If the GAP portal is be given the user-role assignment responsibility, then the GAP framework-services will have expose its role hierarchy.

The proposed solution clearly partitions the authorization function giving each entity their share of authority without loosing autonomy. The federation is used to bridge the partitions to make the authorization decision. Figure 5-5 illustrates the process of mapping users to roles.

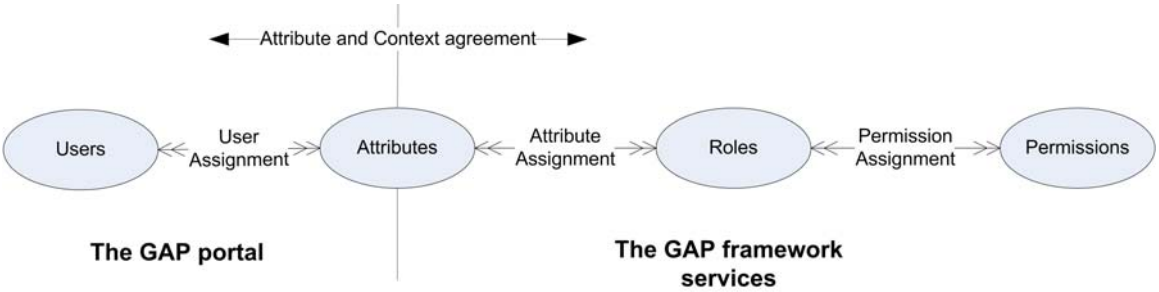

Figure 5-5. Extended RBAC. Instead of directly mapping users to roles, the mapping is dynamically performed indirectly through user attributes. Although this indirection adds additional layer of complexity, it greatly simplifies security administration in a multi-domain distributed system.

RBAC was invented in the context of single organizational security administration. In a single organization, the user attributes had a very close relationship with the roles defined by the security infrastructure. For example, a manager would have the manager role. When a user is given an identity his/her attributes are taken into account to assign a role. Once the role is assigned the attributes are not considered in the security functions.

However, in a multi-organizational context with separate security domains, it is not possible to assign roles at the time of identity provisioning because the identity provider and access control functions may belong to two different security domains. In such situations it is useful to separate the user attributes and roles. When a user is federated, his/her attributes are taken into account to assign the relevant roles. In other words, the users are bound to the roles dynamically when the user accesses a resource. This arrangement provides many advantages to the GAP framework as described below.

1. The GAP portal and the GAP framework-services can agree on the attributes and the context that are relevant for access control decisions. The set of attributes and the context parameters are fairly static and do not change frequently. This provides a looser coupling between the entities compared to an agreement on user identities.

2. The user attributes are completely user dependent. Hence the GAP portal can assign relevant attributes to users without knowing the role definitions.

3. The GAP framework-services can easily use any context-sensitive RBAC technique. Instead of mapping users to roles, it can map attributes to roles.

5.2 Use-cases

5.2.1 A User Executes a Grid-Enabled Application

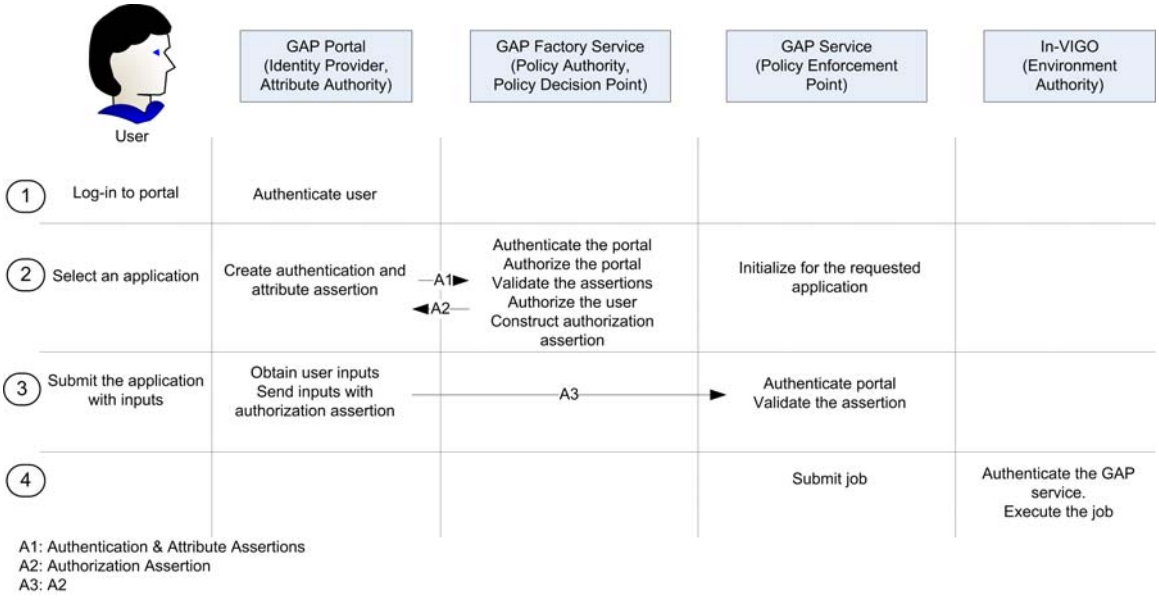

Figure 5-6. The process of a user executing a Grid-enabled application. As can be seen at each step along the way several security-related tasks have to be performed by various components.

Let's consider the process of a user executing a Grid-enabled application. Figure 5-6 shows the steps involved in achieving the task from the security point of view.

1. A User logs into the GAP portal using the credentials (a username/password pair) provided to him/her by the portal (identity provider). The portals authentication module authenticates the user. If the user is authenticated successfully, he/she is able to log-in to the portal. At this point, the other entities are not involved.

2. The user now selects an application to execute on the Grid. The federation agreement between the GAP portal and the GAP framework-services dictates that the portal provide authentication assertion and certain attribute assertions about the user. The GAP portal constructs those assertions, contacts the GAP factory service on behalf of the user and sends the assertions with the request to execute the application. The GAP factory service authenticates the portal using the credentials used to establish trust relationship between them. It makes sure that the portal is authorized to issue the attribute assertions. It validates the assertion, extracts the attributes and authorizes the user based on his/her attributes. If the user is authorized, a GAP service is initialized and its end point together with the authorization assertion is sent back to the GAP portal.

3. Now the GAP portal interacts directly with the GAP service assigned for this session. With each message, the portal sends the authorization assertion. The GAP service extracts and validates the authorization assertion. If it is valid, the GAP service provides the requested service.

4. Now the GAP service accesses the In-VIGO on behalf of the end user. The In-VIGO authenticates and authorizes the GAP service. In the current architecture, the In-VIGO accepts all the decisions made by the GAP service. Hence the authorization assertions are not sent to the In-VIGO. In some sense, the GAP service and the In-VIGO are tightly coupled. However, when the In-VIGO becomes service oriented, the same mechanisms between the portal and services can be applied between the GAP service and the In-VIGO.

Note that the exact identity of the user is not passed around. Only the attributes of the user is used to make authorization decision. When a unique identity is required for a user, a pseudonym can be assigned to the user to uniquely identify him/her.

5.2.2 Adding/Removing a User

Because of the use of the federation, the process of adding/removing a user is fairly simple. Every user has only one identity provided by the identity provider (the GAP portal). When a new user needs to be added, the GAP portal ascertain the credentials of the user and his/her attributes and issues a username/password pair and records his/her attributes. When a user needs to be removed, only his/her issued identity needs to be invalidated in the GAP portal. Note that the other entities are not involved in these processes.

5.2.3 Adding/Removing a Resource

When the In-VIGO wants to add a new resource it can do so easily. As long as the GAP service is able to access the new resource with its credentials, the In-VIGO is free to add a new resource. In the case of removing a resource, the In-VIGO may do so without any problems.

5.2.4 Changing Authorization Policies

The GAP factory service is free to change the authorization policies without needing to change configurations in the other entities. Only when the GAP factory service wants to change the set of attributes on which it bases the authorization decision it has to redefine the agreement with the GAP portal. In this case, the GAP portal needs to issue attribute assertions for the new set of attributes. However it should be noted, change of attributes on which the decision is made is very infrequent. In most cases, the policy changes usually involve mapping of attributes to roles or definition of roles.

5.3 Advantages of the GAP Security Architecture

- Users are able to single-sign-on and obtain services.
- It does not require changes to existing security mechanisms. This is very useful when extending the architecture as described below.
- The entities are loosely-coupled. The agreements between the entities do not change frequently. Hence the security management is easier.
- Every entity has control over its resources. The GAP portal has full control over the users it manages. The GAP framework-services have full control over the applications it manages. And the In-VIGO middleware has full control over the resources (computers, networks etc.) it manages.
- The architecture can be extended in various ways because the GAP portal, the GAP framework-services and Grid middleware are independent and autonomous entities. Figure 5-7 shows the possible extensions.

    a. The GAP portal can aggregate services from many different GAP framework-services groups (as shown in the left-hand side of Figure 5-7). With each group, the portal reaches an agreement on the relevant set of attributes and context information. As long as the portal can provide assertions of the attributes and context information, this agreement will work. An agreement with one service group does not affect the other service group in any way.

    b. A GAP framework-services group can interact with multiple portals (as shown in the right-hand side of Figure 5-7). For each portal, the GAP framework-services can have different security policies by providing a dedicated GAP factory service for each portal client. This dedicated GAP

factory service will implement the specific policies of the agreement. As in the first case, each portal will agree on the set of attributes and context information to be used in the access control decision.

c. A combination of the above two can also be configured where each portal interacts with many service groups and each service group interacts with many portals.

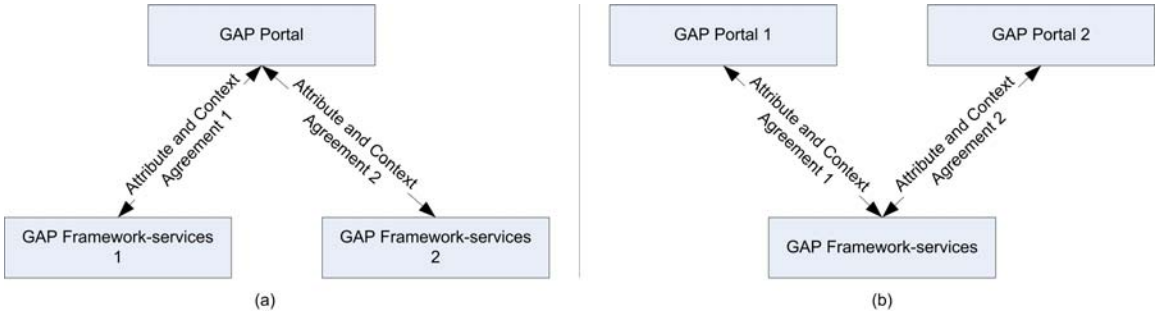

Figure 5-7. Possible extensions to the GAP architecture. (a) The GAP portal may interact with many groups of the GAP framework-services. (b) One group of GAP framework-services may interact with many portals. (c) Combination of (a) and (b) can also be supported.

CHAPTER 6
SECURITY IMPLEMENTATION

As described in Chapter 5, the security architecture consists of several functions and these functions are layered and distributed. This chapter describes how the architecture is implemented in a prototype system. One of the guiding principles is to separate the application logic from the security functions as much as possible so that changes to one does not seriously affect the other. The following sections describe the implementation details of the various security functions.

6.1 Secure Communication

As mentioned in Chapter 5, Globus toolkit 4.0 (GT4.0) [12] is used as the WSRF [5] container for the GAP framework-services implementation. GT4.0 comes with built-in support for secure communication. It supports message level and transport level secure communication. At message level, two standards are supported: WS-Security and WS-Secure communication. At transport level, HTTP over SSH/TLS (HTTPS) is supported. At both level, messages can be signed (integrity protection), encrypted (privacy protection) or signed and encrypted. Further, the choice of secure communication method (WS-Security, WS-Secure Communication or SSH/TLS) and the level of protection (integrity, privacy or both) can be specified per Web service operation. For each service, the values for the configurable parameters are provided via a security descriptor. Table 6-1 shows excerpts of the security descriptor for the GAP service. As can be seen the method *"createVA"* has been secured by transport level security and it is both integrity and privacy protected. Similarly other methods are also protected. Instead of specifying

method-by-method, one may provide one specification that applies to all unspecified methods. This way of specification is shown at the bottom of the excerpts shown in Table 6-1. In this case, all unspecified methods are not protected (auth-method, none).

Table 6-1. Excerpts of the security descriptor for the GAP service. The security configuration can be applied per operation basis or commonly for all unspecified operations.

```
<securityConfig xmlns="http://www.globus.org">
   <method name="createVA">
      <auth-method>
         <GSITransport>
            <protection-level>
               <integrity/>
               <privacy/>
            </protection-level>
         </GSITransport>
      </auth-method>
   </method>
   …
   <auth-method>
      <none/>
   </auth-method>
   <authz value="gridmap"/>
</securityConfig>
```

The Globus toolkit security infrastructure uses public key infrastructure. The private key and the certificate of the WSRF container are provided through the container's security descriptor. Alternatively, each service may have its own key that can be specified as part of the security descriptor. For this implementation, the container's key and certificate are shared by all the services deployed in that container.

On the client side, corresponding security configuration should be provided. The security configuration of the client must match the security specifications of the server. For example, if the server expects to use WS-Security with integrity and privacy protection then the client must be configured to do so. Client side configurations can be

provided either through security descriptors or programmatically. In this implementation, a client-side security descriptor is used. This security descriptor includes the client's key and certificate information as well as the secure communication method and the level of protection. The syntax of client-side security descriptor is similar to the one on the server-side.

Communication between the GAP framework-services and the In-VIGO compute resources is secured using SSH. GAP service uses In-VIGO resource handler APIs to submit jobs to compute resources. The resource handler APIs in turn use SSH to submit job to compute resources. All the compute resources are configured to have the public key of the user account on which the GAP service is deployed. The user account on which the GAP service is deployed is configured with host keys of all the compute resources.

6.2 Trust

The trust between the GAP portal and the GAP framework-services is established by policy agreement and sharing certificates issued by a trusted certification authority. For this implementation, a certification authority has been setup to issue certificates. The Globus toolkit provides a tool called simpleCA to setup a simplified certification authority. The simpleCA tool is deployed as the certification authority for this implementation. Note that the implementation does not mandate the use of simpleCA. Any other certification authority can be used to issue certificates without changing the implementation.

6.3 Identity Provider

In the GAP framework, the GAP portal is the identity provider to the end users. The most important task in providing identity to a user is to ascertain the credentials of a

potential user. It is a human task. It is to be performed by someone who has the portal administrative privilege. Once the credentials are ascertained, the administrator can log-in to the portal and create a user using the user administration portlet provided by the portal. For this implementation, the Gridsphere [18] portal server is used. User identities are provided in the form of username/password pairs. The Gridsphere portal stores the user identities in a database. In this implementation, the Gridsphere portal is configured to store the user identities in a MySQL [19] database.

6.4 Attribute Authority

In the current implementation, the GAP portal functions as the attribute authority. As in the case of identity provider, the main task of the attribute authority is to ascertain the attributes of the users. Once that is done, these attributes need to be stored in a persistent storage. Portal servers usually do not provide the capability to store any arbitrary set of attributes. They only allow storing a set of predefined attributes. Hence the portal user database schema needs to be extended to add the capability to store any attributes. In this implementation, additional tables are created to keep the user attribute information. Note that the set of attributes used in the federation is not fixed. Rather it can be configured in the database.

6.5 Authentication

The portal servers always come with a built-in authentication module. In this implementation, it has been decided to use the built-in authentication module. Once a user is authenticated, the portal server provides APIs to obtain the user identity. Portlets can use this API to obtain the currently logged-in user identity. This can be done by appropriately configuring the portlet deployment descriptor file (portlet.xml). Table 6-2 shows excerpts from the portlet.xml of the GAP portlets application. As can be seen,

user-attributes like user.name, user.id etc. can be made available to a portlet application in this way. Once these attributes are made available, a portlet can obtain the values for these attributes using the *ActionRequest.getAttributes()* API.

Table 6-2. Excerpts from a portlet deployment descriptor

```
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd"
   version="1.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd">
…
<user-attribute>
      <description xml:lang="en">User Name</description>
      <name>user.name</name>
   </user-attribute>
   <user-attribute>
      <description xml:lang="en">User Id</description>
      <name>user.id</name>
   </user-attribute>
   <user-attribute>
      <description xml:lang="en">User E-Mail</description>
      <name>user.email</name>
   </user-attribute>
   <user-attribute>
      <description xml:lang="en">Last Login Time</description>
      <name>user.lastlogintime</name>
   </user-attribute>
…
</portlet-app>
```

However, the Portlet specification [11] mandates only a predefined set of user attributes made available in this way. In the case of the GAP portlets, they need to be able to obtain any arbitrary attributes of a user. Therefore the portlet.xml approach is not sufficient. In this implementation, these attributes are stored in a database table against the user ids. The GAP portlets first obtain the user id using the portlet API and then use the user id to obtain the relevant set of user attributes from the database. These attributes

are then used to construct attribute assertions. These attribute assertions are used to enable federation as described in the next section.

6.6 Federation

As described in Chapter 4, the entities of the GAP framework belong to different security domains. They are loosely-coupled by a federation. The proposed federation uses user attributes in additions to user identity. The use of attribute in the federation enables the resource owners to have flexible access-control over the resources that belong to them.

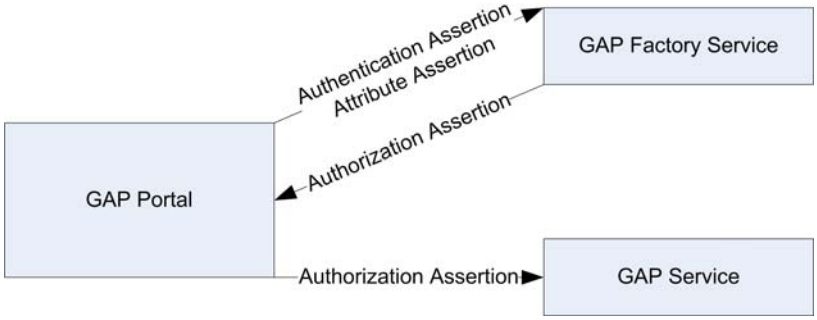

Figure 6-1. Three types of assertions used in the GAP framework. The GAP portal provides authentication and attribute assertions to the GAP factory service. The GAP factory service returns authorization assertion which is used in the subsequent communication with the GAP service.

The entities in the federation use assertions to share security information. Security Assertion Markup Language (SAML) [10] is used as the assertion language. SAML is XML based and widely accepted. It has the broad support of the industry. Hence it is well suited to the GAP framework that employs service-oriented architecture and uses Web services technology. This implementation uses SAML 1.1 [20, 21] compliant OpenSAML [22] software library. Three types of assertions are used in this implementation. They are authentication assertion, attribute assertion and authorization assertion.

As can be seen in Figure 6-1, the GAP portal, after authenticating the user, requests the GAP factory service to execute an application. Along with the request to execute an application, the GAP portal sends the authentication and attribute assertions. The authentication assertion contains the subject, the issuer, the authentication method, the time the authentication was performed and the conditions that are attached to the assertion. The subject in this case is a unique id which need not be known by the GAP factory service in advance. The attribute assertion contains the subject, the issuer, a set of attributes as attribute-value pair and the conditions attached to the assertion. Both these assertions are signed by the issuer (in this case the GAP portal). Figure 6-2 shows an attribute assertion.

```
<Assertion AssertionID="d8a7e9b43eae8b7dbb27137d91074703"
IssueInstant="2005-10-17T21:15:47.112Z"
Issuer="/O=In-VIGO/OU=Sanjee/OU=simpleCA-sanjee.acis.ufl.edu/OU=acis.ufl.edu/CN=Sanjeepan"
MajorVersion="1" MinorVersion="1" xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol">
        <Conditions NotBefore="2005-10-17T21:15:47.112Z"
        NotOnOrAfter="2005-10-17T22:15:47.112Z"/>
        <AttributeStatement>
                <Subject>
                        <NameIdentifier
                        Format="urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified"
                        NameQualifier="http://invigo.acis.ufl.edu/name">sanjee</NameIdentifier>
                        <SubjectConfirmation>
                                <ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:sender-vouches</ConfirmationMethod>
                        </SubjectConfirmation>
                </Subject>
                <Attribute AttributeName="organization" AttributeNamespace="http://invigo.acis.ufl.edu/saml/assertion">
                        <AttributeValue>ACIS</AttributeValue>
                </Attribute>
        </AttributeStatement>
        <ds:Signature>
        ...
        </ds:Signature>
</Assertion>
```

Figure 6-2. An attribute assertion

When the GAP factory service receives these assertions, it evaluates them as follows:

1. Authenticate the sender. This is performed by the secure communication layer by validating the digital signature of the message using the certificate.

2. Authorize the sender. The sender must be authorized to carry these assertions with a request.

3. Authenticate the issuer. This is performed by validating the digital signature of the assertions.

4. Authorize the issuer. The issuer must be authorized to issue these assertions.

5. Validate the assertions. Assertions need to be validated to see at the time of receiving the assertions they are not expired.

6. Extract the attributes and authorize the user. After performing the above tasks, the GAP factory service extracts the attributes. Based on the attributes, it performs the authorization.

If the user is successfully authorized, the GAP factory service initializes a GAP service to serve this request and sends the endpoint of the GAP service back to the GAP portal. Along with the endpoint address, it sends an authorization assertion to the portal. The portal, in turn, sends this authorization assertion the GAP service along with the subsequent messages. The GAP service evaluates the authorization assertion using a similar process.

6.7 Authorization

As explained in the previous section, the GAP factory service obtains the user attributes from the assertions. Using the attributes and the context information, the GAP factory service has to make an authorization decision. This part of the implementation is yet to be completed. A method for attribute-role mapping is described in [23]. Various context-sensitive role-based access control approaches are discussed in [15-17]. A detailed study of these models can be useful in designing an authorization server for the GAP framework.

CHAPTER 7
EVALUATION

This thesis proposes a framework for scalable Grid-enabling of legacy scientific applications. The key to the scalability is the common abstraction of command-line applications. The common abstraction is implemented by the GAP service so that the GAP service handles any command-line applications. The evaluation is performed to validate the scope and applicability of the common abstraction. To that end, the framework has been validated against a variety of applications, from simple UNIX commands like *ls* and *cat* to very complex applications like Dinero and Simple scalar tool set. These applications are very valuable and commonly used by many researchers. Many of these applications have very complex command line interface with large number of inputs, have dependencies among inputs, require external input files and have very complex parameter structures. Figure 7-1 shows a command to invoke Dinero application to use its full capability. As can be seen these applications are really complex to invoke using a command-line based interface.

```
dineroIV -skipcount 0 -flushcount 0 -maxcount 1000 -stat-interval 0 -informat p
-on-trigger 0 -off-trigger 0 -stat-idcombine -l1-ubsize 32 -l2-ubsize 32
-l3-ubsize 32 -l4-ubsize 32 -l5-ubsize 32 -l1-usbsize 4 -l2-usbsize 4
-l3-usbsize 4 -l4-usbsize 4 -l5-usbsize 4 -l1-usize 16k -l2-usize 16k -l3-usize
16k -l4-usize 16k -l5-usize 16k -l1-uassoc 1 -l2-uassoc 1 -l3-uassoc 1
-l5-uassoc 1 -l5-uassoc 1 -l1-urepl l -l2-urepl l -l3-urepl l -l4-urepl l
-l5-urepl l -l1-ufetch d -l2-ufetch d -l3-ufetch d -l4-ufetch d -l5-ufetch d
-l1-upfdist 1 -l2-upfdist 1 -l3-upfdist 1 -l4-upfdist 1 -l5-upfdist 1
-l1-upfabort 0 -l2-upfabort 0 -l3-upfabort 0 -l4-upfabort 0 -l5-upfabort 0
-l1-uwalloc a -l2-uwalloc a -l3-uwalloc a -l4-uwalloc a -l5-uwalloc a -l1-uccc
-l2-uccc -l3-uccc -l4-uccc -l5-uccc
```

Figure 7-1. A command that utilizes the full capability of the Dinero application

The corresponding user interface created by the GAP framework is shown in Figure 7-2. As can be seen, this user interface makes it very easy to use Dinero application. This illustrates the GAP framework's ability to support very complex applications. Further it shows the usability of the GAP framework.

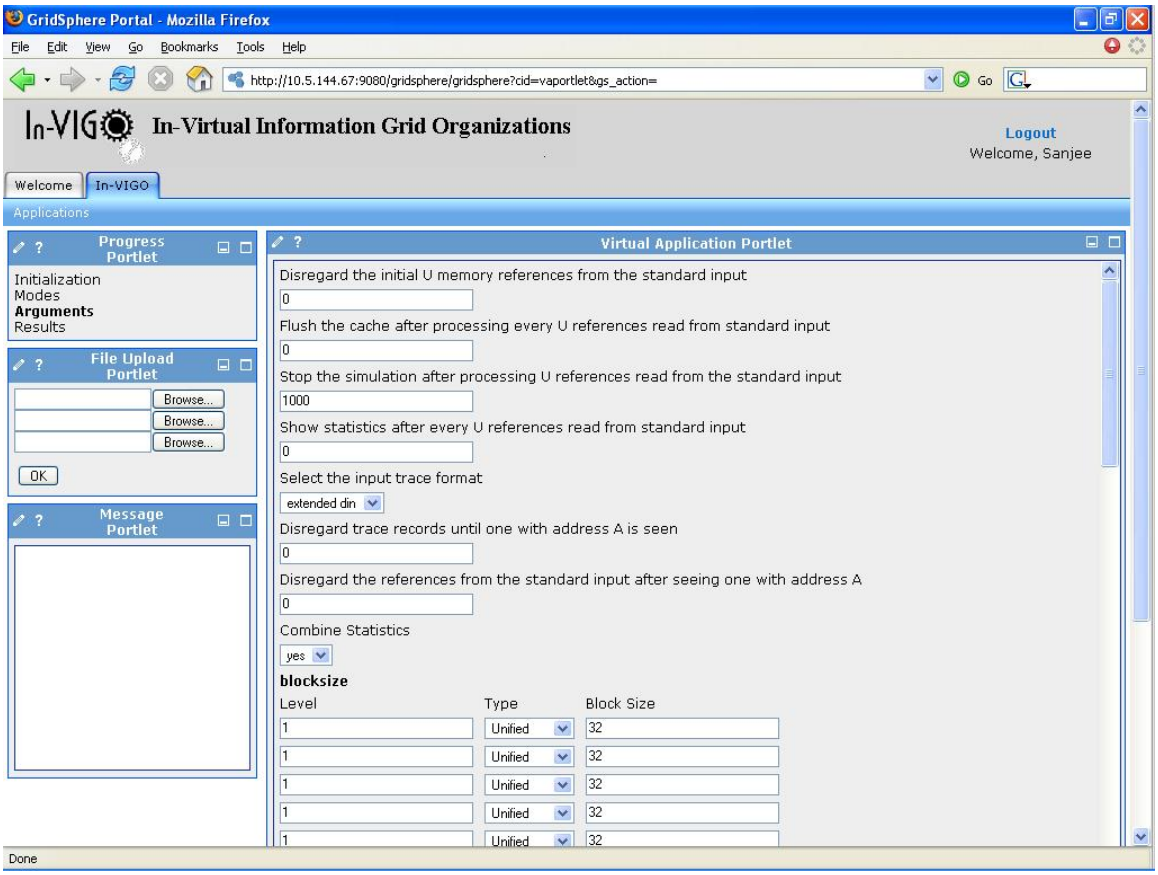

Figure 7-2. The user interface generated by the GAP framework to use the full capability of Dinero application

The applications, used in the evaluation, have been selected to validate various features commonly found in command-line applications. Table 7-1 describes those features that are currently supported by the GAP framework. The set of features are categorized into three groups. Input type considers the variations at the single user input level. Argument complexity looks at the inter-relationships between user inputs. Finally the application type looks at complexities involved in organizing and executing

applications. As can be seen the GAP approach accommodates a wide variety features that are found in command-line applications.

Table 7-1. Various command-line application features that are handled by the GAP framework

Input Type	**Description**
Text	User types in some values. This is the most common input method.
Selection	User selects one of the predefined values. These values can have a description so that user selects the value based on its description.
Boolean	User selects to use or not use a feature
With Flag	A flag is associated with the input. These flags are not shown to the users. Rather, the GAP service adds the flag to the command when the command is constructed
Multi Value	An input may consists of a flag and more than one value can be associated with that flag.
File	User provides an external file as an input to the application. The user may upload this file and then select it or he/she may select an existing file.
Argument Complexity	**Description**
Simple	An argument consists of single user input. This is the most common form of an argument.
Multipart	Certain inputs have to occur together (perhaps in a specific order) in order to construct a complete argument.
Repetitive	In certain applications, some arguments can repeat several times. For example in Dinero the cache specification can be repeated up to five levels.
Application Type	**Description**
Single command	In order to execute the application, only one command is required. This is the most common case.
Multi command	Some applications require more than one command to execute. These commands can be encapsulated in a script and that script can be invoked by the GAP service.
Interactive	Some command-line applications have an interactive interface. These applications can be invoked by a non interactive script. Then the script can be invoked by the GAP service.
Toolset	Some applications are collected as toolset. The GAP framework supports toolset by providing the capability to define the toolset in the application registry. In this way, the GAP framework supports toolsets.

Table 7-2 shows the various applications that have been Grid-enabled using the GAP framework.

Table 7-2. Various applications that have been Grid-enabled using the GAP framework.

Application	**Input Types**	**Argument Complexities**	**Application Type**
Simple scalar toolset. It consists of five complex applications	Text, Selection, Boolean, With flag, File	Simple, Multipart	Single command, Toolset
Dinero	Text, Selection, Boolean, With flag, File	Simple, Multipart, Repetitive	Single command
Cacti	Text	Simple	Single command
Abinit	Text, File	Simple	Interactive
Octave	Text, File	Simple	Single command
Gamess	Text, File	Simple	Single command
Ls	Text, Multi-value	Simple	Single command
Cat	File	Simple	Single command

CHAPTER 8
RELATED WORK

8.1 Grid-Enabling

An overview of research efforts to build interoperable portal services around a Web services model is presented in [24]. It describes a portal architecture, beginning with core portal services that can be used to build Application Web Services, which in turn may be aggregated and managed through Portlet containers.

The paper identifies Job Submission, Data Management, Context Management and Batch Script Generation as the core services. It also attempts to define a general purpose set of schemas that describe how to use a particular application and bind it to the services it needs. These schemas are the foundation for what is called Application Web Services.

An Application Web Service has some similarities with the GAP Service. Both approaches attempts to wrap legacy applications as services. However the difference lies in the capabilities of the service. The GAP Service is capable of supporting applications with very complex command-line syntax (e.g. DineroIV with multi-part arguments and with dependencies). Also the GAP Service provides the capability to customize applications for different user groups. Further the GAP Service is capable of validating user inputs before the job is submitted to a compute resource using regular expressions. Portal interfaces generated with the help of the GAP Service are more detailed and application specific. The description of each input is application specific.

An implementation of a Grid Application Factory Service that is based on a component architecture that utilizes Web services is described in [25]. The factory

service is utilized by Grid clients to authenticate and authorize users to configure and launch instances of distributed applications. This helps solve the problem of building reliable and scalable Grid applications by separating the process of installation and hosting from application execution.

The Application Factory Service creates instances of distributed applications that are composed of well-tested and deployed components, each executing in configured hosting environments. In this model each of the components is assumed to be a Web service. The work reported in this thesis mainly focuses on how to represent legacy applications as Web services. In a sense the work reported in this thesis is complementary to that reported in [25] because application Web services can then be composed using the Application Factory Service.

GridSpeed [26] is a Grid portal-hosting server that automatically generates and publishes customized web interfaces of applications, with minimal effort required from the users. Users need not modify their applications nor write any glue code to publish their application on the web. GridSpeed also attempts to find a solution to web-enabling of applications. However its approach is different from the GAP approach described in this paper. GridSpeed provides a graphical mechanism to develop an application portal so that application specialists do not need to know coding. Each application has its own JSP pages generated by GridSpeed. The approach used in the GAP Service does not create different JSP pages for each application but generates the application interface automatically and dynamically. It eliminates the need to manage many different JSP files, and hence is more scalable and manageable. The other significant difference is that, in GridSpeed's approach, the application specialist is expected to know the Grid computing

environment to bind applications to Grid resources. In the GAP Service approach, the application specialist is only expected to provide the requirements and binding takes place transparently.

Soaplab [27] is a set of Web services providing programmatic access to some applications on remote computers. Applab provides the same functionality using CORBA. Soaplab also wraps command line applications (mainly data analysis applications) as Web services. Soaplab defines an interface for this class of applications and has different implementations for each of the applications. Hence, every time a new application is deployed, a new service implementation is automatically generated and deployed in a Web service container. The GAP Service has a single service implementation that is dynamically reconfigured at runtime. Further the GAP Service approach provides the capability to customize applications and to validate user inputs.

8.2 Federation and Access Control

An approach to federation and access control in the context of federated databases is described in [28]. The objectives (a single-sign-on capability for users, high level of autonomy for database custodians and low maintenance overhead) are similar to that of the GAP framework. The proposed federation in [28] is enabled by sharing user credentials. This effectively means that the custodians should know the identity of the users and should have the capability to evaluate various different forms of credentials. In addition the access control decisions are based on the user's identity and a flexible attribute-based access control is not supported. In contrast, the GAP framework uses SAML assertions, an approach that does not require the owners of the resources to know the exact user identity in advance and does not need to have the capability to evaluate

various different types of credentials. Furthe,r the augmentation of the federation with attributes enables the GAP framework to support flexible multi-class access control.

An alternative approach to federation is delegation. A X.509 proxy certificate based delegation approach is described in [29]. An approach to delegation using SAML is described in [30]. Although delegation based solutions are commonly employed in current Grid computing systems, it does not scale well when there are large number of users. The reason is that the resource providers are required to be identity providers and the portals that act as a front-end to access the resources need to support the credentials provided by the resource provider. These limitations are avoided in the federation-based GAP framework.

Extensive research has been conducted over the last several years in the area of context-aware role-based access control. Some of these works have been reported in [31, 15-17]. A study of the various aspects of context-sensitivity in access control will help devise access control function for the GAP framework. An approach to map attributes to roles is reported in [23]. Since the same problem needs to be addressed in the GAP framework, a detailed study of [23] is very useful.

CHAPTER 9
CONCLUSIONS AND FUTURE DIRECTIONS

This thesis identified the need to have a framework to Grid-enable legacy scientific applications. The proposed GAP framework provides a scalable and secure solution to the Grid-enabling problem. Technical contributions of this thesis are (1) an abstraction that is generic enough to capture the descriptions of numerous types of applications, (2) an XML schema that captures the abstraction, (3) a generic application service to interpret the abstraction and (4) a user-interface generation technique that dynamically generates application specific user interfaces. The security architecture of the framework supports multiple, loosely-coupled security domains using a federation. This loosely-coupled security federation makes it possible for the resource owners, who belong to a security domain, to share them with Grid users, who belong to another security domain, while providing a single-sign-on capability and maintaining access control over their resources without changing the existing security mechanisms. A prototype system has been implemented. The prototype has been evaluated against a variety of applications to evaluate its scalability with respect to Grid-enabling a large number of different types of applications. Being able to handle very complex applications like dinero and simple scalar toolkit gives creditability to the proposed approach.

The choice of service-oriented architecture and Web services provides a unique opportunity to handle workflows by using existing and emerging technologies such as Business Process Execution Language for Web services (BPEL4WS) [32]. A complex simulation process involving multiple applications can be managed and controlled

without frequent user interaction by providing a proper workflow specification and enacting it in a workflow engine. The GAP framework exposes the applications as services. In the future GAP framework can be extended to support workflows that involve a number of GAP services.

The ability to create virtual machines on the fly using VMPlants [33] and the support for Grid file system sessions with WSRF based services [34] provides a possibility for the GAP service to encapsulate not just the application, but the application, its data and the execution environment. Provided a virtual machine for a given specification can be instantiated within a reasonable amount of time, each application session can be isolated by making it run a virtual machine.

Another interesting problem that may be addressed in the future is the authority recognition problem. In the GAP framework, the GAP framework-services recognize the GAP portal as the attribute authority. The GAP framework-services use the services provided by In-VIGO, based on the assertions of the GAP portal. One can notice that the In-VIGO does not directly recognize the GAP portal. In such situation, questions arise as to whether the GAP framework-services must let the In-VIGO know that it is using In-VIGO on behalf of users not directly recognized by it. If so, what information should the GAP framework-services provide to In-VIGO and how should this information be provided so that it can be evaluated without human intervention? Although these questions are raised in the context of the GAP framework, they are very relevant when trust is brokered.

APPENDIX A
FUNCTIONS OF SECURITY ARCHITECTURE

A.1 Identity Provider

Every actor that participates in the process of executing a task will have to have a digital identity. An actor in this context may be an end user, administrator, application specialists or even a system entity (e.g. GAP portal, GAP Framework-services and In-VIGO). The identity provider is responsible for managing digital identities. Identity management encompasses the following tasks.

*Registration***:** Before an identity can be issued to an actor it is necessary to ascertain the credentials. Face-to-face meeting and the use of already existing trustworthy identities (e.g. Gator1 ID, SSN etc.) are some of the common methods to ascertain the credentials of an actor. The process of identity registration may also involve ascertaining the attributes. Usually identities are valid for a limited lifetime. In order to extend its lifetime an identity has to be renewed.

Renewal: If an actor wants to keep a digital identity beyond its lifetime, he/she has to renew it. The renewal process usually is simpler than the registration process.

Revocation: Sometimes it is necessary to revoke an identity for some reasons (fraud, compromised identity, status change, etc.). Identity providers must have a mechanism to inform all relying parties of this revocation within a reasonable amount of time.

A.2 Authentication

Authentication is the process of reliably verifying an identity of someone or something. The username/password-based authentication is the most common method employed to authenticate persons. The address-based authentication is method used to authenticate computer systems. More sophisticated cryptographic authentication methods using challenge-response protocols are also commonly used in critical systems. Recently biometric authentication methods to authenticate persons are also used. The verification method depends on the type of identity. This function may well be co-located with the identity provider or it may be separate entity. In password-based systems, identity provider and the authentication functions are commonly co-located. However, in public key infrastructure (PKI) based systems, these two functions are generally separate. An authentication decision may be expressed as an authentication assertion. These assertions are useful when the relying party and the authenticator are separate entities.

A.3 Attribute Authority

In some cases, a relying party needs the attributes of the actors in addition to the identities for making authorization decisions. The attribute authority is responsible for issuing attribute assertions that vouch for the attributes of actors. This again may be co-located with the identity provider or may be separate. There are several ways an attribute authority may assert the attributes of a subject. X.509 attribute certificates [35] and SAML [10] attribute assertions are two common forms to express attribute assertions.

A.4 Authorization

Authorization is the process of allowing or denying an actor (subject) access to a resource. Figure A-1 shows various sources of inputs that are used by an authorization server to make an authorization decision. The four inputs to the authorization server are

the subject, the resource to which the subject requests access, the policies that define how access control decisions are made and the environmental conditions (like time, resource overloading etc.) that have a bearing on the decision. There are several techniques used to design an authorization server. The role-based access control (RBAC) [14] is the most commonly used technique today. If the decision point and the enforcement point are different, then the authorization decision has to be expressed as an assertion by the decision point. The enforcement point can then use the assertion to enforce the decisions. SAML authorization decision assertion is one way to represent an authorization decision.

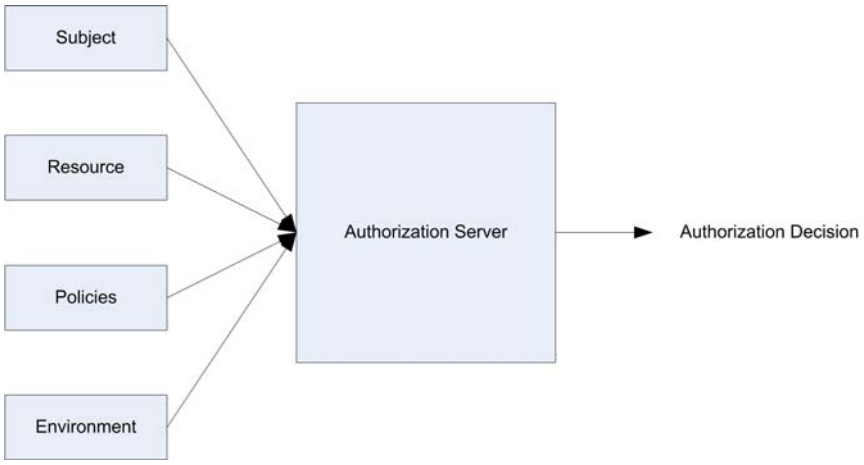

Figure A-1. Schematic representation of authorization decision process. Various sources of information are used by the Authorization Server to make authorization decisions. If the enforcement point is different from the decision point then the authorization decision is expressed as an authorization assertion.

A.5 Secure Communication

The functions that have been described so far are found in almost all security systems, including non-distributed systems. Distributed systems require secure communication. When the entities communicate over unsecured network, there is a danger of messages being intercepted, eavesdropped, modified or replayed. Sufficient safeguards should be in place to maintain the integrity and privacy of messages and to

prevent such malicious acts. Encryption and digital signatures are used in various ways to maintain the integrity and privacy of messages. WS-Security [386], SSL/TLS [37] and IP Security (IPSec) [38] are some of the systems and technologies that enable secure communication in a system based on Web services. WS-Security provides integrity and privacy safeguards at the SOAP message level. This will provide end-to-end security even when there are message layer intermediaries along the communication path. SSL/TLS provides secure communication at the transport layer. IPSec provides secure communication at the IP layer. Figure A-2 shows the different technologies employed at different layers to achieve secure communication.

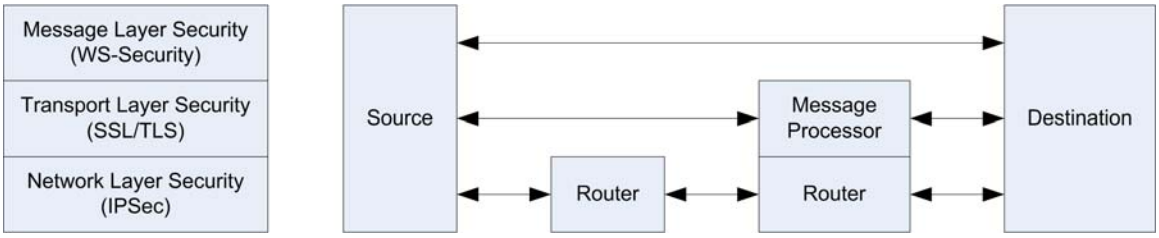

Figure A-2. Various secure communication technologies. As shown in the figure secure communication technologies may be employed at network, transport or message layer.

A.6 Trust

The security architecture for a multi-domain, distributed system must address issues related to trust and federation. This section discusses issues related to trust. Section A.7 discusses issues related to federation. In order for the different entities belonging to different security domains to work together, a trust relationship must exist between the security domains. A trust relationship defines what each party is authorized to do and what its responsibilities are. Trust relationships can be agreed upon by either expressing the security process (e.g. identities of the users are provided following certain procedure) or by expressing commitments (e.g. the entity that provides the identities and/or attributes

undertakes commitments relating to agreed upon set of user actions) or by expressing both the security process and commitments. The trust relationships can be direct or brokered. In direct trust relationships such agreements can be reached at the configuration time using written contractual agreements. In brokered trust relationships, it may be necessary to reach such agreements dynamically. Some of the trust relationships have to be preconfigured at the configuration time in order to be able to construct trust networks. At minimum, the graph formed by the preconfigured trust network must be connected.

A.7 Federation

Identity federation provides a simple, loosely-coupled security model for managing identities. The foundations of identity federation are the trust and secure communication between the security domains. The existence of trust relationships between the participating domains in the federation enables a participant to rely on an authentication decision made by another participant. The authentication decisions are conveyed using authentication assertions. Federation helps to loosely-couple disparate security systems together without loosing administrative autonomy. This loose coupling enables security domains to work together without replicating identities or access control policies. Hence it simplifies security administration.

Identity federation is used to federate identities between the participating domains. This is sufficient when all the users participating in the federation belong to the same user class requiring the same access to the resources. However, when users belong to many different classes, simple identity federation is not sufficient. The GAP security architecture addresses this issue by extending identity federation to include user attributes in addition to user identity.

APPENDIX B
APPLICATION DESCRIPTION SCHEMA

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

<!--
    Document   : vatypes.xsd
    Created on : June 24, 2004
    Author     : sanjee
    Description:
        This schema defines various types used in the schema of an
application
        description.
-->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:va="http://va.com"
        targetNamespace="http://va.com"
        elementFormDefault="qualified">

  <!--
  The complex type tool defines an application. This is the root
element
  of all the application description document. The tool element
consists of
  name (application name), binary (application binary name), path
(application path),
  a set of arguments (defined later), qos (execution environment) and
mode
  (customized applications).
  -->
  <complexType name="tool">
        <sequence>
            <element name="name" type="string" minOccurs="1"
maxOccurs="1"/>
            <element name="binary" type="string" minOccurs="1"
maxOccurs="1"/>
            <element name="path" type="string" minOccurs="1"
maxOccurs="1"/>
            <element name="argument" type="va:argument" minOccurs="1"
maxOccurs="unbounded"/>
            <element name="qos" type="va:qos"/>
            <element name="mode" type="va:mode" maxOccurs="unbounded"/>
        </sequence>
  </complexType>

  <!--
  A part is an indivisible unit of input. A collection of parts is an
argument (defined later).
```

```
  This element consists of description, default value, value (user
provided), valid (is the user
  provided value valid), unique id, type (flagonly, filelist, random
and sequence), flag, minOccurs
  (number of times this part is required), maxOccurs (number to times
this part can occur), pattern
  (a regular expression to validate the value), choice (if the value is
restricted to a few), start
  (start of a sequence if the type is sequence), end (end value if the
type is a sequence) and delimiter
  (the character(s) that seperate flag and value). The elements value
and valid are not used for describing
  an application. They are used when an application is used.
  -->
  <complexType name="part">
    <sequence>
        <element name="description" type="string" minOccurs="1"/>
        <element name="defaultValue" type="string" minOccurs="1"/>
        <element name="value" type="string" minOccurs="0"
maxOccurs="unbounded"/>
        <element name="valid" type="boolean" minOccurs="0"
maxOccurs="1" default="false"/>
    </sequence>
    <attribute name="id" type="string"/>
    <attribute name="type" type="string"/>
    <attribute name="flag" type="string"/>
    <attribute name="minOccurs" type="positiveInteger"/>
    <attribute name="maxOccurs" type="positiveInteger"/>
    <attribute name="pattern" type="string"/>
    <attribute name="choice" type="string"/>
    <attribute name="start" type="integer"/>
    <attribute name="end" type="integer"/>
    <attribute name="delimiter" type="string"/>
  </complexType>

  <!--
  Basic argument consists of name, minOccurs, maxOccurs and
useDefalutValue.
  The elements minOccurs and maxOccurs have the same meaning as in part
but applied
  to an argument. If useDefaultValue is true, the argument is not
visible to user.
  When executing the application, the argument takes the defaultValue.
Basic argument
  element is used to construct modes. Further, it is extended to
defined complete arguments
  as shown later.
  -->
  <complexType name="basicArgument">
    <attribute name="name" type="string" use="required"/>
    <attribute name="minOccurs" type="nonNegativeInteger"/>
    <attribute name="maxOccurs" type="nonNegativeInteger"/>
    <attribute name="useDefaultValue" type="boolean"/>
  </complexType>

  <!--
```

```
  Argument is extended from basicArgument. In addition to the
attributes of
  basicArgument, it contains a set of parts. Each argument is expected
to have at
  least one part. But there is no maximum limit. Arguments are used to
group parts
  that are logically related and hence have to appear together.
  -->
  <complexType name="argument">
    <complexContent>
        <extension base="va:basicArgument">
            <sequence>
                <element name="part" type="va:part" minOccurs="1"
maxOccurs="unbounded"/>
            </sequence>
        </extension>
    </complexContent>
  </complexType>

  <!--
  A mode defines a custom application. It is a collection of
basicArguments.
  -->
  <complexType name="mode">
      <sequence>
        <element minOccurs="1" maxOccurs="unbounded" name="argument"
type="va:basicArgument"/>
      </sequence>
      <attribute name="name" use="required" type="string"/>
   </complexType>

   <!--
   Qos defines the execution environment requirement. Currently some
example parameters
   are defined.
   -->
  <complexType name="qos">
      <sequence>
        <element name="os" type="string"/>
        <element name="version" type="string"/>
        <element name="memory" type="string"/>
        <element name="bandwidth" type="string"/>
      </sequence>
  </complexType>
</schema>
```

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

<!--
    Document   : va.xsd
    Created on : June 24, 2004
    Author     : sanjee
    Description:
        This is the schema of an application description. This schema
        uses the types defined in vatypes.xsd document.
-->

<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:va="http://va.com"
        targetNamespace="http://va.com"
        elementFormDefault="qualified">
    <!--
    vatypes.xsd defines various types used in this document.
    -->
    <include schemaLocation="vatypes.xsd"/>

    <!--
    This is the root element of any application description document.
    -->
    <element name="tool" type="va:tool">

        <!--
        This constraint ensures that the mode names are unique.
        -->
        <unique name="modes">
            <selector xpath="va:mode"/>
            <field xpath="@name"/>
        </unique>

        <!--
        This key refers to argument names.
        -->
        <key name="argumentKey">
            <selector xpath="va:argument"/>
            <field xpath="@name"/>
        </key>

        <!--
        This constraint ensures the argument names used in mode
definition exist.
        -->
        <keyref name="argumentName" refer="va:argumentKey">
            <selector xpath="va:mode/va:argument"/>
            <field xpath="@name"/>
        </keyref>
  </element>
</schema>
```

APPENDIX C
DINERO APPLICATION DESCRIPTION

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

<!--
    Document   : dinero.xml
    Created on : June 24, 2004
    Author     : sanjee
    Description:
        This document defines the application description of Dinero
application.
-->

<tool   xmlns="http://va.com"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="../va.xsd">

    <name>Dinero IV</name>
    <binary>dineroIV</binary>
    <path>#APP_HOME#/d4-7/</path>

    <argument name="skipcount" minOccurs="1" maxOccurs="1">
        <part flag ="-skipcount" type="random" minOccurs="1" maxOccurs
="1" pattern="[0-9]*">
            <description>Disregard the initial U memory references from
the standard input</description>
            <defaultValue>0</defaultValue>
        </part>
    </argument>

    <argument name="flushcount" minOccurs="1" maxOccurs="1">
        <part flag ="-flushcount" type="random" minOccurs="1" maxOccurs
="1" pattern="[0-9]*">
            <description>Flush the cache after processing every U
references read from standard input</description>
            <defaultValue>0</defaultValue>
        </part>
    </argument>

    <argument name="maxcount" minOccurs="1" maxOccurs="1">
        <part flag="-maxcount" type="random" minOccurs="1" maxOccurs
="1" pattern="[0-9]*">
            <description>Stop the simulation after processing U
references read from the standard input</description>
            <defaultValue>1000</defaultValue>
        </part>
    </argument>

    <argument name="statinterval" minOccurs="1" maxOccurs="1">
```

```
        <part flag="-stat-interval" type="random" minOccurs="1"
maxOccurs ="1" pattern="[0-9]*">
            <description>Show statistics after every U references read
from standard input</description>
            <defaultValue>0</defaultValue>
        </part>
    </argument>

    <argument name="informat" minOccurs="1" maxOccurs="1">
        <part flag="-informat" type="random" minOccurs="1" maxOccurs
="1" choice="D=extended din:d=traditional
din:p=pixie32:P=pixie64:b=binary">
            <description>Select the input trace format</description>
            <defaultValue>d</defaultValue>
        </part>
    </argument>

    <argument name="ontrigger" minOccurs="1" maxOccurs="1">
        <part flag="-on-trigger" type="random" minOccurs="1" maxOccurs
="1" pattern="[0-9]*">
            <description>Disregard trace records until one with address
A is seen</description>
            <defaultValue>0</defaultValue>
        </part>
    </argument>

    <argument name="offtrigger" minOccurs="1" maxOccurs="1">
        <part flag="-off-trigger" type="random" minOccurs="1" maxOccurs
="1" pattern="[0-9]*">
            <description>Disregard the references from the standard
input after seeing one with address A</description>
            <defaultValue>0</defaultValue>
        </part>
    </argument>

    <argument name="statcombine" minOccurs="1" maxOccurs="1">
        <part flag="-stat-idcombine" type="flagonly" minOccurs="1"
maxOccurs="1" choice="yes:no">
            <description>Combine Statistics</description>
            <defaultValue>no</defaultValue>
        </part>
    </argument>

    <argument name="blocksize" minOccurs="1" maxOccurs="5">
        <part flag="-l" type="sequence" start="1" end="5" minOccurs="1"
maxOccurs ="1" pattern="[1-5]" delimiter="">
            <description>Level</description>
            <defaultValue>1</defaultValue>
        </part>
        <part flag="-" type="random" minOccurs="1" maxOccurs="1"
choice="u=Unified:i=Instruction:d=Data" delimiter="">
            <description>Type</description>
            <defaultValue>u</defaultValue>
        </part>
        <part flag ="bsize" type="random" minOccurs="1" maxOccurs="1"
pattern="([0-9]+)([kKmMgG]*)">
            <description>Block Size</description>
```

```
            <defaultValue>32</defaultValue>
        </part>
    </argument>

    <argument name="subblocksize" minOccurs="1" maxOccurs="5">
        <part flag="-l" type="sequence" start="1" end="5" minOccurs="1"
maxOccurs ="1" pattern="[1-5]" delimiter="">
            <description>Level</description>
            <defaultValue>1</defaultValue>
        </part>
        <part flag="-" type="random" minOccurs="1" maxOccurs="1"
choice="u=Unified:i=Instruction:d=Data" delimiter="">
            <description>Type</description>
            <defaultValue>u</defaultValue>
        </part>
        <part flag="sbsize" type="random" minOccurs="1" maxOccurs="1"
pattern="([0-9]+)([kKmMgG]*)">
            <description>Sub Block Size</description>
            <defaultValue>4</defaultValue>
        </part>
    </argument>

    <argument name="size" minOccurs="1" maxOccurs="5">
        <part flag="-l" type="sequence" start="1" end="5" minOccurs="1"
maxOccurs ="1" pattern="[1-5]" delimiter="">
            <description>Level</description>
            <defaultValue>1</defaultValue>
        </part>
        <part flag="-" type="random" minOccurs="1" maxOccurs="1"
choice="u=Unified:i=Instruction:d=Data" delimiter="">
            <description>Type</description>
            <defaultValue>u</defaultValue>
        </part>
        <part flag="size" type="random" minOccurs="1" maxOccurs="1"
pattern="([0-9]+)([kKmMgG]*)">
            <description>Size</description>
            <defaultValue>16k</defaultValue>
        </part>
    </argument>

    <argument name="associativity" minOccurs="1" maxOccurs="5">
        <part flag="-l" type="sequence" start="1" end="5" minOccurs="1"
maxOccurs ="1" pattern="[1-5]" delimiter="">
            <description>Level</description>
            <defaultValue>1</defaultValue>
        </part>
        <part flag="-" type="random" minOccurs="1" maxOccurs="1"
choice="u=Unified:i=Instruction:d=Data" delimiter="">
            <description>Type</description>
            <defaultValue>u</defaultValue>
        </part>
        <part flag="assoc" type="random" minOccurs="1" maxOccurs="1"
pattern="[0-9]*">
            <description>Associativity</description>
            <defaultValue>1</defaultValue>
        </part>
    </argument>
```

```
    <argument name="replacement" minOccurs="1" maxOccurs="5">
        <part flag="-l" type="sequence" start="1" end="5" minOccurs="1"
maxOccurs ="1" pattern="[1-5]" delimiter="">
            <description>Level</description>
            <defaultValue>1</defaultValue>
        </part>
        <part flag="-" type="random" minOccurs="1" maxOccurs="1"
choice="u=Unified:i=Instruction:d=Data" delimiter="">
            <description>Type</description>
            <defaultValue>u</defaultValue>
        </part>
        <part flag="repl" type="random" minOccurs="1" maxOccurs="1"
choice="l=LRU:f=FIFO:r=Random">
            <description>Replacement Policy</description>
            <defaultValue>l</defaultValue>
        </part>
    </argument>

    <argument name="fetch" minOccurs="1" maxOccurs="5">
        <part flag="-l" type="sequence" start="1" end="5" minOccurs="1"
maxOccurs ="1" pattern="[1-5]"  delimiter="">
            <description>Level</description>
            <defaultValue>1</defaultValue>
        </part>
        <part flag="-" type="random" minOccurs="1" maxOccurs="1"
choice="u=Unified:i=Instruction:d=Data" delimiter="">
            <description>Type</description>
            <defaultValue>u</defaultValue>
        </part>
        <part flag="fetch" type="random" minOccurs="1" maxOccurs="1"
choice="d=Demand:a=Always:m=Miss:t=Tagged:l=Load Forward:s=Sub Block">
            <description>Fetch Policy</description>
            <defaultValue>d</defaultValue>
        </part>
    </argument>

    <argument name="prefetchdistance" minOccurs="1" maxOccurs="5">
        <part flag="-l" type="sequence" start="1" end="5" minOccurs="1"
maxOccurs ="1" pattern="[1-5]" delimiter="">
            <description>Level</description>
            <defaultValue>1</defaultValue>
        </part>
        <part flag="-" type="random" minOccurs="1" maxOccurs="1"
choice="u=Unified:i=Instruction:d=Data" delimiter="">
            <description>Type</description>
            <defaultValue>u</defaultValue>
        </part>
        <part flag="pfdist" type="random" minOccurs="1" maxOccurs="1"
pattern="[0-9]*">
            <description>Prefetch Distance</description>
            <defaultValue>1</defaultValue>
        </part>
    </argument>

    <argument name="prefetchabort" minOccurs="1" maxOccurs="5">
```

```
        <part flag="-l" type="sequence" start="1" end="5" minOccurs="1"
maxOccurs ="1" pattern="[1-5]" delimiter="">
            <description>Level</description>
            <defaultValue>1</defaultValue>
        </part>
        <part flag="-" type="random" minOccurs="1" maxOccurs="1"
choice="u=Unified:i=Instruction:d=Data" delimiter="">
            <description>Type</description>
            <defaultValue>u</defaultValue>
        </part>
        <part flag="pfabort" type="random" minOccurs="1" maxOccurs="1"
pattern="[0-9]*">
            <description>Prefetch Abort</description>
            <defaultValue>0</defaultValue>
        </part>
    </argument>

    <argument name="writeallocation" minOccurs="1" maxOccurs="5">
        <part flag="-l" type="sequence" start="1" end="5" minOccurs="1"
maxOccurs ="1" pattern="[1-5]" delimiter="">
            <description>Level</description>
            <defaultValue>1</defaultValue>
        </part>
        <part flag="-" type="random" minOccurs="1" maxOccurs="1"
choice="u=Unified:i=Instruction:d=Data" delimiter="">
            <description>Type</description>
            <defaultValue>u</defaultValue>
        </part>
        <part flag="walloc" type="random" minOccurs="1" maxOccurs="1"
choice="a=Always:n=Never:f=Nofetch">
            <description>Write Allocation Policy</description>
            <defaultValue>a</defaultValue>
        </part>
    </argument>

    <argument name="computecapacity" minOccurs="1" maxOccurs="5">
        <part flag="-l" type="sequence" start="1" end="5" minOccurs="1"
maxOccurs ="1" pattern="[1-5]"  delimiter="">
            <description>Level</description>
            <defaultValue>1</defaultValue>
        </part>
        <part flag="-" type="random" minOccurs="1" maxOccurs="1"
choice="u=Unified:i=Instruction:d=Data" delimiter="">
            <description>Type</description>
            <defaultValue>u</defaultValue>
        </part>
        <part flag="ccc" type="flagonly" minOccurs="1" maxOccurs="1"
choice="yes:no">
            <description>Compute Compulsory/Capacity/Conflict miss
rates</description>
            <defaultValue>no</defaultValue>
        </part>
    </argument>

    <argument name="stdin" minOccurs="1" maxOccurs="1">
        <part type="random" minOccurs="1" maxOccurs ="1">
            <description>Std Input File Name</description>
```

```
            <defaultValue>#APP_HOME#/d4-
7/benchmarks/cc1.din</defaultValue>
        </part>
    </argument>

    <argument name="stdout" minOccurs="1" maxOccurs="1">
        <part type="random" minOccurs="1" maxOccurs ="1">
            <description>Std Output File Name</description>
            <defaultValue>dinero_out</defaultValue>
        </part>
    </argument>

    <argument name="stderr" minOccurs="1" maxOccurs="1">
        <part type="random" minOccurs="1" maxOccurs ="1">
            <description>Std Error File Name</description>
            <defaultValue>dinero_err</defaultValue>
        </part>
    </argument>

    <qos>
        <os>Linux</os>
        <version>2.4.*</version>
        <memory/>
        <bandwidth/>
    </qos>

    <!--mode name="default">
        <argument name="informat" useDefaultValue="true"/>
        <argument name="blocksize" useDefaultValue="true"/>
        <argument name="size" useDefaultValue="true"/>
        <argument name="stdin" useDefaultValue="true"/>
        <argument name="stdout" useDefaultValue="true"/>
    </mode-->
    <mode name="basic">
        <argument name="informat" useDefaultValue="true"/>
        <argument name="blocksize"/>
        <argument name="size"/>
        <argument name="stdin" useDefaultValue="true"/>
        <argument name="stdout" useDefaultValue="true"/>
    </mode>
    <mode name="basic2">
        <argument name="informat" useDefaultValue="true"/>
        <argument name="blocksize" maxOccurs="1"/>
        <argument name="size" maxOccurs="1"/>
        <argument name="statcombine"/>
        <argument name="computecapacity" maxOccurs="1"/>
        <argument name="stdin" useDefaultValue="true"/>
        <argument name="stdout" useDefaultValue="true"/>
    </mode>
    <mode name="advanced">
        <argument name="informat"/>
        <argument name="statcombine"/>
        <argument name="blocksize"/>
        <argument name="size"/>
        <argument name="associativity"/>
        <argument name="replacement"/>
        <argument name="fetch"/>
```

```
        <argument name="prefetchdistance"/>
        <argument name="prefetchabort"/>
        <argument name="writeallocation"/>
        <argument name="computecapacity"/>
        <argument name="stdin" useDefaultValue="true"/>
        <argument name="stdout" useDefaultValue="true"/>
    </mode>
    <mode name="fullblown">
        <argument name="skipcount"/>
        <argument name="flushcount"/>
        <argument name="maxcount"/>
        <argument name="statinterval"/>
        <argument name="informat"/>
        <argument name="ontrigger"/>
        <argument name="offtrigger"/>
        <argument name="statcombine"/>
        <argument name="blocksize"/>
        <argument name="subblocksize"/>
        <argument name="size"/>
        <argument name="associativity"/>
        <argument name="replacement"/>
        <argument name="fetch"/>
        <argument name="prefetchdistance"/>
        <argument name="prefetchabort"/>
        <argument name="writeallocation"/>
        <argument name="computecapacity"/>
        <argument name="stdin" useDefaultValue="true"/>
        <argument name="stdout" useDefaultValue="true"/>
    </mode>
</tool>
```

LIST OF REFERENCES

1. Ian Foster, Carl Kesselman, Steven Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal on High Performance Computing Applications*, vol. 15, no. 3, pp. 200-222.

2. G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. van Nieuwpoort, A. Reinefeld, F. Schintke, T. Schott, E. Seidel, B. Ullmer, "The Grid Application Toolkit: Toward Generic and Easy Application Programming Interfaces for the Grid," *Proceedings of IEEE*, vol. 93, no. 3, pp. 534-550.

3. G. von Laszewski, I. Foster, J. Gawor, P. Lane, "A Java Commodity Grid Kit," *Concurrency and Computation: Practice and Experience*, vol. 13, no. 8–9, pp. 643-662.

4. Grid Application Framework for Java, http://www.alphaworks.ibm.com/tech/GAF4J, Accessed on 08/12/05.

5. Web Services Resource Framework, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf, Accessed on 08/12/05.

6. OGCE Portal, http://www.ogce.org, Accessed on 12/22/04.

7. HotPage Portal, https://hotpage.paci.org/, Accessed on 12/22/04.

8. In-VIGO Portal, http://invigo.acis.ufl.edu/, Accessed on 01/19/05.

9. PUNCH Portal, http://punch.purdue.edu/, Accessed on 01/19/05.

10. OASIS Security Services (SAML) TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security, Accessed on 09/26/05.

11. *Java Portlet Specification Version 1*, JSR 168, October 2003.

12. Globus Project, http://www.globus.org, Accessed on 01/19/05.

13. Liberty Alliance Project, https://www.projectliberty.org/, Accessed on 10/19/05.

14. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, Charles E. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38-47.

15. Arun Kumar, Neeran Karnik, Girish Chafle, "Context Sensitivity in Role-based Access Control," *ACM SIGOPS Operating Systems Review*, vol. 36, no. 7, pp. 53-66.

16. Rafae Bhatti, Elisa Bertino, Arif Ghafoor, "A Trust-based Context-Aware Access Control Model for Web-Services," *Proceedings of International Conference on Web Services*, San Diego, CA, July 2004, pp. 184-193.

17. Junzhe Hu, Alfred C. Weaver, "A Dynamic, Context-Aware Security Infrastructure for Distributed Healthcare Applications," http://www.cs.virginia.edu/~acw/security/A%20Security%20Infrastructure%20for%20Distributed%20Healthcare%20Applications.pdf, Accessed on 10/19/05.

18. Gridsphere, http://www.gridsphere.org, Accessed on 01/19/05.

19. MySQL Open Source Database, http://www.mysql.com, Accessed on 09/26/05.

20. *SAML 1.1 Specification: Assertions and Protocols*, OASIS Standard, September 2003.

21. *SAML 1.1 Specification: Bindings and Profiles*, OASIS Standard, September 2003.

22. OpenSAML, http://www.opensaml.org, Accessed on 09/26/05.

23. Mohammad A. Al-Kahtani, Ravi Sandhu, "A Model for Attribute-Based User-Role Assignment," *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC)*, Las Vegas, NV, December 2002, pp. 353-364.

24. Marlon Pierce, Geoffrey Fox, Choonhan Youn, Steve Mock, Kurt Mueller, Ozgur Balsoy, "Interoperable Web Services for Computational Portals," *Proceedings of the ACM/IEEE conference on Supercomputing*, Baltimore, MD, November 2002, pp. 1-12.

25. Dennis Gannon, Rachana Ananthakrishnan, Sriram Krishnan, Madhusudhan Govindaraju, Lavanya Ramakrishnan, Aleksander Slominski, "Grid Web Services and Application Factories," *Grid Computing: Making the Global Infrastructure a Reality*, New York: John Wiley & Sons, 2003, pp. 251-264.

26. Toyotaro Suzumura, Satoshi Matsuoka, Hidyemoto Nakada, Henri Casanova, "GridSpeed: A Web-based Grid Portal Generation Server," *HPCAsia*, Tokyo, Japan, July 2004, pp. 26-33.

27. Soaplab, http://industry.ebi.ac.uk/soaplab/, Accessed on 01/18/05.

28. Kerry Taylor, James Murty, "Implementing Role Based Access Control for Federated Information Systems on the Web," *Australasian Information Security Workshop*, Adelaide, Australia, January 2003, pp. 87-95.

29. Von Welch, Ian Foster, Carl Kesselman, Olle Mulmo, Laura Pearlman, Steven Tuecke, Jarek Gawor, Sam Meder, Frank Siebenlist, "X.509 Proxy Certificates for Dynamic Delegation," *3rd Annual PKI R&D Workshop*, Gaithersburg, MD, April 2004, pp. 1-17.

30. Jun Wang, David Del Vecchio, Marty Humphrey, "Extending the Security Assertion Markup Language to Support Delegation for Web Services and Grid Services," *International Conferences on Web Services*, Orlando, FL, July 2005, pp. 67-74.

31. Michael J.Covington, Mattew J. Moyer, Mustaque Ahamad, "Generalized Role-Based Access Control for Security Future Applications," *23rd National Information Systems Security Conference*, Baltimore, MD, October 2000.

32. *Business Process Execution Language for Web Services Specification Version 1.1*, OASIS Standard, May 2003.

33. Ivan V. Krsul, Arijit Ganguly, Jian Zhang, Jose A. B. Fortes, Renato J. Figueiredo, "VMPlants: Providing and Managing Virtual Machine Execution Environments for Grid Computing," *Proceedings of the ACM/IEEE conference on Supercomputing*, Pittsburg, PA, November 2004, pp. 7-15.

34. Ming Zhao, Vineet Chadha, Renato J. Figueiredo, "Supporting Application-Tailored Grid File System Sessions with WSRF-Based Services," *In Proceedings of High Performance Distributed Computing*, Research Triangle Park, NC, July 2005, pp. 24-33.

35. S. Farrell, R. Housley, "An Internet Attribute Certificate Profile for Authorization," *RFC 3281*.

36. OASIS Web Services Security (WSS) TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss, Accessed on 10/09/05.

37. T. Dierks, C. Allen, "The TLS Protocol Version 1," *RFC 2246*.

38. R. Atkinson, "Security Architecture for the Internet Protocol," *RFC 1825*.

39. Sumalatha Adabala, Vineet Chadha, Puneet Chawla, Renato Figueiredo, José Fortes, Ivan Krsul, Andrea Matsunaga, Mauricio Tsugawa, Jian Zhang, Ming Zhao, Liping Zhu, Xiaomin Zhu, "From Virtualized Resources to Virtual Computing Grids: the In-VIGO System," *Future Generation Computer Systems*, vol. 21, no. 6, pp. 896-909.

40. DineroIV – Cache Simulator for Memory References, http://www.cs.wisc.edu/~markhill/DineroIV/, accessed on 01/04/05.

41. Jason Novotny, Michael Russell, Oliver Wehrens, "GridSphere: An advanced portal framework," *EUROMICRO*, Rennes, France, September 2004, pp. 412-419.

BIOGRAPHICAL SKETCH

Vivekananthan Sanjeepan graduated with a MS degree from the Department of Electrical and Computer Engineering at the University of Florida in December 2005. His research interests are distributed computing, networking, network security, operating systems, virtual computing and database management systems. Sanjeepan received his BS degree from the University of Moratuwa, Sri Lanka. He is a member of IEEE. Contact him at sanjee@ufl.edu.