A COMPONENT FRAMEWORK FOR BUILDING WEB SCIENCE GATEWAYS AND PORTALS

MEHMET AKIF NACAR

Submitted to the faculty of the Indiana University Graduate School
in partial fulfillment of requirements
for the degree
Doctoral of Philosophy
in the Department of Computer Science
Indiana University

March 2008

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

_____
Geoffrey Fox (Principal Advisor), Ph.D

_____
Donald McMullen, Ph.D

_____
Minaxi Gupta, Ph.D

_____
David Leake, Ph.D

November 27, 2007

Acknowledgements

First, I would like to thank my advisor, Professor Geoffrey C. Fox for his endless support and encouragement during all stages of my research. His advice and research projects were very crucial towards finishing my study.

I am delighted to thank research committee members Professor Donald McMullen, Professor David Leake and Professor Minaxi Gupta for their feedback to complete my dissertation.

I also owe thanks to Dr. Marlon Pierce, assistant director of Community Grids Lab for his insightful discussions and support. I am thankful to Professor Donald McMullen, director of Knowledge Acquisition and Projection Lab at Indiana University for friendly suggestions. I am pleased to thank Professor Gordon Erlebacher, Florida State University, for his feedbacks and criticism.

Without the friendly atmosphere in Community Grids Lab, my studies would not be finished. I would like to thank my colleagues Dr. Hasan Bulut, Dr. Kangseok Kim, Ahmet Topcu, Ali Kaplan, Ahmet Sayar, Fatih Mustacoglu, Harun Altay and Necati Aysan for their company. I also thank staff members at Community Grids Lab for their help and support.

Last but not least, I am also indebted to my parents for their patience and understanding. My beloved wife Aise Nur deserves many special thanks for her care, support and dedication. My lovely children Fatma Sevde, Omer, and Feyza also deserve thanks for their tremendous patience and devotion. Without their support, this research would never have been completed.

Abstract

Portlet-based Grid portals have become a crucial part of the international distributed computing infrastructure to support computational science ("cyberinfrastructure") provide component-based problem solving environments for scientists. Although Web portals are intended to provide user-friendly environments with easy-to-use interfaces, the development of portals and their portlet components is time consuming. We aim to address this problem by providing reusable components for rapid portlet development. Our approach, Grid Tag Libraries and Beans (GTLAB), encapsulates common Grid operations with reusable XML tags, providing a dramatically simplified programming interface to common cyberinfrastructure services. GTLAB also provides a way for creating composite tasks that models the requirements of computational science portals. In addition to standard Grid job submission and remote file operation tags, we also provide management and monitoring capabilities for Grid tasks. This system can persistently store job metadata, which results in a permanent storage for archiving and reference.

In this dissertation, we have studied and observed two distinct science gateways as use cases. First, the QuakeSim portal is a problem solving environment to develop a solid Earth science framework for modeling and understanding earthquakes. In this study, we have proposed an evolutionary approach to allow TeraGrid usage in addition to clusters for QuakeSim portal. Second, VLab is a Grid and Web Service-based system for enabling distributed and collaborative computational chemistry and material science

applications for the study of planetary materials. The requirements of VLab include job preparation and submission, job monitoring, data storage and analysis.

Although GTLAB provides support for simple Grid workflows, we must also investigate the problem of integration with existing workflow systems. We have thus extended GTLAB to support the widely used Condor DAGMan and Taverna workflow systems. These extended tags demonstrate that large workflows can be integrated within Grid portlets without burdening the developers.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Over the last decade, the improvement of Web technologies has produced *e-Science* [1], in which scientific communities adopt, extend, and influence the developments of Web computing. Grids and Grid computing [2, 3] provide the distributed computing infrastructure ("cyberinfrastructure") foundations for e-Science activities. Communities want to see daily impacts of their research and arrange daily activities based on given information. Many science applications are broadcasted over the Internet through portals. Examples range from Earth sciences to space exploration. While Genomic research encrypts genetic sequences of the cells in micro level, space research seeks knowledge about the universe in largest possible scales. Scientific Web applications in a sense monitor all the aspects of human life with instruments from microorganisms to cosmos.

The efforts of building science knowledge management environments can be categorized in two aspects: 1) Core Grid applications that serve to construct foundations, and 2) management of the Grid applications. Figure 1.1 illustrates three-tier architecture that contains resource providers, middleware and portals.

Grid infrastructure spans multiple organizations in different administrative domains (that is, creating "Virtual Organizations" [2]). Each autonomous institution provides computing resources and data capacity through organizational boundaries. Therefore, Grid resources must respect the security and privacy concerns to tie resource providers and scientists. Examples of large virtual organizations providing data storage, computing power, and Grid services include TeraGrid [4], the Open Science Grid (OSG) [5] and Enabling Grids for E-SciencE (EGEE) [6].

The Grid initiatives must provide middleware layer to access supercomputing resources and data seamlessly. Grid middleware that is serving Grid users includes the Coordinated TeraGrid Software and Services (CTSS) [7] on TeraGrid, the Virtual Data Toolkit [8] on OSG, and gLite [9] on EGEE. Grid middleware services range from security, file management, information service, and schedulers.

Grid Computing Environments (GCEs) [10, 11] provide a user view through the client tier of computational Grid technologies. GCEs are often associated with Web portals, but in general they may be any type of client management environment. GCEs also come in two primary varieties: Problem Solving Environments (PSEs), which provide custom graphical user interfaces for working with specific sets of applications, visualization tools, etc; and shell-like system portals, which provide direct access to

Grid middleware such as file manipulation and command execution. These terms also evolve over the time. Efforts to build GCEs are now often called science gateways [11].

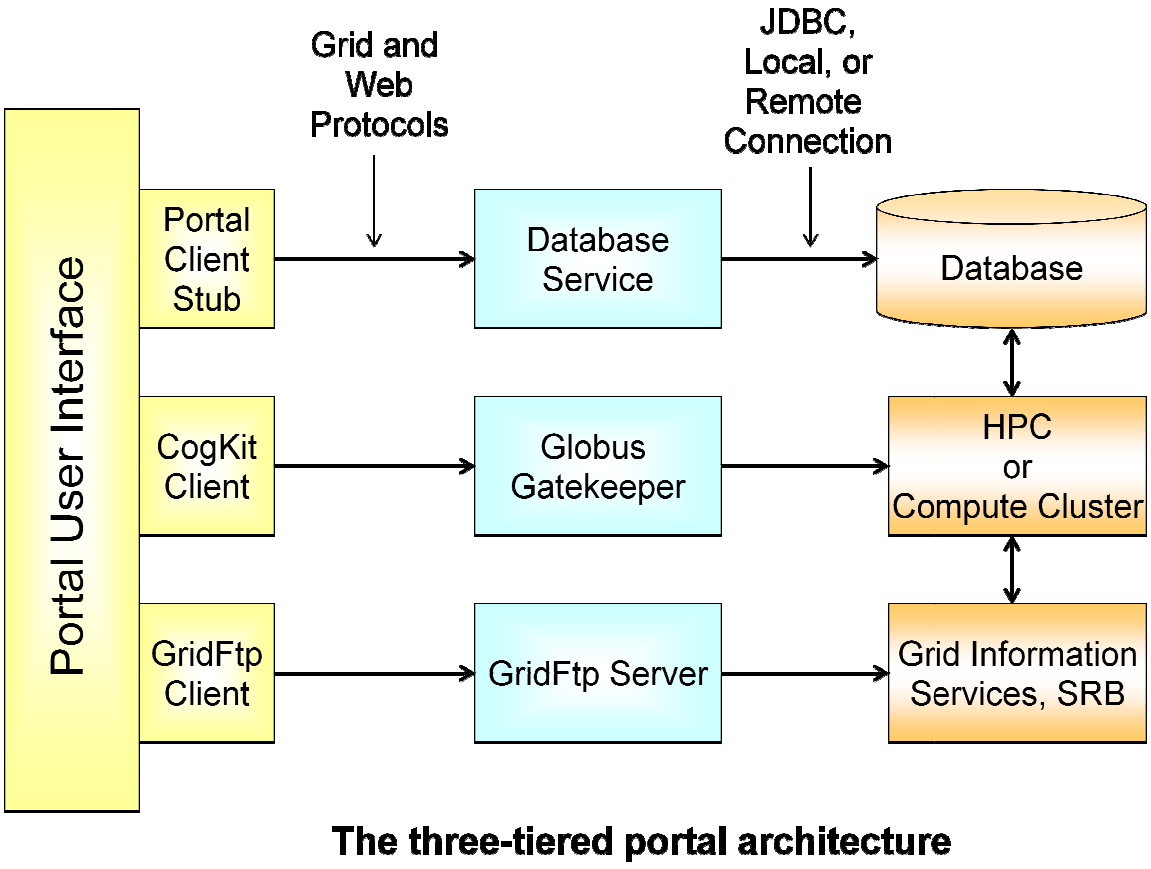


Figure 1.1 Classic three-tier architecture illustrates the portal in the client tier (left), service middleware in the middle tier and computing and data resources in the resource layer (right). The core of our work will address the components used to assemble the client tier.

There are a variety of application portals available to solve numerous problems. Examples range from atmospheric discoveries in Linked Environments for Atmospheric Discoveries (LEAD) [12] to virtual observatories described in the National Virtual Observatory (NVO) [13]. The QuakeSim portal [14] is a problem solving environment to develop a solid Earth science framework for modelling and

understanding earthquakes. The Virtual Laboratory for Material Sciences (VLab) project [15] is an example of a science gateway for computing the properties of planetary materials under extreme conditions and provides the specific motivating cases for our research.

A science portal supports the work of a scientific team or community by combining a Web portal and associated Web Services. By using a Web browser, a scientist can access both private and shared workspaces of discipline-specific data and tools. Science gateways are also access points to Grids of computational and data resources, allowing a user to leverage the capability of a Grid without forcing the user to deal with the complexity of Grid technology.

Science portals have been developed for over a decade, and much progress has been made to standardize their architecture and component models. Many science gateway initiatives use the so-called portlet component model, defined by the Java Specification Request (JSR) 168 [16]. Many open source JSR 168 containers have been implemented, with the GridSphere [17, 18] container serving as a very popular implementation in the scientific community. Other examples include uPortal [19], Pluto [20], Jetspeed [21], Sakai [22], LifeRay [23], JBoss [24], and eXo [25]. General purpose, pluggable Grid portlets for remote job submission, interactions with Grid information services, remote file management, and security credential management have been developed by the Open Grid Computing Environments (OGCE) [26, 27] collaboration and the Grid Portlets [28] project. Most Java-based Grid portals use the Java COG kit [29] to build their Grid clients.

Most Web applications employ server-side technologies to provide rich dynamic content to its viewers. Java Server Faces (JSF) is a dynamic Web application framework that is similar to other dynamic Web application templates like velocity (http://velocity.apache.org), Java Server Pages, and Spring (http://www.springframework.org). JSF provides dynamic content using Java servlet technology. Dynamic content is generated based on the request and response paradigm. Unlike static HTML pages, Web application frameworks allow interaction with users. The server pages take requests, process them and respond to the user through a Web interface. Besides its inherent virtues, we believe that JSF is very well suited for science gateway development, as we will discuss.

JSF applies the Model-View-Control (MVC) software design pattern [30, 31] to decouple data models, action controllers and user interface widgets into separate components. Within the framework of JSF, the model corresponds to a backing JavaBean: a piece of Java code that is responsible for managing application data. The backing bean itself is typically a client to a database or a remote Web service. One of JSF's hallmarks is that these beans can be developed outside the JSF framework; that is, it is not necessary to import any JSF-specific code. This means that the beans can be reused in non-Web based applications and can run on stand-alone applications. This is accomplished through a design pattern known as "inversion of control" [30]. The controller corresponds to a JSF servlet that manages user requests. Finally, the view corresponds to Web interfaces of JSF. This architecture separates the data access and user interaction. MVC encourages the reuse of backing beans within different applications.

We need a component model for User Interface (UI) matching Grid services as a component model for Grid middleware. Portlets attached to services do not work except in simple cases. There is no good way to link portlets, therefore most UI's need workflow services. Therefore we must use tags as component model and allow multiple tags in a portlet application. Our solution to building science gateways is providing a modular component framework. This framework provides reusable parts (tags) to construct portlet pages as well as composing and managing these parts.

1.1 Motivation

Scientific knowledge is shared by the community in terms of its needs and levels of understanding. To accomplish this missions, Grid portals [32] distribute the knowledge resources on the Web and restrict accesses to the groups of people. In that sense Grid portals are science gateways for people ranging from researchers at national research labs to school children. Therefore, all age groups and educational levels can access to the information thorough science portals on demand.

In the Grid portal field there are numerous efforts to build intuitive gateways and construct standards to build these applications using common platforms that help to interchange the knowledge between the institutions. These developments require computer scientists to study the Grid portals field.

Grid services vary from Globus toolkit [33], Condor [34], Unicore [35], and gLite [9]. We want to provide generic and ready to use clients for these services. There are efforts to provide programming level abstractions to the Grid such as Java Commodity Grid Kit (CoG Kit) [29], Simple API for Grid Applications (SAGA) [36] and Portlet

Vine [37]. One of our goals is to utilize Grid resources transparently that range from TeraGrid resources to our Gridfarm clusters at Indiana University.

The first generation Grid portals for interacting with this middleware were stovepipe solutions to fit the needs of scientists that basically have capabilities to run and manage applications through Web applications [10]. Second generation portals are aware of the need to build reusable software components and propose a common framework for scientists. The Java Specification Request (JSR) 168 standard [16] and portal frameworks provide a standard portlet specification that defines programming interfaces and portlet container model.

The challenge for the community now is to define the third generation of Grid portal technology. In our research we have brought the problems of second-generation portal development on the table, and we have proposed solutions to these research area. We have proposed to model fine-grained Grid portal components that can flexibly compose client tier applications to Grid capabilities.

1.2 Problem Statement

Emerging Web technologies leverage Grid communities to use Web resources efficiently within the GCEs. As a result, legacy applications can be accessed through Web interfaces. These changes impact applications such as preserving the access to the restricted resources. We need to maintain the policies and rules that exist in the low level application and map them to the Web application level smoothly. Thus, Web applications are converging to look like desktop applications over the time. Web

applications must provide look and feel pages and customization to access the back end resources.

To achieve interoperability and to avoid relying on unsustainable custom solutions, science portals and e-Science generally follow Web specifications closely. For the portal community, the dominant specification has been JSR 168. After completing repeated efforts on building OGCE and QuakeSim portal [14, 38] we have learned the shortcomings of JSR 168. An important aspect of the JSR 168 is to build portlet components by implementing low level application program interfaces. There are two drawbacks of this approach: i) Portlets are coarse-grained components that wrap entire Web applications, not their functional components; and ii) portlet implementations are not reusable if one needs to combine one or more portlet capabilities. Consequently, we conclude science gateways have an inadequate component framework: JSR 168 is a useful starting point, but insufficient to develop science gateways in all aspects.

Our research is about a novel approach building a better component model for Grid portals that can enable all Grid operations in one application. Unlike the previous OGCE Grid portlets, Grid operations can be contained in portlets with reusable widgets as shown in the snippet at Figure 1.2. This approach adds one more layer on the JSR 168 portlet container, since it wraps JSR 168 programming interfaces with reusable widgets. The new layer may cause performance degradation and latencies because of additional processes. In our study, we measure the latency to show whether it is acceptable.

```
<html>
  <body>
    <f:form>
      <o:submit id="test" action="next_page" />
        <o:myproxy id="pr" hostname="gf1.ucs.indiana.edu"
              port="7512" lifetime="2" username="mnacar"
              password="***" />
        <o:jobsubmit id="task" hostname="cobalt.ncsa.teragrid.org"
              provider="GT4" executable="/bin/ls"
              stdout="tmp/result stderr="tmp/error" />
      </o:submit>
    </f:form>
  </body>
</html>
```

Figure 1.2 Grid tags are used to build a sample portlet application that calls services including: a) Myproxy service to get user credential, and b) GRAM service to execute a script.

There are many ways to deal with a collection of tasks in sequence. This approach leads to workflows, and workflows are commonly used in the Grid community. The complexity of applications and staging requirements enforce workflow usage. Grid applications apply existing Grid workflows [39] to process sophisticated execution flows. Our research scope covers strategies to leverage these workflow clients to integrate to Grid portals. The problem here is to find out the correct representation of workflow execution and monitoring within Grid portals. Workflow composition is out of scope of our research.

Another problem of portals in general is to provide access control to the portlet contents. That is, we must consider what happens after a workflow is executed and data is generated. This is particularly important when making a scientist's unpublished scientific data and computations available through Web interfaces: only the scientist and designated collaborators should have access, although many other people will use the portal (and the same components) to view different data sets. Portals frameworks restrict user or group access rights on portlet components. Users will see the same content of accessible portlets no matter what their rights are or what group they are in. The portlet Application Programming Interface (API) does not constrain the content access; in other words there is no policy enforcement on customizing portlet contents per users. We studied enforcement methods to restrict the portlet access to the user groups such as administrators, directors, researchers, providers, customers, study groups, students and anonymous users.

1.3 Research issues

When we are seeking solutions to the all problems listed on the problem statement, we have raised many critical research issues that we believe contributes the current field. We will list major research problems including the level of granularity on Grid operations. We observed these leading problems are following: a) Grids are complicated, b) existing client tools are also too complicated (e.g., the COG), and c) portlets are not an adequate component model for Grids. Therefore, our main research focus is about finding correct architecture for Grid operations including proxy credential, file operations and transfers, and job submission.

The key research issue is the lack of a simple client interface to Grid services, particularly for Grid portals. There is also the need for a component environment compatible with Web Model-View-Controller architectures. This component model should be extensible beyond any specific implementation.

How do we apply Grid components to Web applications to build modular portals? There are server-side Web application frameworks to build portals. JSR 168 is one of the standard portlet container that support servlet-based Web applications. Velocity, Java Server Pages (JSP) [40], Struts and Java Server Faces (JSF) [41, 42] are some of the technologies to develop portals. JSF is interesting because its [43] component-based architecture fits better to our component model. JSF is an MVC approach to data binding. Therefore we have chosen JSF application framework to prototype Grid tag libraries. The detailed discussion of JSF is given in Section 5.6. However, we are able to extend our model to any other server-side Web framework.

One of the problems with the Grid operations is how to represent them in the context of Web applications. Java servlets [44] already provide an API to program server-side applications by using object oriented programming paradigm. Thus, application logic in the context of Web applications is reusable among different applications. But there is no way to reuse server pages in presentation logic. To come up with solutions for building modular Web application pages, we introduce widgets as tag libraries. Tag libraries encapsulate reusable Grid components so that allowing an extensible architecture for future additions.

Another interesting research problem is applying workflows and their dependencies in a generic way to Grid portals. There are different approaches to apply workflows on

demand. Some science disciplines closely rely on composing workflows such as life science workflows defined in [45]. Thus, they need graphical tools to generate complex workflows easily. Other types of scientists usually work on the well-established workflows composed by workflow authors [46]. They only need to enact and monitor the workflows. Depending on the nature of our case studies, we have focused on the first type of workflow demands by implementing graphs. Then we studied workflow enactment and monitoring integrated into Grid portals.

How do we handle and identify the jobs within a user context? One of the research problems we have to deal with is user metadata management. For example, a user session lasts from when a user logs into a portal until the user logs out or session lifetime expires or browser window is closed. We needed to have a mechanism to manage user metadata for tracking Grid jobs that running on Grid servers. Therefore we need to persist and store job metadata within a user's context for both archiving and reusing reasons.

We have identified another interesting research issue in the scope of portlet content authorization, when we developed the Common Instrument Middleware Architecture (CIMA) [47] portal. We need to find out how we can deal with user access rights and groups in the context of collaborative environments. There are authorization frameworks that comply with these requirements. But the problem is portals do not apply any of this authorization schemes in the scope of portlet contents. We have worked on CIMA case to find solutions that are explained in Section 7.3.

1.4 Grid Tag Libraries

Grid Tag Libraries and Beans (GTLAB) [15, 48] provide a set of JSF tag libraries for Grid portal development. This library encapsulates atomic Grid operations as well as multi-staged operations. We explain GTLAB component model and its job management capabilities in Chapter 5.

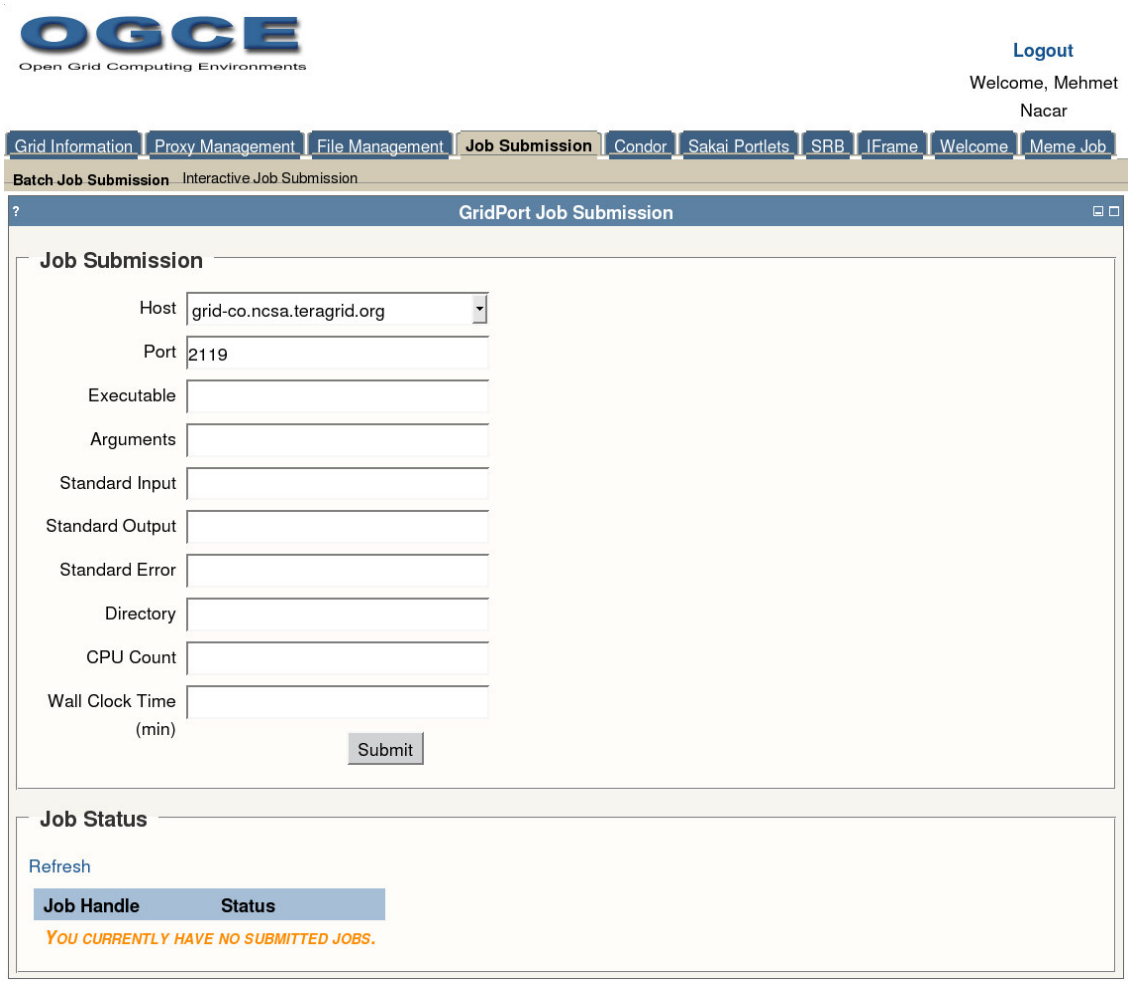


Figure 1.3 OGCE portlets screenshot where all Grid operations are implemented as portlets (i.e., tabs). The portlet shown is a generic interface to Globus GRAM middleware.

Although OGCE portlets are functionally similar to GTLAB, OGCE portlets are based on standard Web applications and portlet API. Portlet API defines a specification

[16] for portlet developers to reuse portlets among different portal frameworks such as GridSphere, Liferay and Jetspeed. Thus, Application developers have to customize the portlets to comply with specific needs of the gateways. Another aspect of OGCE is that the Grid capabilities are separate portlet applications in Figure 1.3. Developers need to assemble several portlets to get workflow capabilities. The problem is that portlets are not composable elements. They are independent Web applications that live in a container. It is not possible to express dependencies for example between portlets. All these efforts require substantial effort of programming. The developers need to reuse and modify some of the codes, view pages, configuration and deployment descriptors. On the other hand, GTLAB enables all capabilities within a Web application that requires minor customization on the view pages. All other APIs, libraries, and deployment descriptors will be the same.

To provide a component model for Grid portals, we must provide abstract and extensible interfaces and APIs. The advantage of this approach is that new tags and beans can be added by deriving the interfaces. For example, Condor and Taverna support can be added in the same way.

1.5 Contributions of this research

The major contribution of this thesis is to provide a component framework for science gateways. We have designed, developed and applied such a framework through several science gateways in various fields. We have studied the problems and our approaches to the solutions by designing Grid tag libraries that are the building blocks of the Grid portals. Grid tags are reusable and customizable on portal platforms instead

of re-implementing all capabilities for each portal instance repeatedly. We must therefore design Grid operations as components that application developers can get them together to build Grid portals. This work is discussed in Section 5.2.

Aggregation of services and capabilities requires control over the flow of the execution which in sense enforces us to match workflow mechanisms to the portal. Grid portals naturally require one or more Grid operations running in a sequence that is defined in dependency rules. Generally simple flow controls can be represented as a DAG. We also have investigated applications going beyond simple DAGs with the full-fledged workflows. The example usage cases may include using filters to refine data, orchestrating Web services and Grid services within a context of workflow. This work is discussed in Section 6.6.

Grid tag libraries framework manages user accounts within portal sessions. User sessions are able to list ongoing jobs from previous sessions. Metadata and status information can be monitored by the Grid users.

Persistency feature stores and archives metadata information about Grid tasks in the permanent storages for each user. Although metadata information is stored for archiving, they are also used for resubmitting the same parameters for repeated experiments. Collecting all these features together, Grid tag libraries provide a component framework for Grid community.

We have experienced that Grid tag libraries provide rapid development for application programmers without any additional overhead on runtime. For the VLab portal we have composed DAGs for the several VLab usage scenarios. Examples include

- Collecting input parameters and files needed to run computational experiments.
- Moving output files and staging them to visualization services and monitoring them in overall.

We also have applied Grid tags to QuakeSim Eartquake science portal to run Disloc [49] and Geofest [50] applications and monitor the progress of these simulations and manipulate output data to generate mash-ups, plots and visualizations.

We have summarized the Grid portal efforts of scientific communities including VLab, CIMA and QuakeSim. These use cases have shown that similar issues and solutions are applicable to different community requirements. Our generic Grid tags have been applied to these portlets by adjusting configuration and customization principles for each case. Therefore we have observed rapid development of these portals.

1.6 Organization of the Thesis

The organization of the rest of this thesis is as follows:

Chapter 2 surveys the state of the art technologies used within science gateways literature.

Chapter 3 builds background on Grid computing environments. It summarizes Grid and Web services as gateway to virtual organizations. We also evaluate Grid portal approaches and techniques that used in the past. We motivate through usage scenarios.

Chapter 4 mentions the case studies that we have developed as Grid portals. VLab, CIMA and QuakeSim portals are our major applications.

Chapter 5 presents the foundations of Grid Tag library approach and evaluates it in architectural perspective. We mention about design principles of Grid tags and beans. We identify the modules of Grid tags are: session manager, component parser. We also showed that how Grid tags deal with DAGs and workflows and handle persistency and archiving.

Chapter 6 covers workflow and DAG management for various Grid workflow mechanisms including OGSA services, Condor DAGMan and Taverna.

Chapter 7 mentions about access control mechanisms on portlet contents and our solution to the CIMA portal problem.

Chapter 8 summarizes the work done in this thesis, highlights major contributions, discusses possible extensions of Grid tag libraries, and presents future directions for our research.

Chapter 2

Background

2.1 Overview of Grid Computing Environments

Grid Computing Environments (GCEs) [10] provide a user view of computational Grid technologies. GCEs are often associated with Web portals, but in general it may be any type of client management environment. Web portals are accessible gateways to the resources and services. GCEs come in two primary varieties:

- Problem Solving Environments (PSEs), which provide custom interfaces for working with specific sets of applications, visualization tools, etc; and
- Shell-like system portals, which provide direct access to basic commands such as file manipulation and command execution.

This situation began to change rapidly in early 2002 with the emergence of two important concepts: reusable portal components (portlets) and Web service

architectures. Java portlet components became standardized with the Java Specification Request JSR 168 [16]. Web services architectures are summarized in [51]. Modern portal systems have adopted these two cornerstones and follow a general architecture. Standard-based portlets provide reusable functional components that can be shared between different portal installations. Web services decouple the portal functionality from its presentation layer.

We will summarize the efforts from building GCEs to science gateway in last decade.

2.1.1 GCE Shell

We describe the design and features of our Grid Computing Environments Shell system, or GCEShell [52]. We view computing Grids as providing essentially a globally scalable distributed operating system that exposes low level programming APIs. GCEShell environments are separated from specific user interface rendering. GCEShell consider here a general engine for managing Grid and Web service clients. This GCEShell engine is initially implemented as a command line interface, is inspired by the Unix shell environments [53], which provide a more user friendly environment for interacting with the operating system than programming directly with system level libraries.

2.1.2 GPDK

The Grid Portal Development Kit (GPDK) [54] provides classical three-tiered architecture for middleware applications. GPDK operates Model View Controller (MVC) [30] to separate control and presentation from the application logic for

accessing Grid services. GPDK utilizes large-scale scientific applications through Web interface. GPDK has its own Portal Engine (PE) that redirects user requests to the server and responses from the server. PE implements essential user management, application management and presentation services for GPDK portal.

2.1.3 Gateway System

Gateway system [55] is one of the early computational portal effort that provides seamlessly secure and uniform access to computational backend services. It applies three-tiered architecture by supporting distributed component-based middleware. Gateway Web portal implement services including user interface, metadata, security and shared visualization.

2.1.4 GridSphere Portal Framework

Until the portlet concept started with JSR 168, there were stovepipe Web applications for Grid services. GridSphere [17, 18] is one of the first implementations of JSR 168, along with uPortal [19] and Liferay [23]. GridSphere can be used to build Web portals that are compatible with JSR 168 portlet container standards. GridSphere additionally provides Grid portlets that access to a number of services. GridSphere's Grid portlets provide a set of capabilities that supports Grid services available by the Globus toolkit, including Globus Resource Allocation Manager (GRAM) [56], GridFtp [57], Monitoring and Discovery System (MDS) [58], MyProxy [59], Web Service Resource Framework (WSRF) [60] for GT4 and Open Grid Services Architecture (OGSA) [61]. GridSphere container additionally provides portlet services for Grid. Thus, GridSphere Grid portlets are strictly dependent on the GridSphere portal

framework; as a result these portlets are not portable among portal containers. However, the Vine project [37] is trying to decouple Grid portlets from GridSphere. This effort is extensible for stand-alone applications through Grid portlets.

2.1.5 OGCE Portlets

OGCE portlets [27] are built on Velocity and provide access to common Grid services through the Java CoG abstraction layer [62]. OGCE also provides portlets for Condor and Storage Resource Broker (SRB) services [56]. These portlets are compliant with JSR 168 and portable among portal frameworks. For example, one can deploy OGCE portlets on either GridSphere or uPortal. Each portlet provides a single Grid capability. JSR 168 does not support inter-portlet communication in its specification; however, OGCE portlets has simple in-memory hash-tables that share session data between portlets. This is typically used to access proxy credentials.

2.2 Typical Grid Portal Usage Scenarios

Grid portals address wide science community problems ranging from atmospheric discoveries to virtual observatories. There are various types of users needed to be served. There are also numerous applications to being utilized within portal environment. To overcome these problems, there are different types of Grid portal approaches exist starting from GPDK, the following OGCE and Grid portlets of GridSphere.

In this chapter we discuss several different use cases with scenarios to build a Grid portal. A science gateway building process starts with demands from research/science communities, such as if they have long running applications in their lab and they want

to expose these applications to the world and to observe their impact. They need to use the Web and broadcast the dynamic information to international communities. First, Grid portals need to design efficient interfaces to enable scientists to do their work more effectively. Second, they need to advertise their backend services and make them available through standard interfaces like Web services. Third, when they are doing all this web development, they also need to keep their services and data secure. In this case, they need to enforce policies for user authentication and to describe access rights of users. These are classic Computer Science problems, but they become much more difficult in distributed environments.

The Grid community wants to discuss problems at appropriate scope levels. For instance, Grid providers have problems publishing the services on a broad range of platforms, Grid users willing to access capabilities that are aggregated on a gateway and application developers wants to comply with all these needs. Thus, Grid architects have to come up with solutions to these issues in great detail, although they need to provide high level tools. Successful practices and useful science gateways come from following the requests mentioned before. For example, LEAD, NVO, VLab, QuakeSim, and CIMA portal can be considered that implementing requirements accomplished successfully. There are success stories about these portals in applications chapter.

2.2.1 User scenario for VLab portal

VLab portal is primarily focused on computational material science environments, by collaborating scientists, sharing information and keeping repositories. want to support the Web technologies for easy to use and portable environments. The Plane-

Wave Self-Consistent Fields (PWscf) [63], part of the Quantum Espresso suite is used to do simulations for computational material science. It is desirable to make PWscf into a gateway because PWscf requires challenging processes. These processes include computational experiment preparation, running applications and evaluating results either in the visual or non-visual environments. Our main contribution to VLab is managing the complicated flow control of processes by the Web interfaces.

Any given tasks might involve input file generation, code submission, data analysis, and visualization. These tasks are often linked and have dependencies among them. A need thus arises for a workflow and associated software, such as a workflow tags. Workflows might themselves be composed of simpler workflows. An example is given that illustrates the PWscf job submissions in section 4.2.

After we describe the community demands, we evaluate different Web application frameworks and portal frameworks. We found GridSphere and JSF are the best candidates, because JSF provides component based Web environment compared to equivalents JSP and Velocity. GridSphere provide JSR 168 standard portlet component model with a robust framework, well supported and maintained. We then developed a fine grained component model to build portlets out of reusable tags. This improvement guarantees lower cost and time contribution, and rapid development.

We have studied security extremes on the CIMA portal case, besides application management concerns.

2.2.2 User scenario for scientific workflows

Users are looking from their specific perspective. When you talked to people from different communities, you will see the problems and concerns are similar. We can group these concerns based on our observations as following:

1. Efficient user interface design
2. Manage services and applications
3. Follow the progress
4. Display the results
5. Manage all of the above in a workflow
6. User account management

These demands are common and there are common solutions to these problems. PSEs, Grid portals and science gateways all seek solutions to these issues. There are practice of submitting jobs, account management, interface design and display technologies available. But we wanted to introduce supporting workflows that tied into Web application framework as components. Thus, we sketched a solution to all possible demands from different communities. We applied an abstract component model which is tag libraries. For example, we can design interface and workflow for VLab people and deploy it as a science gateway. Similarly, we can design a different interface and workflow by using the same component model and make CIMA science gateway available.

2.2.3 User scenario for access control of portlets

CIMA users collaborate on crystal sample data such as crystallographers giving feedback for the samples from different labs. There are crystal owners, labs, and

students on CIMA portal. Crystal owners are responsible for publishing samples and their data. Crystallographer analyze and classify the samples, users can see their own samples and the others' samples unless they are private. Crystallography labs are responsible for broadcasting video and lab conditions real-time. Students demand all these capabilities on the portal to understand crystal features. In some cases, students will be replaced by pharmaceutical companies that demand crystal analysis.

CIMA users have roles when they registered to the portal in first place. Portal administrators assign the users to the relevant groups. CIMA roles and groups restrict user access rights to the resources. Thus, every other user can see different sample data on their window, and they can customize display on demand. To this extent, CIMA portal is a science gateway to Crystallography people that they can share data and make collaboration on the data.

2.2.4 User scenario for Web 2.0 portals

In this aspect, we will evaluate Web 2.0 usage cases to build portals. Most of the portal engines are heavyweight server page applications such as JSP, JSF, and Struts. Google iGoogle [64] and Netvibes [65] bring new portal environments to the Web. Web technologies usually constructed on request/response paradigm and support synchronous communication. Web 2.0 techniques not only support request/response model, but also enable asynchronous communication. Thus, Web pages can handle events occurred similar in desktop environments. Web 2.0 methods also brought new components called Web gadgets that integrated to the desktop environment. Gadgets are easy to develop and useful tools that binds Web features to user desktop.

In one step further, we want to build Grid portals out of Web 2.0 widgets. Thus, Grid portals run as thin clients on the client side browsers. Grid gadgets will exist in public repositories and the users can integrate these features to their portal pages with no hassle. As Web 2.0 tools growing up, we need to adapt existing security methods to protect Grid resources without any breaches. So far Grid servers apply tight security policies and mechanisms to Grid services. Gadgets also should support these security measures to end up with integrity.

2.3 Grid programming interfaces

2.3.1 Java Commodity Grid Kit

Java Commodity Grid (CoG) Kit [66] enable Grid users, Grid application developers, and Grid administrators to utilize, program, and control Grids from a higher-level framework. Java CoG Kit allows for easy and rapid Grid application development. It also encourages collaborative code reuse and avoids the duplication of effort among problem solving environments, science portals, and Grid middleware.

OGSA Grid services are interfaced by Java CoG abstractions [62]. These programming interfaces have capabilities to generate proxy certificates, submit jobs, transfer files and make file operations. They also provide composite task submissions and their handling. We use Java CoG abstractions to build OGSA based Grid clients.

2.3.2 Condor Web Services

Condor provides a Web services interface called Birdbath [67]. Birdbath aims to augment some of the core Condor daemons with Simple Object Access Protocol (SOAP) [68] interfaces so that they can be queried and controlled through programs

other than the standard Condor command-line tools. This provides an XML abstraction of the programming interfaces that can be bound to any programming language such as Java. Therefore, Birdbath client stubs can be generated from the Birdbath Web service interface. We have built Birdbath client stubs as an API to program Condor Grids.

2.3.3 Simple API for Grid Applications (SAGA)

SAGA [36] defines a Grid programming API on top of Grid middleware. It aims to abstract Grid Middleware accesses and provide a common programming environment. Such Grid services are Globus, Condor [69], Unicore [35] and Secure Shell (SSH) [70] clients. This approach adds an additional layer in between Grid middleware and clients. They claim that the cost of this layer is tolerable in compare to benefit. They provide task processing models including DAG representation. SAGA differs from Java CoG Kit by providing a programming language independent API.

Chapter 3

Survey of technologies

3.1 Introduction

In this chapter we review state of the art in science gateways and best practice technologies. Various important specifications and industry standards exist, and these must serve as the starting point for our work in Grid portal area. The next sections summarize Grid, Grid portals, and Web services technologies. Portal frameworks and portlet standards are reviewed in the later sections. We then review Web application frameworks after Grid account management.

3.2 Grid Portals

Grid portals [32] are science gateways for providing scientists a problem solving environment in which they execute distributed Grid applications from Web browsers, desktop tools, or mobile devices. Grid Portals provide seamless access to Grid services and resources. However, as science gateways evolve, novice users with a broad range

of abilities can be supported by Science portals. Grid portals have advantages as follows: a) provide single point of access to distributed information and services, b) utilize Grid services on behalf of the user, c) endure environment changes at remote hosts such as policies sometimes differ and allocations of servers change.

In early stages of building science portals, stovepipe solutions have been used to build Web interfaces for Grid portals, but the usage of emerging technologies among Grid community required them to adopt a portlet component model for Grid portals.

A number of Grid portals exist including QuakeSim portal for earthquake sciences, VLab portal for Material Sciences, TeraGrid User Portal [71] for managing TeraGrid resources and LEAD portal for atmospheric discoveries. Grid portals utilize Grid resources, services and data on virtual organizations. Grid portals [32] address the problems of building Web applications to enable virtual organizations on the Web. These problems can be summarized as below:

- Security services
- File transfer management
- Job management
- Accessing to metadata services
- Resource sharing

Grid portal efforts generally have focused on Grid portlets. Grid services are wrapped by equivalent client components that are called Grid portlets. Grid portlets generally points out solutions can be summarized below:

- Supporting portlet development
- Simplifying the development of Grid clients

- Integrating collaboration tools to support user communities
- Developing supplemental Grid services to manage science applications and data.

3.3 OGSA and WSRF services

Open Grid Service Architecture (OGSA) [3] is a comprehensive architecture for the Grid based on Grid services and Web services. Web Services Resource Framework (WSRF) [60] is a set of specification by OASIS [72]. These specifications describe how to implement OGSA using Web services. Globus toolkit (GT) [33, 61] is one of the well known Grid service providers that support OGSA (GT3) and WSRF (GT4) based services . Virtual organizations (VO) can be built using GT. VOs either manage clusters on the systems where machines are locally located or VOs can construct a Grid out of several other VOs as well. The major problem of constructing Grids among different administrative domains is the requirement of major security policy enforcement. GT introduces Globus Security Infrastructure (GSI) [73] security mechanism that is credential based security that uses X.509 certificates. In this approach, there two types of certificates: 1) resource (or see terminology) certificate, 2) user certificate. All certificates are created and signed by Certificate Authorities (CA). There is CA signing hierarchy to allow different administrative domains to trust. User certificates are required to access VOs. Users also are registered to grid-map file which is an access control file for the Grid. Grid services have to comply with GSI security.

Starting in 2001, Globus team released GT2 release and the following major releases respectively GT3 and GT4. GT4 services are provided on Web services framework. Globus services are listed as following: GRAM, MDS, and GridFtp.

3.3.1 Globus job management services

The GRAM service enables Linux based systems to run remote scripts by using daemons called Gatekeepers. Web Service GRAM (WS-GRAM) service differentiates from non WS-GRAM services by using 'sudo' accesses on Linux. GRAM service basically runs any command line script on behalf of the user. To abstract command line shell, they have introduced RSL [2] description language. The disadvantage of GRAM is to use socket connections that sometimes causes problems when the systems use firewalls.

3.3.2 Globus File Management Service

Another crucial service is to transfer files between remote hosts so called GridFtp. It is a FTP protocol that builds GSI security on top of it. GridFtp allows third party file transfers as well as file operations such as making directories, renaming files, deleting files etc.

3.3.3 Monitoring and Discovery Service

MDS provides metadata information about Grid system. In other words, MDS catalogs resource information with an annotation schema. This service gives users information about system maintenance, number of machines on the Grid, the load of system, available compute power, amount of data storage etc.

3.3.4 Condor

Condor [2] services are specialized providing compute intensive resources for high throughput computing. Condor can utilize Grid resources across administrative

boundaries. Condor-G [69] incorporates Grid technologies and provides interoperability with VOs are managed by Globus. However Condor is only available on the shell environments and being managed by using command line scripts. This drawback makes Condor platform dependent. To allow platform decoupling, they have used Web services and have provided Birdbath [2] services. Thus, using Birdbath service interface, you can generate client stubs for any platform such as Java, Python, and C#.

Although Condor-G allows user submitting jobs to run on high performance resources, it does not support multiple job submissions. To overcome this side effect they have introduce Condor DAGMan [2] to compose DAG graphs among Condor resources.

3.3.5 Credential management service

MyProxy is a credential repository introduced by [2]. MyProxy stores credentials on behalf of users in repositories. When user wants to use its credential, it gets the credential by using username password pairs different than certificate pairs. MyProxy prevents to type credential password frequently.

3.4 Web services

Web services provide a standard means of interoperability among software applications and frameworks which are called services. Web services interactions can be done using message exchanges between client and server. As a result Web services utilize request/response paradigm by applying Simple Object Access Protocol (SOAP) [68] protocol on the wire. A more general definition of Web services quoted below

A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by internet protocols [51].

Web services are defined using the Web Services Description Language (WSDL) [74]. WSDL describes the location of services, types of ports, method descriptions, types of messages so on. WSDL is a platform independent XML document which Web services clients can be created from. Therefore, many different clients can talk to Web service using WSDL description through SOAP channels.

SOAP is a wire transport protocol on top of Transport Control Protocol (TCP) layer. SOAP basically provide request/response paradigm by using message envelopes. Each message is packaged in an envelope with a return address. Envelopes contain the message body. When the message arrived at the service location, SOAP engine encrypts the envelopes and extracts the message itself for the application.

Web services are used in numerous distributed applications. There are services available all sorts of applications like industrial, commercial and academic. To discover available services, Universal Description Discovery and Integration (UDDI) [75] is defined to advertise services. The clients search through UDDI repositories to explore new services or similar services what they are looking for.

Apache Axis [76] is a useful Web services framework that runs SOAP provider and Web services interfaces. Axis provides tools to build and deploy services. Axis also provides tools for clients to generate stubs from WSDL interfaces.

3.5 Portal frameworks and their components

Web-based portals leverage personalized content and provide access to backend resources. First of all, we need to distinguish portals and portal frameworks. Portals are deployed instances of portal software frameworks that are adopted for specific application or science Grid. For instance, VLab [2] and CIMA [2] portal projects have adopted GridSphere as a framework, but many other Java frameworks exist, such as uPortal, Jetspeed [21], eXo [77], Liferay, and so on. It is desirable to be able to exchange software components between portals and to be able to swap frameworks (e.g. GridSphere for Jetspeed2). Portal frameworks utilize various components known as portlets. Portal frameworks plug these capabilities within deployment descriptors for these portlet Web applications. The framework provides a portlet container and portal specific capabilities such as login, access management, and layout. The Java JSR 168 specification provides the means for building standard portlet frameworks and portlet components.

JSR 168 defines a Java API for portlet development and provides portlet lifecycle management. Portlet Web applications are interchangeably used among portal frameworks such as GridSphere, uPortal, Jetspeed2, and others: one may use the same portlet Web applications without modifying the application code. Portal frameworks do require portlet registration/deployment in a container specific procedure, as well as some container specific modifications to the Web application's configuration file.

JSR 168 portlets generates markups from different applications such as from databases, scripts and Web services. All these markup elements are shown as in HTML by using MVC style Web application languages like JSF, JSP, Spring, and Velocity.

Each portlet application renders an HTML markup fragment for presentation. Portal framework aggregates portlet markups a unified style and display them in one Web page.

3.5.1 Shortcomings of JSR 168

JSR 168 has from our point of view several significant shortcomings. JSR 168 does not standardize description of users, groups and roles. Thus, portal frameworks provide their own style of user and group concepts that may cause mapping problems. We have faced such kinds of problem with the CIMA portal. More importantly, these components and others are not clearly defined as Web services. This creates problems when using the portal as a front end to Grids and Web service oriented science applications. Another major shortcoming of the JSR 168 specification is that it does not provide a sophisticated development environment for creating the portlets themselves out of reusable components.

A parallel effort to JSR 168 is Web Services Remote Portlets (WSRP) [78]. WSRP aims to support remote portlets using Web services standards. Portlets can run in the remote portlet container that is called WSRP producer. WSRP consumer makes requests on that remote portlet and gets response as HTML markup fragment. WSRP consumer could be JSR 168, PHP or Microsoft .NET portlets. Consuming portals only get markup fragments from WSRP producer and present them in the portal page through WSRP Consumer. Unlike JSR 168, WSRP consumer does not render HTML markups.

3.6 Grid account management

Portals in general manage user accounts in implementation specific and usually non-portable ways. But Grid portal user management requires handling standard Grid users with their credentials. Credential mapping with user identities are always issue with Grid portals. There are various solutions are studied by Grid community.

Point-solution attempts to Grid authentication and authorization problem are available such as Grid Account Management Architecture (GAMA) [79] for Grid account management and Portal-Based User Registration Service (PURSe) [80] for Portal user management.

Grid portlets mentioned so far assumes portal users already have Grid accounts available. They only issue proxies to existing Grid accounts. However, there are portal and Grid account management portlets handle Grid account creation and management. GAMA and PURSe are well known Grid and portal account management portlets in open Grid community. Both systems have pros and cons in terms of their designs and underlying technologies. Also portability of servers and user interfaces are general aspects to be considered. The comparison of those account systems are shown on the Table 3.1.

3.7 Grid authorization infrastructures

GridShib [81, 82] is focused on attribute based authorization for Grids. It is combination of Shibboleth [83, 84] and Globus toolkit to enable various administrative domains for federation. Identity of users is carried by X.509 credentials and access to certain resources is determined by identity providers. Service providers always get

permission to access the resources. They facilitated Security Assertion Markup Language (SAML) [85] attribute assertion model within this architecture.

Table 3.1 Comparison of GAMA and PURSe

Criteria	PURSe	GAMA
Server	SimpleCA has to be on PURSe server.	Same with GAMA
Communication	Non-secure server communication	Secure server communication
User interface	Standard JSR 168 portlets	Portlets tightly integrated to Gridsphere authentication module
Access	PURSe server only accessible by local JSP pages	GAMA server provides Web service access

There exist external services providing single functionality like Central Authentication Service CAS [86] for authentication or PERMIS [87] for authorization. Also those available services are not enough to enable a portal entirely service based. If the portal services are available, but there is no portal services coordination available for portals as Web or Grid services.

3.8 Web application frameworks

Web pages can be easily grouped into two categories, *static* and *dynamic*, when

describing their content. Static content does not change; it is always output the same way ignoring any external variables. For example, HTML provides a means to serve static content. HTML describes the way a document should be displayed in a browser but provides no means to change that display. On the other hand, dynamic content can be influenced by external variables often passed through the URL or HTTP Headers. For example, Java Server Pages (JSP) provides a scripting engine that allows accessed pages to output different HTML based on external variables. The client-server model the Web uses only allows two points where dynamic content can be processed on the client and/or the server. Due to limitations on the client side, complex operations are left for server side executing. With standardizations from the World Wide Web Consortium (W3C) [88] client side technologies like JavaScript should work similarly among standard compliant browsers. Of course, many browsers do not conform to the standards, leading to many complications. However, JavaScript provides a common platform for dynamic content on client side and introduce many benefits over standard server-side manipulations. Most Web sites and/or Web applications employ a combination of client-side and server-side technologies to provide rich dynamic content to its viewers.

3.8.1 Java Server Faces (JSF)

JSF is an acronym for Java Server Faces, which is an extension of the well-known and widely used Java Server Pages (JSP) [40, 89]. JSF provides an abstraction on top of JSP to allow rapid development of Web interfaces. Fundamentally, JSF components are collections of reusable JSP codes which are described by XML-like tags.

With standard JSF add-ons it has become easy to bridge JSF applications into portlet applications. Such components include so-called "bridges" currently being developed by Apache [90] that serve the purpose of adapting JSF applications to function as individual portlets within standard containers. These bridges have worked to optimize development time of the VLab portal.

3.8.2 Web Interfaces and JSF

JSF is a dynamic Web application framework that similar to other dynamic Web application templates like Velocity, JSP and Spring. JSF provides dynamic contents using Java Servlet [44] technology. Dynamic contents are generated based on request and response paradigm. Unlike static HTML pages Web application frameworks allow interaction with users. The server pages take requests, process them and respond to the user through a Web interface.

JSF applies MVC software design pattern [31] that decouples data, controller and user interface. The data corresponds to model in JSF terminology is called backing beans. The controller corresponds to JSF servlet that manages user requests. Finally, the view corresponds to Web interfaces of JSF. This architecture separates the data access and user interaction. Also MVC allow reusing backing beans within different applications without rewriting them.

JSF framework provides a component model that supported by user interface (UI) widgets in XML format to implement view pages. In contrast, JSP usually mixes and matches java code and html markups in the view pages. Rather JSF completely separates view pages and beans by using JSF core tags and html tags. Such tags are

<f:form/>, <h:commandButton> etc. These tag instances binds attributes to backend beans that defined on the configuration file of the application called faces-config.xml. Thus, we can say that JSF uses Model2 (is a variation of MVC) pattern better than JSP. Model and View logics completely separated; Model is provided by using Java Beans. JSF Core tags and HTML tags enable developers to inject bean methods by using attributes.

3.8.3 JSF Portlets

JSF portlets are built on top of the standard portlet API (JSR 168) so that allowing them to be deployed within various portal frameworks. Java portlet development generally requires implementing portlet API within the Web application. But there is a tool called JSF portlet bridge that makes portlet programming easier for developers. As a result, developers are not required to construct their portlets using the portlet API directly.

3.8.4 JSF portlet bridge

JSF portlet bridge injects Java beans with portlet API so that there will be no portlet programming phase. JSF portal bridge [90] provides a servlet to deploy JSF applications as portlet. This bridge consumes portlet API so that deploying any stand-alone JSF application as portlet. JSF bridge supports MyFaces [91] reference implementation of JSF which may not support different reference implementations such Sun reference implementation.

Chapter 4

Applications

4.1 Introduction

In this chapter, we discuss several different science gateway initiatives with scenarios to build a Grid portal. A science gateway building process starts with following demands including portability and accessibility. As matter of fact, this transition process is burdening in the following aspects. First, the science gateways must design efficient interfaces to attract users from different backgrounds like core scientists to elementary school students. Second, the gateways need to advertise their backend services and to make service interfaces available for exploration. When they are exposing all the services and visual interfaces, they also need to keep those services and its data secure. In this case, science communities need to enforce policies for user authentication and access control rights of the users. Our Grid tag libraries present solutions for these issues.

The “Grid tag libraries” has been used in real applications like The Virtual Laboratory for Earth and Planetary Materials (VLab) [15], Common Instrument Middleware Architecture (CIMA) [92], and Indiana University Big Red portal [93]. We summarize the efforts to build science gateways on the following use cases. We also evaluate and discuss repeated problems and their solutions sticking with these practices.

4.2 VLAB: Virtual Laboratory for Earth and Planetary Materials Portal

The VLab is a National Science Foundation-funded interdisciplinary research collaboration whose primary objective is to investigate planetary materials at extreme conditions based on computational techniques to better understand the processes that create earth-like and other planetary objects. Such calculations typically involve hundreds or thousands of computer runs. These runs occur in stages, with complex interactions. They are often managed by several researchers. To address challenges in collaborative and distributed computing, VLab brings together a team that includes researchers in computational material science, geophysics, scientific visualization, Grid computing, and information technology. Additional information on VLab is available from [94].

Some of the many problems that VLab must address include the ability to create input files through portals, submit jobs, store and retrieve the job input and output data on demand, analyze and visualize the data, and store the data. These tasks must be possible in a distributed environment and the flow of information must be accessible to multiple collaborating researchers, although they might not be co-located. An

additional constraint on our system is that it must be robust, i.e., fault tolerant. When working in a complex multi-user environment, it is inevitable that some components will fail. However, these failures should not affect the work of an individual researcher. Thus, we have chosen to connect the users of the systems (referred to as clients) and the various tasks requested by the users (storage, visualization, analysis, job submission, etc.) as services using NaradaBrokering [95], a middleware system that builds in many of the required features.

In its initial phase, VLab follows a well-established pattern for building Grids: application codes on remote machines are accessed securely through Grid services through a browser portal. This follows the common three-tiered architecture. A user interacts with a portal server through a Web browser. The portal server in turn connects to remote Grid services that manage resources on a backend system that provides the computing power for running the codes. The longer term research goal however is to go beyond these traditional approaches. The distinguishing feature of our research is the use of the publish/subscribe paradigm, which completely decouples the clients from the services. Users have no knowledge of the resources allocated to their requests, although they will have the capability to monitor task progress.

The VLab science gateway is based around the JSR 168 portlet model, and the initial set of VLab portlets are described in detail in [15]. We began by developing Grid portlets using the OGCE [27] software. In this model, each portlet application was responsible for an individual task. For example, one portlet is used for submitting jobs to PWSCF resources, another one is for GridFtp file operation, and a third is to get Grid credentials from MyProxy [59] repository. This traditional approach is not useful for

the case of VLab. Instead, we need to collect all capabilities within a single portlet application and to handle complicated PWSCF-based job executions and file transfers in a sequence; that is, we must define dependencies between atomic job tasks. Consequently, we have determined that we can represent job dependencies using DAGs.

To implement these graphs, we chose the Java CoG abstraction [62] interfaces for DAG executions in Grid. These provide a convenient Java programming interface that can be easily integrated into Java portlets. However, we identified the need to provide a higher-level development environment that encapsulates common tasks needed to assemble a DAG in a portlet. Our approach is to design XML-based tag libraries for expressing DAGs and to embed them in the Web pages. For that reason, we found the JSF application framework to be appropriate. JSF is a component-based Web framework that can be extended to add new components, such as our DAG XML tags. As we discuss in this paper, we have implemented and have used initial prototype of Grid tags within JSF. In case of VLab portal, we have learned that we need a more comprehensive workflow engine to support loops, parallel jobs and conditional branches.

Figure 4.1 shows the flow control of end user and portal resource interactions. End users login to VLab portal securely and then store their Grid credentials to access Grid services. While these processes are initiated by end users, all of the events are persistently stored on Metadata server.

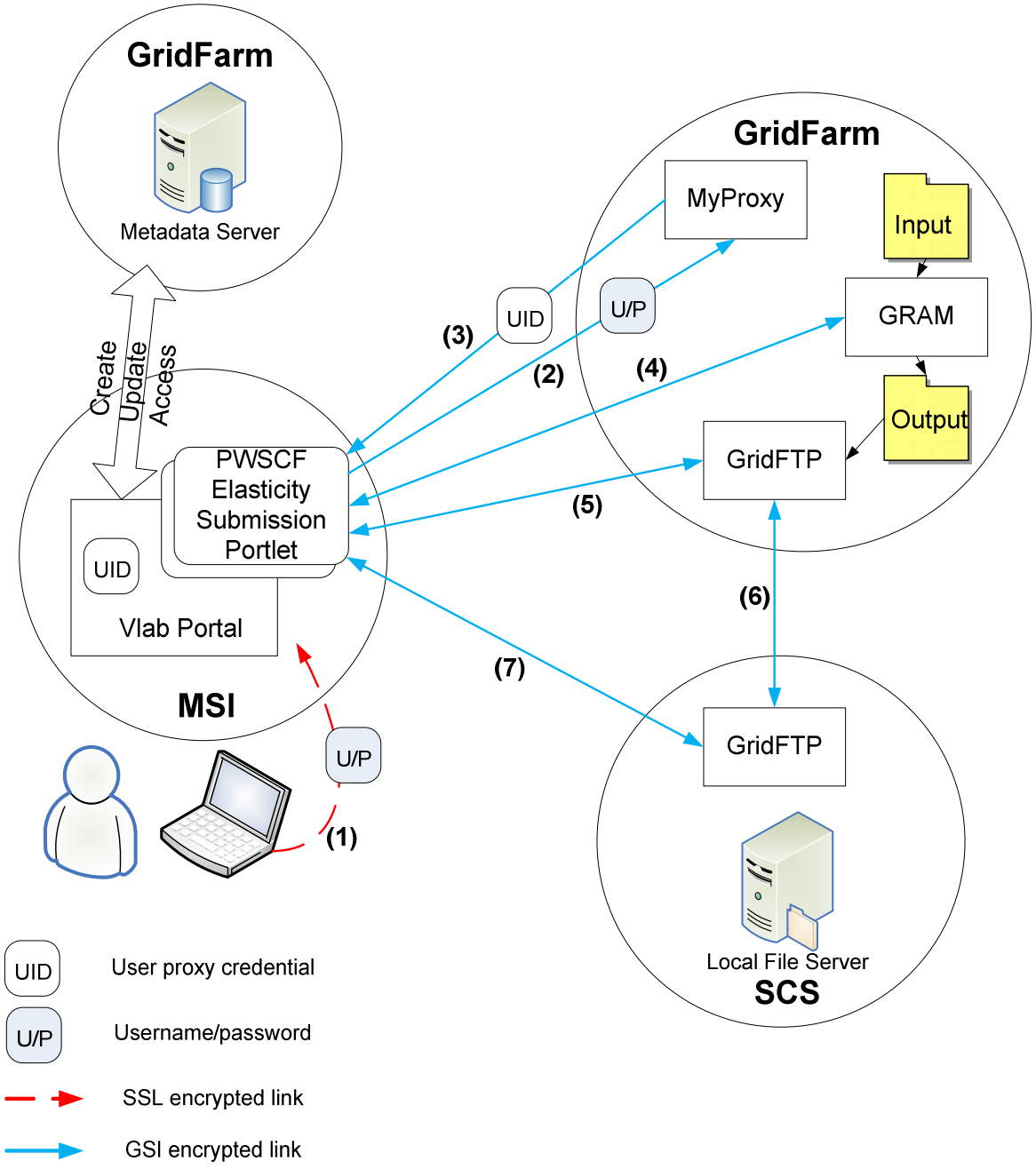


Figure 4.1 VLab portal serves to the end users by utilizing remote resources.

4.3 QuakeSim Portal

The QuakeSim portal is a problem solving environment to develop a solid Earth science framework for modeling and understanding earthquake and tectonic processes. The multi-scale nature of earthquakes requires integrating many data types and models to fully simulate and understand the earthquake process. The QuakeSim gateway includes portlets and services for accessing real time and archival data. The data sources (Global Positioning System data, earthquake fault models) can be integrated with computational applications for event detection and seismic deformation calculations. These latter include finite element methods (GeoFEST [50]) that can be computationally intensive and best run on parallelized platforms. In this study, we aim to utilize TeraGrid [4]resources to solve computational problems of QuakeSim project [96].

The QuakeSim portal has served the community since 2002 and is currently undergoing several major revisions. In terms of using the portal frameworks, it initially used the Jetspeed framework. It has subsequently been updated to use the standard compliant, second generation portal framework GridSphere, which is compatible with JSR 168 portlet specification. In its current form, the QuakeSim portal uses portlets developed with the JSF Web application development framework. JSF is component and tag based, and allows extensions. QuakeSim portlets are typically designed as clients to remote Web services that constitute the QuakeSim middleware. These portlets aggregate user information and data through JSF interfaces and invoke the actions matching the Web services. QuakeSim services use Apache Ant [97] based services to

manage jobs and to build multiple steps of jobs that depend each other (i.e., to handle simple workflows).

QuakeSim's computational services are suitable for many of its applications, but it must be extended to support more extensive computations for parallel applications. Grid services of TeraGrid (e.g., Globus, Condor) provide this capability, so we need a way to modify existing JSF-based portlets to work with these services. To simplify this transition and to provide test cases for our GTLAB framework, we decided to combine the two efforts.

We describe the application of GTLAB to QuakeSim as a case study. In this case backend applications run on TeraGrid and we access these legacy applications with GRAM, GridFtp, MyProxy services. We will show the integration and implementation of Disloc and Simplex portlets with GTLAB. We also evaluate development time and runtime performance results based on the tests that we conducted on different geographical locations.

4.3.1 QuakeSim Gateway Architecture

QuakeSim portal architecture was previously designed for Web services invocations in the middleware. These portlets aggregate user information and data through JSF interfaces and the actions invoke matching Web services methods. QuakeSim services utilize Apache Ant-based services for managing executable invocations, interacting with the operating system, and controlling simple workflows. Ant build scripts serve as templates for defining the operations of a particular application service. These server-side Ant build scripts can be converted into portlet-side GTLAB XML tags.

Instead of altering QuakeSim service interfaces synchronizing with Grid services, we remove the Web services layer. Therefore we use Grid services to invoke remote applications, to make file transfers and to provide security. However, we also need to allow implementing workflows within the scope of QuakeSim scripts. In other words, we are able to translate Ant scripts to series of Grid service invocations that are represented as graphs. This new approach has advantages to the previous architecture. First, there is no need to alter service interfaces when the Ant scripts change. Second, in the previous system, service clients cannot access the service layer to change scripts. Therefore the clients have to request required changes that involve additional management efforts as well.

Enabling QuakeSim portlets such as the Disloc interface to work with GTLAB requires a few changes on the portlet pages. First of all we preserve all JSF pages that collect information from users such as input forms and parameters. Next, we replace the JSF form page that invokes QuakeSim Web services with Grid tags. Therefore the embedded Grid tags that are invisible to the end users will call Grid services by using Grid beans. As a result of these simple changes we gain from development time.

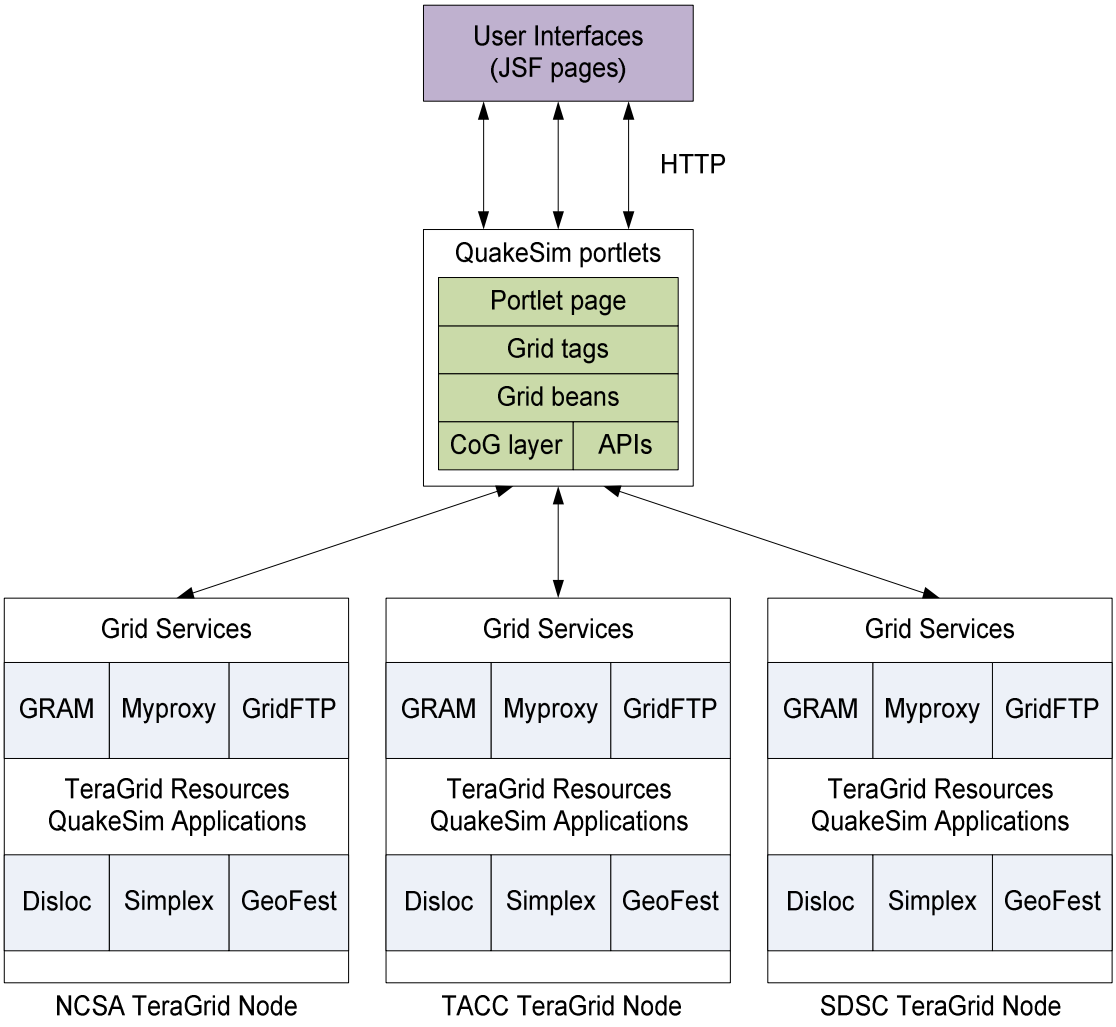


Figure 4.2 QuakeSim portal architecture with Grid services invocations of TeraGrid nodes.

As shown in the Figure 4.2, QuakeSim architecture utilizes GTLAB to access TeraGrid nodes. We customize portlet pages to connect which TeraGrid nodes beforehand. Therefore end users would not worry about TeraGrid availability. It is also possible to involve the end users in the node selection stage. In which case, users have to be knowledgeable about the nodes. In our design, users get their MyProxy credentials before using any other Grid service. Then they can use one of the services

such as GRAM for invoking applications or GridFtp to transfer files from one server to another.

4.3.2 QuakeSim Portlets

QuakeSim portal previously built and production with different technologies. In this case study, we rebuild QuakeSim portal with Grid portlets by integrating GTLAB. Therefore, we choose GridSphere portal framework to build QuakeSim portal. In the building process we provide portlets for QuakeSim applications including Disloc and Simplex.

```
<o:multitask id="multi" persistent="true"
 taskname="#{resource.taskname}">
  <o:myproxy id="mypr" hostname="gf1" lifetime="2"
   password="manacar" port="7512" username="manacar"/>
  <o:jobsubmit id="make" arguments="/home/manacar/disloc-
   work" executable="/bin/mkdir"
   hostname="gf1.ucs.indiana.edu" provider="GT2"
   stdout="/home/manacar/tmp/out-make"/>
  <o:jobsubmit id="disloc"
   arguments="/home/gateway/GEMCodes/Disloc/input.txt
   /home/manacar/disloc-work/disloc.out"
   executable="/home/gateway/GEMCodes/Disloc/disloc"
   hostname="gf1.ucs.indiana.edu" provider="GT2"
   stdout="/home/manacar/disloc-work/out-disloc"/>
  <o:dependency id="dep" dependsOn="make" task="disloc"/>
</o:multitask>
```

Figure 4.3 Disloc portlet page contains multi-staged jobs with DAG representation

4.3.2.1 Disloc Portlets

Disloc models multiple dipping dislocations (faults) in an elastic half-space. In the view of portlet development, Disloc is an application that we need to run by providing parameters and input files. Disloc run on TeraGrid and the users either can by using

command-line tools or shell scripts. But portal users can only access by using Grid services to access TeraGrid in a secure way.

GTLAB provides a client layer on top of Grid services that is bridge to the portal users. In other words, the portal users can access Disloc transparently through portal user interfaces (i.e., Web forms). Not only using an application is possible, but also a DAG could run the multiple steps of Disloc such as in Figure 4.3 making a directory on the file system to save output file, then running Disloc application that depends on the first task.

4.3.2.2 Simplex Portlets

Simplex is an inversion code based on Disloc. Similar to Disloc, Simplex applications are run by DAGs that describe order of the tasks and their dependencies. Then JSF pages collect parameters and information about the task to submit it to TeraGrid.

4.4 CIMA: Common Instrument Middleware Architecture Portal[1]

One of the key issues in developing shared instrument systems is how to create an open and flexible approach to user interfaces for access to instruments and the data streams coming from them. In related work [98] we have described how portals can be used to organize access to instruments through the Common Instrument Middleware

[1] This section is taken from [92]

Architecture (CIMA) [99, 100] and how individual portlets can provide specialized, role and task specific functionality as users, technicians and system administrators interact in the generation, analysis and management of data from shared instrument resources. In this paper we will focus on the approach taken to develop portlets for managing crystallographic data in a group of cooperating laboratories.

X-ray crystallography is an analytic technique to help scientists understand and determine the precise molecular structure of a crystalline substance. However the instruments (called X-ray diffractometers [101]) required to perform these types of studies are quite expensive and require a highly trained operator. The relevant data from a crystallography experiment contains a series of diffraction images usually captured by a CCD detector, and a number of environmental variables including crystal temperature, crystal alignment image, CCD cooling status, and the temperature and relative humidity of the lab.

In some cases, due to the nature of some crystalline materials such as proteins or microcrystalline compounds, the successful structure determination of these compounds require the use of high brilliance radiation sources available at national synchrotron facilities. Gaining access to beamlines at these national synchrotron facilities to collect data is not straight forward. Travel to these remote facilities is costly and time consuming, and once there, the facilities must be used in an intensive manner. By developing methodologies to remotely monitor and access instruments and their data we can provide the remote users with a "same as being there" experience with additional flexibility in scheduling around problem samples and equipment failures. Additionally on-site users and technicians can share data coming from the beamline's

instruments with remotely located colleagues to discuss the quality of a diffraction pattern. This remote consultation capability can facilitate decision making such as continuing with a questionable sample or abandoning it and starting a new one. Effective shared access to instruments ensures a more efficient use of the beam time, potentially improving throughput of the beamline as a whole. This paper will focus on the implementation details of the CIMA crystallography portal and the mapping of end-user functional requirements to portlets.

Grid tags are also being developed for use in CIMA crystallography portal. CIMA provides access to X-Ray crystallography, instrument and sensor data. Sample data includes CCD images of crystals as well as laboratory conditions such as temperature and humidity. The CCD images may also be post-processed. One of the post-processing applications used is SAINT [102], used to integrate CCD image frames, sort reflection lists, scale, filter, and merge reflections. In this case, crystallographers launch a SAINT application using *multitasks* to initiate an image analysis. This process results in image files that are being downloaded to a portal server and are made available for users.

4.4.1 Requirements

For the current work, a subset of requirements relating to user and administrative interaction with data was chosen. These include the following:

- Remote users and in-lab crystallographers must be able to monitor an experiment in progress, including viewing current and previously collected CCD frames and associated relevant environmental and technical parameters;

- All raw data is owned by the lab which performed the experiment and collected the data. In addition to the lab, represented by one or more lab administrators, individual users can view (but not modify or delete) their samples;
- Lab administrators must be able to control sample ownership and visibility;
- Because the notion of when an experiment ends is not clearly defined (*e.g.* experiments may be truncated after the fact or additional frames may be gathered based on evaluations made during a run), lab administrators should be able to set the end time of an experiment;
- Lab administrators must be able to add and remove users to an access control list for a sample;
- Users must be able to view their samples, including all files and sensor readings related to the experiment;
- Some sample data may be provided to the general public for educational or public science awareness purposes;
- Users must be able to view the current status of the lab as a whole;
- Individual functions that are of general utility should be implemented in a reusable, pluggable, standards-based manner as portlets that can be added or removed by administrators or end-users as appropriate;
- The portlets must interact with a lab's data manager software via Web services calls;
- Users and groups will be managed by the portals container and access to all functions of the portal will be provided by a single sign-on through the portal.

A prototype implementation of the crystallography portal was completed using Jetspeed1 and CGI scripts. Although in the right direction, this implementation did not meet our modularization requirement and so with the ramp-up of the NSF middleware project and the availability of support from the OGCE group, we migrated to GridSphere and JSF-based portlet clients to CIMA services as a fully JSR 168 compliant portal container. This assures a degree of survivability and lateral flexibility to move the science process specific functionality to other containers if the need arises.

The requirements outlined above led us to develop the following portlets:

- A lab overview portlet that provides the current status of a facility and its instruments;
- An administrative *Admin* portlet to support management of sample ownership and other parameters related to individual experiments;
- A *PublicSample* portlet that provides sample data to all portal users and the general public;
- A *UserSample* portlet that shows a logged-in user their samples and other samples on whose access control lists they appear.

PublicSample and *UserSample* portlets also allow users to scan through all data objects in an experiment. Per design choice some functions of the portlets are made available as pop-up windows. These portlets provide all of the functionality listed in the requirements above. Extensions to the portal's basic group authorization mechanism provide an access control list associated with each data collection. Scientists can view and modify sample data from their X-ray diffraction crystallography experiments based

on their roles in this project and can add users to the access control lists of their data sets. Nothing more than a web browser is needed to interact with the system.

4.5 Big Red Portal

Big Red is a major new TeraGrid resource and one of the most powerful computers in the world. As with all TeraGrid resources, it runs the Coordinated TeraGrid Software and Services, which includes Globus services. One of Big Red's initial applications is Mutiple EM for Motif Elicitation (MEME) [103]. MEME is used to discover common motifs in groups of DNA or protein sequences. Due to its computational complexity, MEME should be executed in a rich resource environment such as Big Red. However, to execute MEME on Big Red, a user must not only be familiar with the application itself: he or she must also understand various network tools such as FTP for uploading and downloading input and output files, and he or she must understand Big Red's LoadLeveler and MOAB-based scheduling and queuing system in order to submit, monitor and control jobs. This kind of inconvenience can be easily overcome by making a specific portlet that allows a user to execute the MEME application by using a science portal based on the OGCE [27]. In addition to MEME execution, we can add file management and job control functionality into the portlet by using Java CoG kit to utilize Big Red's Grid infrastructure.

Portlets provide a common component for building portals out of reusable parts. For example, as mentioned previously, the OGCE portal has portlets for job submission, credential management, and file management that can be plugged into any standard compliant container. Often, however, as in the case of the MEME portlet described

above, portlets are not quite fine-grained enough components. We would like to build portlets that combine several Grid operations in the same portlet. Our work on GTLAB provides a set of JSF tag libraries and backing JavaBeans (called Grid beans) that attempt to solve this problem. A full discussion of JSF is out of scope here, but briefly, JSF generates HTML from a set of XML tags. HTML form actions are associated with so-called backing JavaBeans, which in turn may act as Web service clients or connect to databases. Developers can extend these libraries to provide their own XML tags.

The goal of GTLAB is to simplify the process of Grid portlet development by encapsulating common Grid operations as XML tags that can be embedded in portlet pages, enabling rapid development. GTLAB capabilities include credential management, remote file operations, remote job executions, and file transfers.

The JSF Web application framework provides us with extensible component architecture. Each XML tag is associated with a backing Grid bean that implements the actual Grid clients, which we build with the Java CoG kit. We use JSF's built-in functionality to pass attribute values from the XML tags to the backing beans. Grid beans are associated with Grid tags and their action methods are fired by our 'submit' tag. Tracking the jobs and monitoring is also part of the GTLAB framework.

4.5.1 Integrating GTLAB with Big Red Portlets

Typically a Grid portlet stages various related tasks in response to a user-generated event. These are usually the nodes of a DAG, which our Grid tags are designed to support. The DAG, or composite task, is called *multitask* in GTLAB. Currently, multitasks only allow dependent task units and prevent parallel tasks and cycles.

After building the sub-tasks, multitask and their dependencies, GTLAB then registers multitasks in the browser session. In addition, it registers their handler information within the session to track their lifecycle. All of the objects are stored in hash tables with a unique key. The job handler information can be stored persistently to a backend storage system (i.e., a database) by setting *persistent* attribute of multitask.

The following scenario shows the building of multitask for MEME with dependent multi-staged tasks. Assume a developer has been assigned the job of creating a portlet to do the following basic tasks. First, Task A makes a working directory on Big Red. Then, Task B transfers an input file from a remote host to the newly created directory. Finally, Task C is responsible for submitting a command script on Big Red using the input file. The following sections explain the scenario in detail through the use of Grid tags. After the portlet is finished and deployed, users will then submit and monitor jobs using the developer's portlet. Users will not see the tag libraries and will interact with standard HTML pages that get generated when the portlet is rendered.

4.6 Summary

We have summarized the Grid portal efforts of scientific communities including VLab, CIMA and Big Red. These uses cases have shown that similar issues and solutions are applicable to different community requirements. Our generic Grid tags have been applied to these portlets by adjusting configuration and customization principles for each case. Therefore we have saved development time for each portal.

Chapter 5

Architecture of Grid Tag Libraries

5.1 Introduction

We aim to provide a set of Grid tags in JSF that can be used to build Grid portlets. We have described our tag libraries, called Grid Tags Libraries and Beans (GTLAB), in [48] that provide common Grid capabilities such as proxy credential management, job submission, file operation, and workflow by means of multi-staged tasks. Grid tags are associated with Grid beans to access Grid services. Grid bean methods are bound to tags with attributes. These can then be used to simplify the building of new Grid portlets.

Figure 5.1 shows the general picture of science gateways and where we locate the GTLAB within this big picture. In this architecture, Grid portlets are built using JSF

Grid tags. Grid tags and beans use local services on the portal server such as bean repository, listener repository and MyProxy repository.

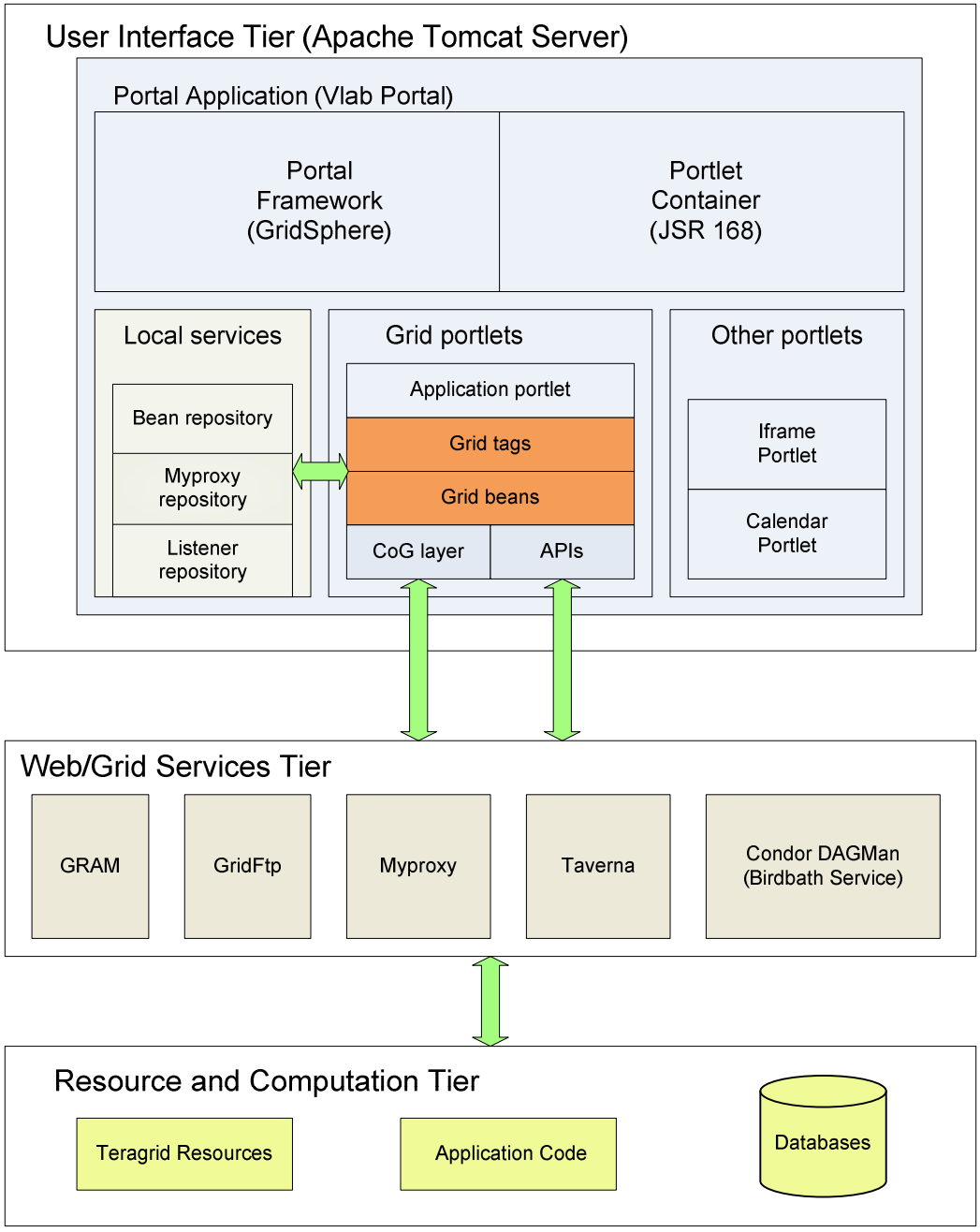


Figure 5.1 Big picture of Grid portlets using GTLAB libraries and JSF framework

5.2 Design

Grid portlet programming is a burden some process for application developers. GTLAB provides several important features for application developers. First, it provides modular components (tags and beans) to construct science gateway portlet pages. Second, it represents Grid service clients using abstract XML tags. Therefore, portal developers do not need to understand underlying details of Grid services. Finally, it provides a component model for developing Grid portlets out of reusable parts.

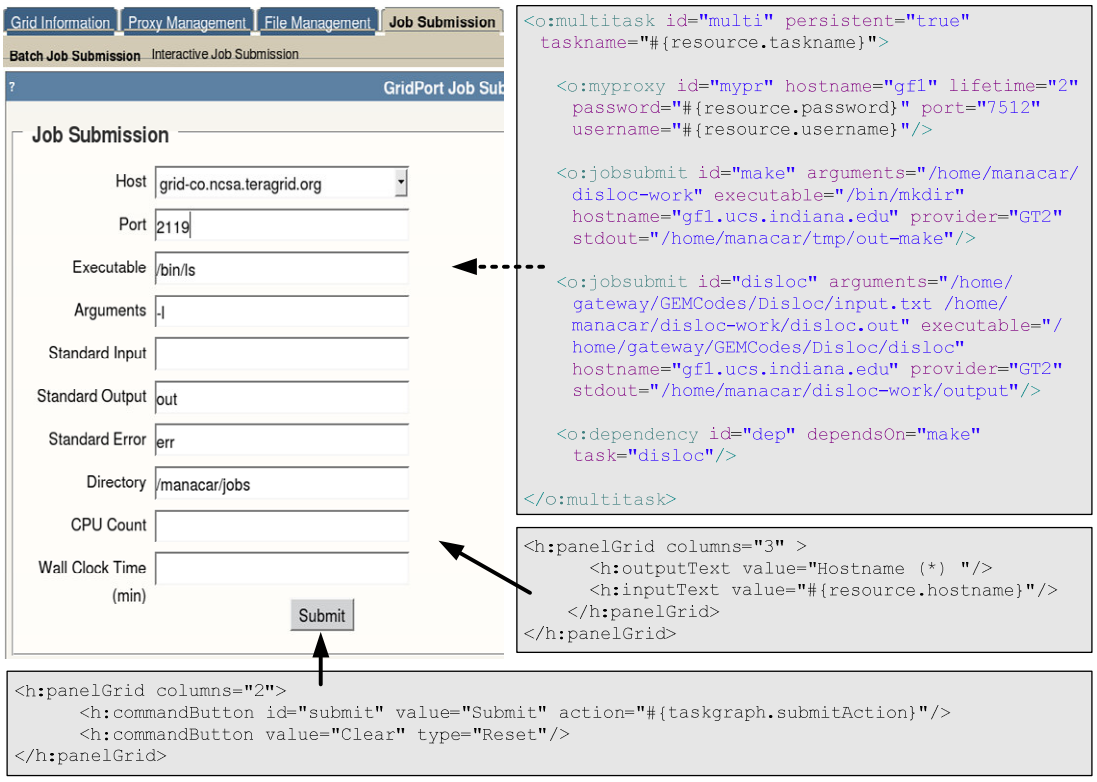


Figure 5.2 Grid tags are embedded into JSF view pages with visual HTML tags

The design of GTLAB requires three essential parts: 1) Grid tags and 2) Grid beans 3) Session management. These components are explained in greater detail in the coming sections.

Figure 5.2 shows a JSF view page that builds a Grid job submission portlet page by using Grid tags. The portlet page contains a Web form, input and output text fields and submit button. JSF view pages are built and rendered by JSF container to construct HTML pages. So a JSF page is different than what you see in the browser's resource page. Basically XML based tags build the view page. We're using core, html and Grid tags to build a Grid portlet page.

The Grid tag components in Figure 5.2 also bind Grid bean features to call appropriate Grid service clients. Grid beans are using underlying Globus, Condor and Taverna [104] services by abstracting their client APIs to work with Grid beans.

Grid beans and listener information are managed by session manager. Session manager basically keeps track of every submitted Web forms in the portlet page and identifies them with unique ids within a user (e.g. browser) session.

Application developers only plug these tags into their portlet applications and add required libraries to their Web applications to enable GTLAB. There are only simple settings to customize GTLAB within the portlet application.

We have used the strategy of returning immediate results to the user such as passing the control to the next page since Grid operations can take a long time to complete. Thus, a user submits the job in one page and is not required to wait until the job finishes. Instead, users are able to monitor their jobs in another page. To maintain this scenario, either we need to keep callbacks for each job or to store listeners for each job

in the servlet HttpSession object. We have therefore used Component Builder Bean (CBB) that take care of each request in the session. Then we store bean instances and their listeners into tables (Hashmap) among the session with *taskname* key. The *taskname* key is created by putting the user-defined taskname (collected from Web form input) and the timestamp together to provide a reasonable ID.

Figure 5.11 illustrates the user interaction with the Grid beans and tags are illustrated. When user hits a submit button, CBB takes control. First, CBB constructs a multitask with the components defined by the Grid tags. CBB is also responsible for submitting the multitask and for managing the lifecycle with associated listeners. After the submission is completed, control is passed to MonitorBean shown on the right. MonitorBean interacts with the session to retrieve the information of submitted tasks.

5.3 Grid tags

Grid services are interfaced by Java CoG abstractions. These programming interfaces have capabilities to generate proxy certificates, submit jobs, transfer files and make file operations. They also provide composite task submissions and their handling.

JSF technology helps to build user interfaces based on an object-oriented component approach. JSF tags are built from Java classes that can be extended using JSF component model. New components derive from JSF base component classes. Each component should define its attributes, which can bind values, methods or actions. A full discussion explaining how to extend JSF components is beyond the scope of this paper. We recommend [105] for a tutorial on this subject.

The main goal is to make Grid portlet development easier by encapsulating standard Grid operations with JSF tags. These tags can be assembled to create composite tasks. In traditional Web frameworks such as Velocity and JSP backing bean objects and HTML tags are mixed within the server pages. Instead JSF eliminates this intervention by proposing JSF tags that separate backing bean and server pages.

5.4 Grid tag schemas

JSF Grid tags are defined in the Java based tag libraries and supported by JSF backing beans. JSTL tags are in the XML format. Tags and their attributes are predefined within *.tld* files. We can define optional attributes besides required ones. Also we can define tags in nested structure as in XML. Figure 5.3 shows the XML schema of <o:multitask> that may <o:myproxy>, <o:filetransfer>, <o:jobsubmit>.

Table 5.1 shows the attributes of <o:multitask> tag with denoting which attributes are required. Similarly, Table 5.2 shows the attributes of <o:myproxy> tag. Further information about tag tables can be found at Appendix B.

The Grid tag schema is extensible for new tags and attributes. For example, we have added <o:scufl> and <o:condor> tags later to support Taverna workflows and Condor Grid services. Similarly, we have added persistency feature to available services by adding *persistent* attribute to all Grid tags.

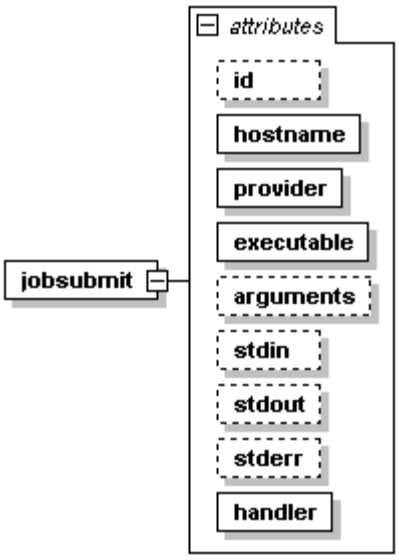


Figure 5.3 Grid tags schema for job submission to GRAM server

5.5 Use case example

Typically a Grid portlet must do several related tasks in response to a user-generated event. These may be thought of as simple workflows. These workflows can be considered the nodes of a DAG, which are Grid tags are designed to support. The DAG, or composite task, is called *multitask* in our approach. Multitasks only allow dependent task units and prevent parallel tasks. Figure 5.4 shows a multitask with tasks and their dependencies. In this example, Task A makes a directory. Task B transfers an input file form a remote host to newly created directory, and Task C is responsible for submitting a job on the remote computer. When Task C completes, Task D transfers output file to another location. The following explains the scenario in detail through the use of Grid tags.

This example demonstrates a composite Grid task with Grid tags. The JSF snippet below (Figure 5.6) shows how a portlet developer would create a custom Grid portlet. First, a myproxy tag generates a proxy credential from `gf1.ucs.indiana.edu` MyProxy server. Second, using this credential, it makes a directory on the TeraGrid resource `cobalt.ncsa.teragrid.org`. Third, it transfers an input file called *input_file* from `gf1.ucs.indiana.edu` to `cobalt.ncsa.teragrid.org`. Fourth, it then executes a script called *execute*. When the execution is completed outputs are written to the file named *result*. If an error occurs it is also written to the file named *error*. Finally, result file is transferred back to `gf1.ucs.indiana.edu`.

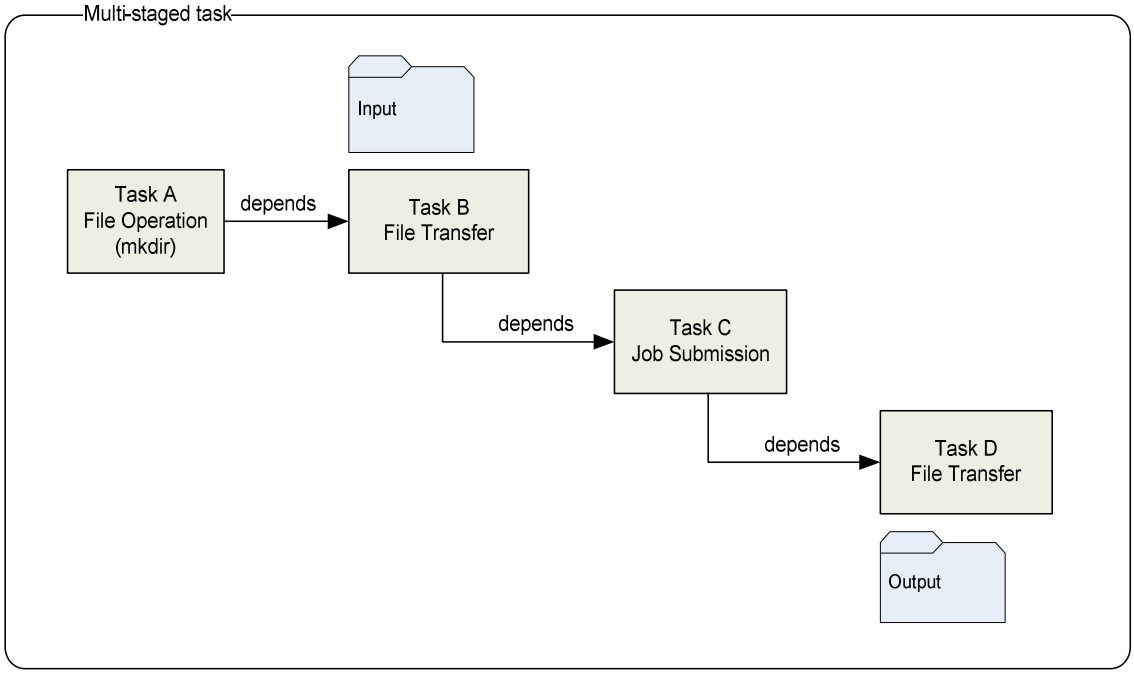


Figure 5.4 A typical multistage Grid job involves four sub-tasks: moving an input file to a particular execution host, submitting the job, and moving the output to a storage host.

The `<%@taglib uri="http://www.ogce.org/gsf/task" prefix="o"%>` tag is used at the top of the page to define the custom tags called with the *"o"* namespace. Application developers must define Grid operations in a Web form. The <o:submit> tag is a submitting button for the composite task that is bound to a JSF action method. The <o:multitask> defines composite task and <o:dependency> defines their dependencies. The tasks <o:myproxy>, <o:fileoperation>, <o:filetransfer> and <o:jobsubmit> are unit tasks for this composition. The dependency tags indicate that taskA must complete successfully before taskB will run, taskB must complete successfully before taskC can be run and taskC must complete successfully before taskD could run. Complete XML schema specifications of Grid tags can be found at [106]. Each Grid tag is associated with a UI component and tag class that is explained in great detail in the next section.

5.6 Design and Implementation of Grid tags

Grid tag libraries are built using JSF custom component development techniques. A standard JSF tag requires at least two classes to be implemented: the ComponentTag and UIComponent classes must be extended. Tag names and attributes have to be defined in a *tld* file and this file is added to *web.xml*. Component names and classes are defined in *faces-config.xml*. A full explanation of JSF custom tag development is available from [105].

Custom component classes extend the UIComponentBase class and are normally associated with HTML or other rendered widgets (input fields, buttons, etc.) in the user interface. We have implemented several custom UI components, including UISubmit

and UIMultitask, as discussed here. Components can access a map (specifically, a java.util.Hashmap) of attributes and child components. If the component is visual like UISubmit (which we associate with the HTML <submit> button), it also implements encoding and decoding methods to process HTML markup. If the component is non-visual (i.e. does not need to be converted into HTML), it is associated with a null renderer. UIMultitask class is a non-visual component. In addition, the JSF ComponentTag class extension has to implement release(), setProperties(), getComponentType(), and getRendererType() methods. The setProperties() method binds attribute values and methods to the associated UIComponent.

In JSF, the tags and attributes are used to render displays and communicate attribute values (see Figure 5.6). We encapsulate the actual logic of the page (associated with user button clicks) in several beans that are called by the UISubmit's action method. Besides tag and component classes, there are core beans as following:

- ResourceBean: A general bean to collect property values used in JSF form pages. By default it loads property values from a *resources.properties* file.

5.7 Handler tag manages monitoring of the jobs

Grid tags launch Grid operations. Keeping track of lifecycles and archiving are also important aspects of Grid portlets. Thus, we define a <o:handler/> tag in Figure 5.5 that provides capabilities allowing users to manage lifecycles manually such as canceling, suspending, and resuming the jobs. The <o:handler> tag is visual and it is rendered as HTML button. The session tables only persist until the servlet session expires or terminates. So we need to have mechanism to persistently preserve them in a permanent

storage. The *persistent* attribute of the multitask tag switches archiving on and off (see Figure 5.6). A context server [15] provides archival facilities that store bean values and the status in a structured way.

```
<f:view>
  <h:form id="first" >
    <h:dataTable value="#{tasklist.tasks}" var="task">
      <h:column>
        <f:facet name="header">
          <h:outputText value="Handler" />
        </f:facet>
        <o:handler id=”delete” action="#{monitor.cancel}" >
            <f:param id="task" name="taskname" value="#{task}"/>
        </o:handler>
      </h:column>
    </h:dataTable>
  </h:form>
</f:view>
```

Figure 5.5 The handler tag is used with <h:dataTable> to create a table of tasks and enable cancellation actions.

Figure 5.11 illustrates the user interaction with the Grid beans and tags are illustrated. When user hits a submit button, CBB takes control. CBB first constructs a multitask with the components defined by the Grid tags. CBB also submits the multitask and manages its lifecycle with associated listeners. After the submission is completed, control is passed to MonitorBean shown on the right. MonitorBean interacts with the session to retrieve the information of submitted tasks.

Table 5.1 Attributes of multitask tag

Attribute name	**Required**	**Description**
id	yes	String: component id
taskname	yes	String: task name is for multitask
persistent	no	String: stores task information
handler	no	String: defines bean method to submit this component

Table 5.2 Attributes of myproxy tag

Attribute name	**Required**	**Description**
id	yes	String: component id
hostname	yes	String: myproxy server name
port	yes	String: myproxy port number (default is 7512)
lifetime	yes	String: myproxy lifetime (default is 2 hours)
Username	yes	String: user name for stored credential
password	yes	String: password for stored credential
handler	no	String: defines bean method to submit this component

5.8 Grid Beans

Grid tags and beans work together to perform Grid tasks. Grid tags provide the JSF components for Grid applications, while Grid beans provide the business logic of Grid applications. We have implemented Grid beans in a generic and standard way to support underlying Grid technologies. We have also attempted to design our tag libraries to support other Grid bean implementations. The Grid beans are generic tasks that may be extended using other toolkits besides Globus. For example, the JobSubmitBean for job submission uses Globus resources in our implementation. Developers can create their own beans with other toolkits. For example, Condor can be used for job submissions rather than Globus. However, this requires that Grid bean method names should be standardized and required bean methods has to be provided. For example, actions methods should be called *submit* in all beans. Parameter names should also be consistent throughout the beans e.g., hostname, provider, username and executable etc.

```
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<%@taglib uri="http://www.ogce.org/gsf/task" prefix="o"%>
<f:view>
   <h:form id=”myform” >
      .......
      .......
      <o:submit id=”test” action=”next_page” />
         <o:multitask id=”mytask” taskname=”test” persistent=”true” >
            <o:myproxy id=”proxy” hostname=”gf1.ucs.indiana.edu” port=”7512”
                       lifetime=”2” username=”manacar” password=”******” />

            <o:fileoperation id=”taskA” command=”mkdir”
                             hostname=”cobalt.ncsa.teragrid.org”
                             path=”/home/manacar/tmp/” />

            <o:filetransfer                                          id=”taskB”
from=”gridftp://gf1.ucs.indiana.edu:2811/home/manacar/input_file”
to=”gridftp://cobalt.ncsa.teragrid.org:2811/home/manacar/tmp/input_file” />

            <o:jobsubmit id=”taskC” hostname=”cobalt.ncsa.teragrid.org”
                         provider=”GT4” executable=”/bin/execute”
                         stdin=”tmp/input_file” stdout=”tmp/result”
                         stderr=”tmp/error” />

            <o:filetransfer                                          id=”taskD”
from=”gridftp://cobalt.ncsa.teragrid.org:2811/home/manacar/tmp/result”
to=” gridftp://gf1.ucs.indiana.edu:2811/home/manacar/result” />

            <o:dependency id=”dep1” task=”taskB” dependsOn=”taskA” />
            <o:dependency id=”dep2” task=”taskC” dependsOn=”taskB” />
```

```
            <o:dependency id="dep2" task="taskD" dependsOn="taskC" />


         </o:multitask>
      </o:submit>


   </h:form>
</f:view>
```

Figure 5.6 Grid tag libraries are used to build a sample Grid portlet page.

- MyproxyBean: This bean generates user proxies and stores the Grid credential in the session.
- JobSubmitBean: Executes GRAM job submissions.
- FileOperationBean: Performs common file and directory operations like *rm, mkdir, put, get*
- FileTransferBean: Transfer files among GridFtp servers
- MultitaskBean: Creates composite tasks and execute them.

Note that these are independent of the JSF framework. The Grid tag libraries shown in Figure 5.6 are built from these, as we describe in the next section.

5.9 Session Management

Portal systems work on Web browsers and maintain the browser features using session cookies. Cookies provide communication between the user and the tomcat server. The HTTP protocol only allows one-way asynchronous communication. Cookies get a unique user id for each user session and keep these IDs in the server side to maintain lifecycle of the user interactions within request/response systems.

There are several cases we need to handle with portal users including: 1) Many users from many different computers. 2) One user logs in from different computers 3) One user opens up many browser pages from the same computer. 4) One user within the same browser session makes many requests. Most of the cases listed above are handled by Web server or Web container (e.g. Apache Tomcat). The last part is that we need to handle in the system. Because there is no way for handling such cases in Web servers. We have designed session management architecture to keep track of many requests and their responses within a user session. It can also maintain information to failures and history of the user.

The session manager keeps track of every submitted Web form in the portlet page and identifies them with unique ids within a user (e.g. browser) session. The manager assigns an id for each request and keeps them in a bean repository. Also session manager saves handler information within the listener repository.

5.10 Caching

Portals provide convenience to the people and reduce redundancy of tasks and gets quicker responses to the typical requests. To end up with an elegant solution to quick response times, caching is important aspect of GTLAB. On the client side of the Grid services, in other words, portal server (e.g. comprehensive client application) provides a caching mechanism. There are certain cases where caching is required including the following: 1) Client cache keeps the latest input parameters of jobs and displays them when the submission page is accessed later on. 2) Getting job tracking information is a time consuming process. To reduce response time for the user, GTLAB internally keeps

track of the jobs periodically. So it can show the tracking list on demand from the cache. User sessions (e.g. browser sessions) do not take so long. So caching makes sense for most of the cases.

In case of GTLAB caches, there is no shared session between users. Each user manages its own session. The advantage of this limitation is that does not cause inconsistency. These types of caches are considered as monotonic read [107]. As result, we should not apply coherence detection policies.

5.11 Synchronous and asynchronous

Typical HTTP requests are synchronous. So all the events occurred in GTLAB are considered as synchronous. However, backend job processing is not always synchronous. In most of the cases, jobs go into a batch queue, and results polled by the client which is asynchronous. GTLAB jobs usually take longer time to finish than a user session expires. As a result, portal server should catch responses from the backend and store them in the storage. In this case portal server runs an intermediary module to manage lifecycles of the jobs beyond the user sessions. Alternatively, job handlers to submitted jobs are stored for future references. Whenever the jobs are done the handler is used to retrieve regarding results and status information.

5.12 Architecture

ResourceBean, MonitorBean and ComponentBuilderBean (CBB) are managed by JSF's session handling mechanisms and are declared in the faces-config.xml file. CBB is not normally used directly by developers in their JSF pages. They instead interact

with this object through Tag libraries. Application developers can directly use ResourceBean and MonitorBean to build up pages.

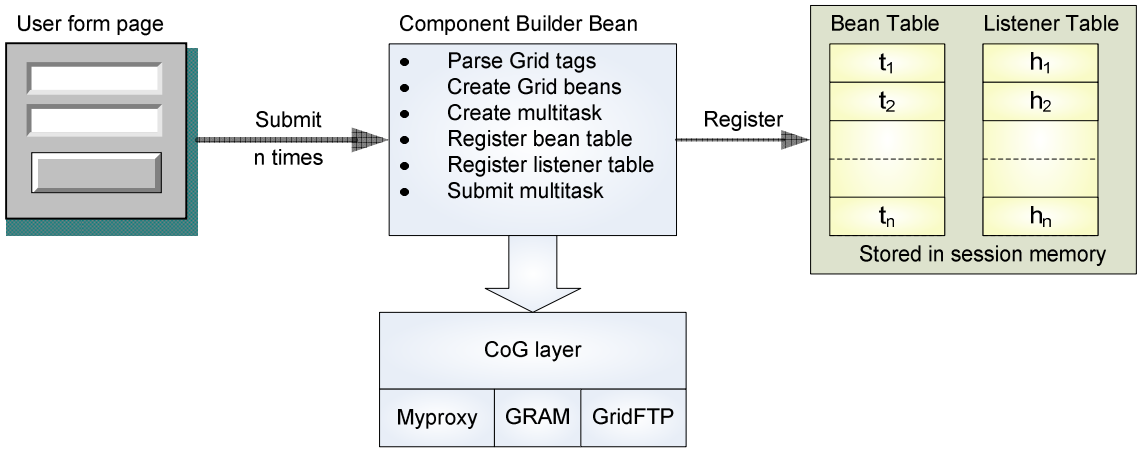


Figure 5.7 Shows architecture of ComponentBuilderBean and its components

Figure 5.7 shows the architecture of components. In this diagram, bean and listener tables are in the HttpSession and tables store bean and listener objects in a Hashmap. CBB handles user requests on the server side using Grid bean property values provided by ResourceBean. The actions are fired off by the Grid *submit* tag that is bound to the submit method of CBB. Its action listener catches the event and calls required methods to parse custom components. FactoryBean then constructs corresponding sub-tasks. Next, CBB constructs a taskgraph using MultitaskBean. CBB adds child components which are Grid beans and their dependencies. It then submits the taskgraph and passes the control to the submit button's action attribute. The JSF engine handles the value of the action attribute, while a navigation rule points to the destination page based on the attribute value.

The above classes (particularly the Factory Bean) are designed to accommodate a common use case in Grid portlets that is not handled well by JSF: we need to construct many beans for encapsulating many submissions by a single user in a single session. JSF manages the sessions (lifecycle) of beans but these are statically configured in faces-config.xml, so we need an approach to create and manage lots of Grid beans. We must also address a disparity of time scales: JSF event processing may take milliseconds, while the corresponding backend action may take much longer. Our solutions are described in the following section.

5.13 Component Parser

Component parsing based on Faces context components. The component parser processes JSF components to extract GTLAB widgets. After that attributes of each Grid tag are extracted. In the final stage, if the attribute values are not static values then value binding is performed.

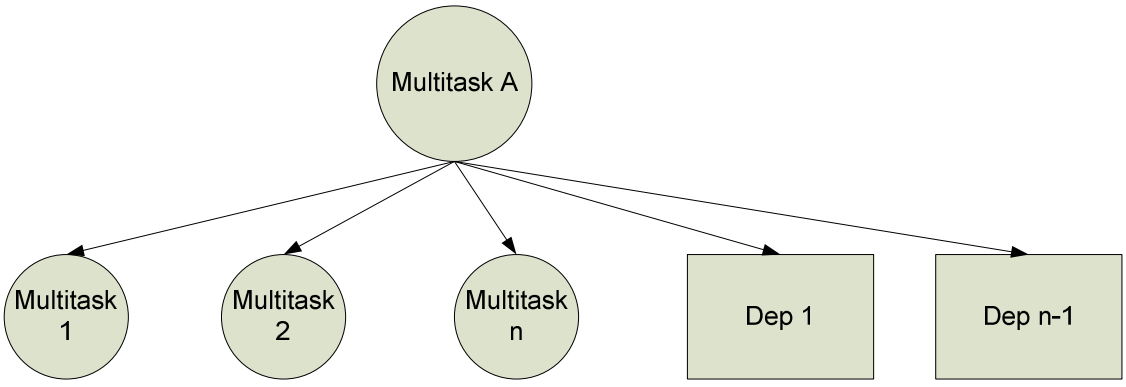


Figure 5.8 Parsing the JSF component tree that only shows tags widget

Figure 5.8 shows a component tree with the root node multitask A. Multitask A contains three Grid components file operation (fop), job submission (js) and file transfer (fs) respectively. The last two components define dependency among the Grid components.

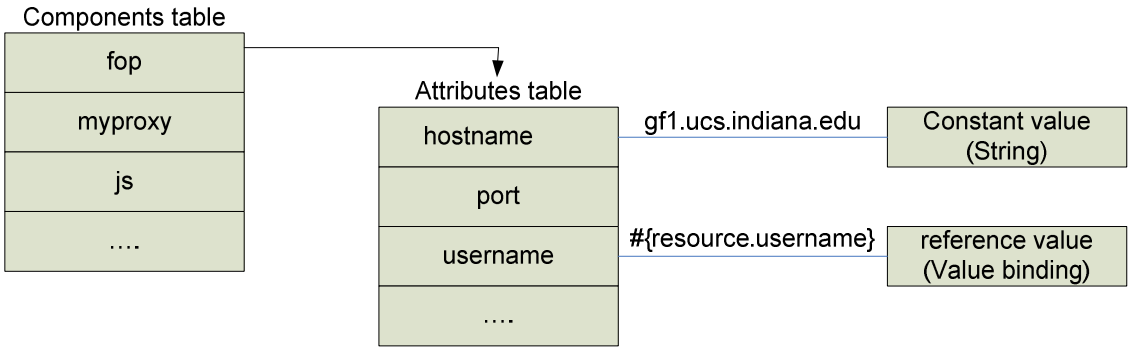


Figure 5.9 Each component has its own set of attributes and attributes can be given as constant or reference value

Figure 5.9 shows the hash tables of components and the hash tables of attributes within JSF session scope. In this figure, each component listed on the component table has an attribute table. While the component parser parsing the GTLAB tags, components and attribute values are assigned to a tree representation.

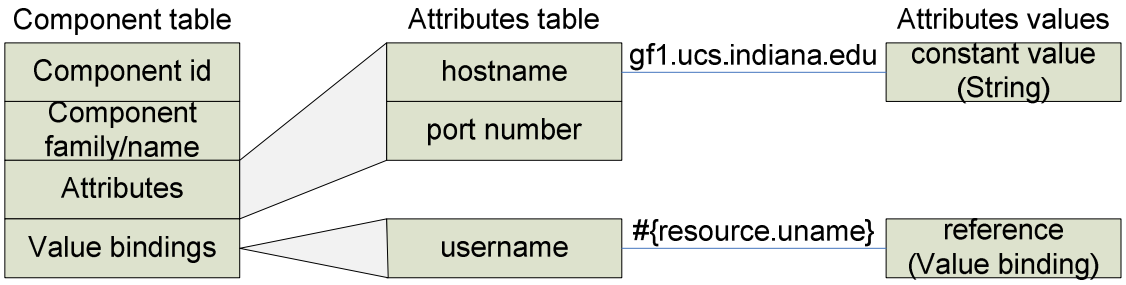


Figure 5.10 Properties of a component stored in a JSF session during component parsing

Figure 5.10 shows the component properties that stored into hash tables. These properties contains JSF component ID that is specific to JSF session management. Component name is given by the developers to label each component. Attribute and value binding references are stored in the component table as well.

5.14 Monitoring and management of jobs

Monitoring pages are responsible for keeping track of submitted tasks. Grid tasks usually take time to process. Consequently, managing the persistence of the tasks and archiving the results and input parameters are important for portal users. CBB provides a mechanism to store task handlers into persistent storage in the user's workspace. Monitoring pages collect status information and task parameters from user's workspace with a key named *taskname*. In general, CBB provides status information and updates archival storage accordingly. This has an important advantage of caching the monitoring information in the session. On the other hand, CBB stores URL handlers of submitted jobs which are provided by the Globus API. A URL handler is important for persistence. In case the user logs out or a session expires, the handler can always be accessible from archive and the user can retrieve status information with it.

Monitoring pages check the status of submitted tasks. We model tasks with a Java Bean class called JobData. Each submitted task has an associated JobData object. The collection of JobData objects is stored in a java.util.List. Job status information is displayed in HTML using the JSF HtmlDataTable component (which JSF converts to an HTML <table>). Properties stored in the JobData object include taskname, input parameters, output and error file locations, start time, finish time and status.

Portal users can manage the tasks that resume, cancel or resubmit jobs. The MonitorBean supports these capabilities for active (running) tasks. The MonitorBean allows users to manage their job archive: failed tasks may be deleted or renamed for resubmission. Successful task results and output files can be downloaded or transferred to permanent storages.

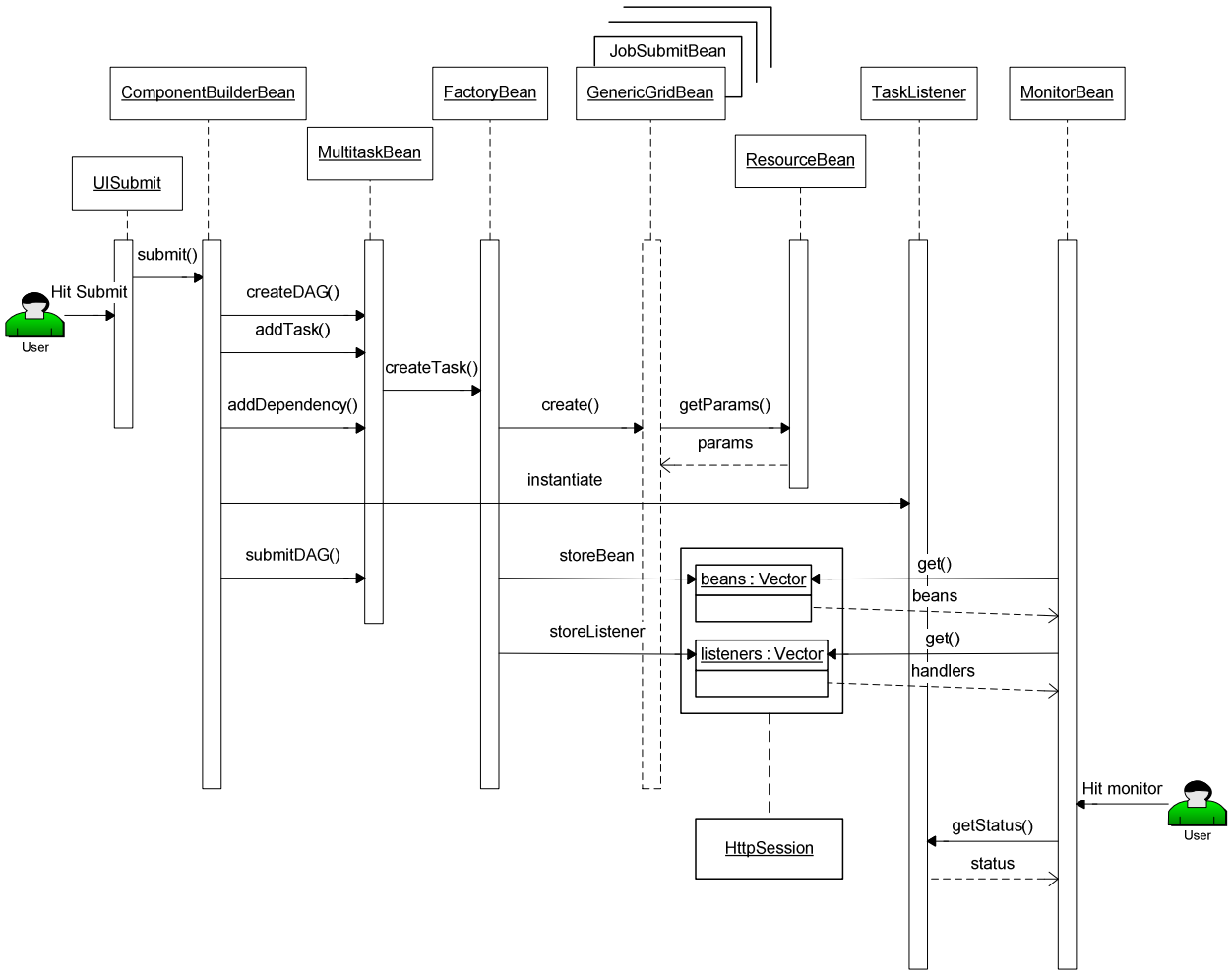


Figure 5.11 Sequence diagram for Grid tags and beans including user interaction.

5.15 Metadata management

Metadata management has been investigated by projects such as the Storage Resource Broker [108] and Scientific Annotation Middleware [109]. For VLab, we are evaluating the use WS-Context [110], a lightweight, Web Services based metadata system. A "context" is simply a URI-named collection of XML fragments. To support linked contexts in GTLAB, we used extended WS-Context implementation to support parent-child relationships between contexts [111]. Context servers are normally used as lightweight metadata storage locations that can be accessed by multiple collaborating web services.

The data collected from the user interface input form is written into a unique context associated with that user session. This data is stored persistently using a MySQL database, although this implementation detail is not relevant to the PWSCF portlet developer. Each user has a base context, which is subdivided into one child context per user session. These child contexts are used to store specific input parameter values for that particular job submission. These sessions may then later be browsed and the data recovered for subsequent job submission.

Although we may store and recover values one at a time from the context storage, we are developing a way to more easily store and recover entire pages using Java Bean serialization. We are developing XML serialization of the entire input page using software from the Castor Project (www.castor.org). This will enable the serialization of entire page contents, storing them into the WS-Context server, and then un-serializing them to reconstruct the input parameter values.

Following WS-Context specifications, a Java object may be considered to be a "context", i.e., metadata associated with a session. When storing a context, we first create a session in WS-Context store. Here, a session can be considered an information holder; in other words, it is a directory where contexts with similar properties are stored. Each session directory may have associated metadata, called "session directory metadata." Session directory metadata describes the child and parent nodes of a session. This enables the system to track the associations between sessions. One can create a hierarchical session tree where each branch can be used as an information holder for contexts with similar characteristics. These contexts are labeled with URIs, which give structured names to tree elements. For example, "vlab://users/jdoe/session1" may refer to a session directory where contexts are stored and linked to a session name "session1" and user name "jdoe". Upon receiving the system response to a request for session creation, the user can store the context associated to the unique session identifier assigned by the WS-Context Store. This enables the WS-Context store to be queried for contexts associated with a session under consideration. Our WS-Context implementation normally allows for the specification of the lifetime of the metadata. For VLab, each context is stored with unlimited lifetime as the WS-Context Store is being used as an archival data store.

5.16 Collecting User Input Values and Handling Navigation

Our Grid tags are primarily non-visual components in a JSF page that are associated with submit button actions. However, many of the tag attributes (e.g., which host to use or input file to copy) must come from user input. This is done using Web forms.

Thus, Grid tags are embedded into a complete JSF page that contains a Web form that has visual input and output text elements. There are only two exceptions: the <o:submit> and <o:handler> tags are bound to a button that triggers series of actions behind the scenes. Since Grid tags are unable to get inputs from the page, we need a mediator to communicate these user-provided inputs to our Grid tags.

ResourceBean provides a simple way to represent common property values across the application. We define common property values for Grid beans such as hostname, provider, username etc. Each of these values corresponds to Grid tag attributes. Thus, ResourceBean gets its value from the Web form dynamically and assign it to the Grid tag attribute. ResourceBean enables users to enter dynamic values in the form and submit their tasks with these values.

JSF page navigation is somewhat complicated compared to JSP page navigation, as the JSF pages' links and HTML form actions do not directly point to the next page to load. Instead, JSF navigation rules for a particular Web application are configured in the faces-config.xml file. Similar to standard JSF, advanced navigation controls the page with constant values as well. The <o:submit> button provides *action* attribute (see Figure 5.6) that assign a constant value for the destination page. Action methods and action listener methods of the <o:submit> tag is hidden from the application developers to reduce the complexity. But the navigation is left to application developers. The advantage of this architecture is that users need not wait on the submit page until it is completed. Instead they are directed to the destination page immediately (i.e., asynchronously).

5.17 Experiments

GTLAB is aimed to decrease Grid portal development time, at the time GTLAB should not introduce unacceptable request processing overhead. The overhead is the cost of processing of job requests within GTLAB framework. As shown in Figure 5.13, end user requests are caught by portal server and GTLAB parses and extracts Grid tags from portlet pages. In the next step, Execution steps are created by calling appropriate Grid bean instances. By this time, job parameters and bean handlers are stored in the hash tables for future references such as tracking the progress of the jobs. Finally, the request is passed to the Grid service by invoking corresponding services like GRAM or GridFtp.

We performed run-time tests to analyse GTLAB architecture by determining overhead in the overall processing time of the requests. Our testing baseline and testing framework is explained in great detail in the next section.

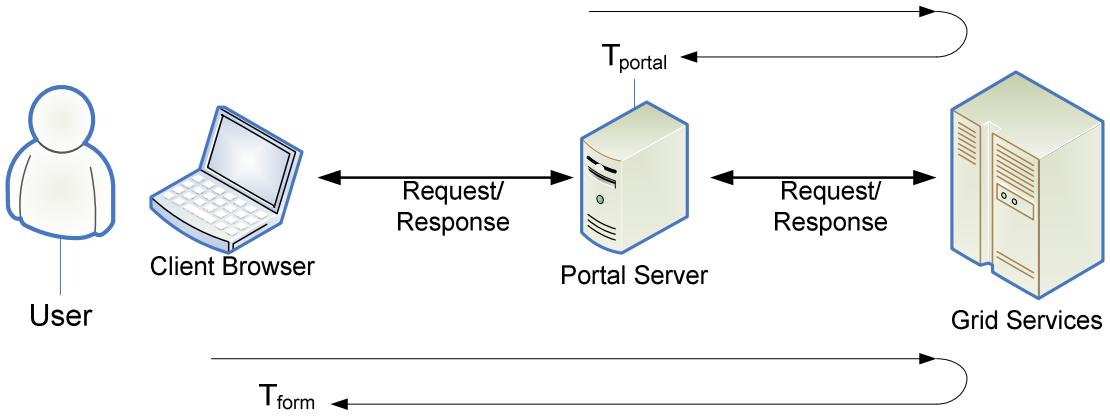


Figure 5.12 JSF applications uses Web forms through lightweight Web browsers. HTTP requests goes to the Web application on Tomcat and responses get back to the browser.

We expect that the most time consuming task of portlet development is creating Grid bean instances. While we integrate GTLAB to construct portlet pages, we observed that it reduces development time. We experienced these during the course of our use case Grid portals including QuakeSim, and VLab.

GTLAB testing server runs on a Tomcat server and these tests aim to measure turnaround time on the server and client sides. Clients make extensive number of requests to show performance and thresholds. We have measured elapsed time for starting and ending of requests. There are two testing case: i) Response time for requests are initiated from browser client by submitting Web forms. ii) Response time for requests first comes to the tomcat server and responses leaves the tomcat server.

5.17.1 Testing Setup

GTLAB testing server runs on GridSphere portal framework and these tests aim to measure response time between the portal server and end user sides. End users make extensive numbers of requests to measure the timings. The elapsed time is measured at each request and response in Figure 5.12. Our testing case is for response time in between end user and Grid services denoted by T_{form}.

The end user requests are launched by using HttpClient programming interfaces [112]. HttpClient provides an interface to feed Web form parameters and submit actions. We have embedded simple DAG into portal submission pages to execute scripts at GRAM service. When the DAG runs, it first obtains the Myproxy credential from Myproxy repository and then submits shell script commands. In order to get

elapsed time accurately, we have taken "submitted" message that is the initial acknowledgement from the Grid service into account. Since GRAM jobs are queued at service location, departure time cannot be determined by waiting for "completed" message.

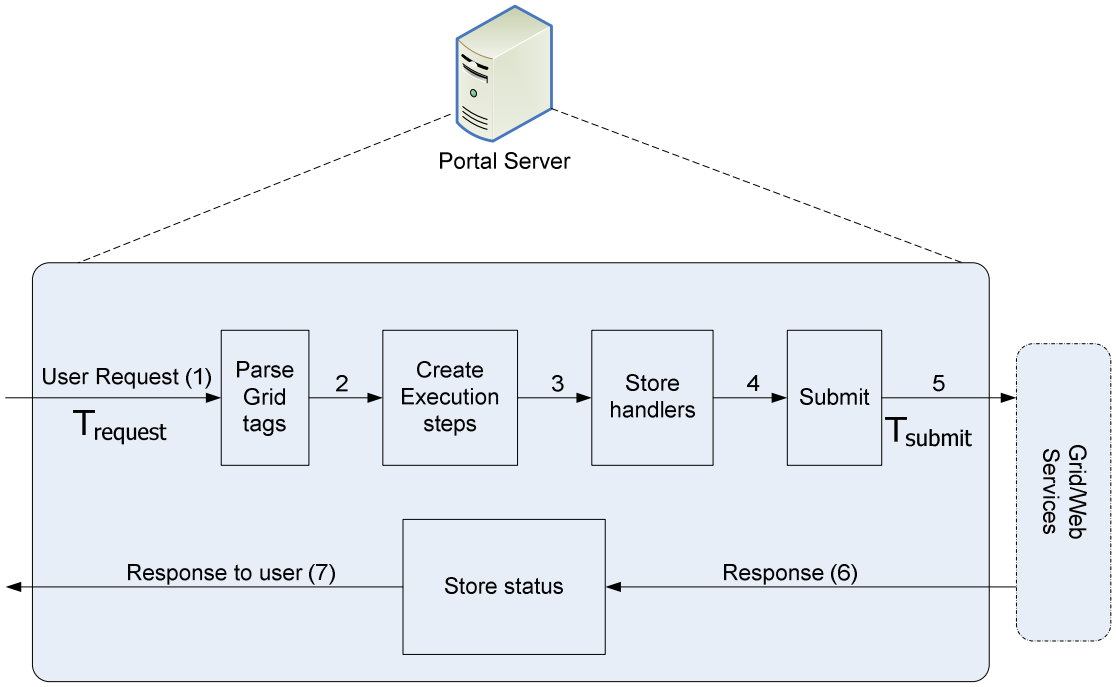


Figure 5.13 Request processing stages and their timing in portal server

Figure 5.13 shows detailed steps that are magnifying the processing stages at the portal server. At the first stage, user requests (1) are parsed as using JSF component model. The Grid tag components are extracted from portlet pages and then the graph structure is constructed by keeping the dependencies among Grid operations. Next, Grid operations on the graph are assigned to Grid beans that are supported by Java CoG [66] libraries at (2). While Grid beans are created, the handlers of the tasks are stored in hash tables at (3). Finally, the submit action is called at (4) that invokes a Grid service.

The acknowledgements and status changes are stored by handlers of the jobs. Grid services send response messages that may be a “submitted”, which is for successful job submission or “failed”, which is for failure of job submission. At the final stage, the response message is directed to the end user through the Web browser (7).

5.18 Analysis of GTLAB architecture

Test scenarios are conducted to measure the overhead on TeraGrid nodes including IU, NCSA and TACC. The test results have shown that GTLAB framework has acceptable overhead as indicated on Table 5.3. The average overhead is about 150 millisecond.

Figure 5.14 shows the results of average response time. The heights of the bars indicate response time, T_{form} . In this test, we wanted to give an idea of response time measures. Therefore, the average response time is always less than 1 second which is an acceptable time for Web applications. Since T_{form} includes network transfer time between users, portal server and Grid services, in addition to processing time on the portal and Grid service.

Figure 5.15 shows the average network latency, when users access to the portal server. Comparing with the average processing time on the portal server, the network latency is an acceptable amount of time. Another interesting aspect of this measurement is about the end user locations. Locating the users among different cities does not cause a significant delay on user requests.

Table 5.3 Timings of GTLAB processing stages on the portal server

	GTLAB Processing	JSF Processing	Handler storing	Submitting
Time (msec)	2	153	1	410

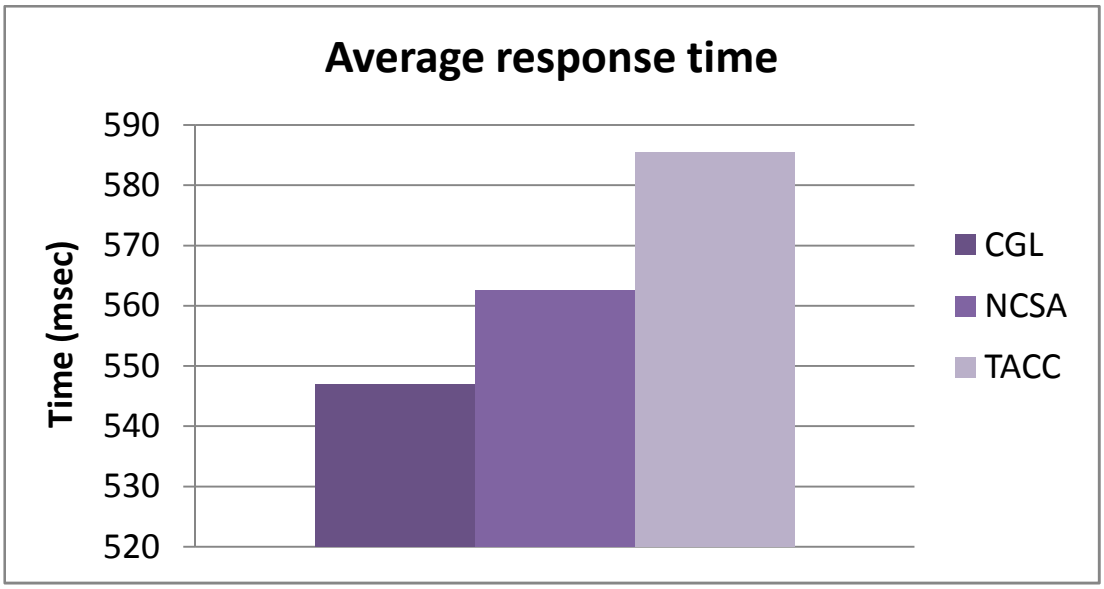


Figure 5.14 Average response time of requests initiated by end users, T_{form}

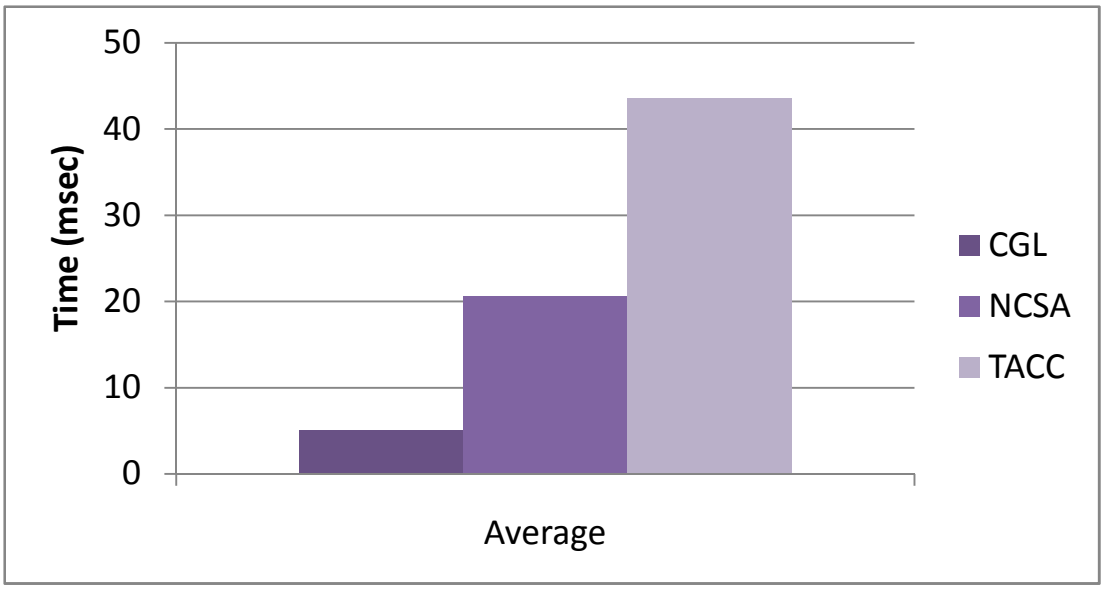


Figure 5.15 Average response time in between and end user and portal server

5.19 Future works: Applying GTLAB into Web 2.0

This section addresses the issues of applying Grid tags to Web 2.0 tools. Grid tags should support client side implementations for browser clients. This will show that GTLAB architecture is portable among various technologies. The important aspect is to convert Grid beans as services that Web 2.0 gadgets and widgets can directly access to them and utilize these services through well known client tools such as Google Web Toolkit (GWT) [113], Direct Web Remoting (DWR) [114] or Asynchronous JavaScript and XML (AJAX) [115]. We have sketched these main questions to review the GTLAB components. GTLAB components would work generally within any technology or tool.

5.19.1 Discussion

This discussion addresses the issues of applying Grid tags to JavaScript (JS), AJAX or Web 2.0 tools. Grid tags should support client side implementations for browser clients. This will show that GTLAB architecture is portable among various technologies. The important aspect is to convert Grid beans as services that Web 2.0 gadgets and widgets can directly access and utilize these services through well known client tools such as GWT, DWR or AJAX. We have sketched these main questions to review the GTLAB components. GTLAB components would work generally within any technology or tool. We have raised the following design questions:

Find out how to manage session in JS?

How to embed tags in the JS?

How to maintain user/cookie with JS?

Cookie management is only for JSF type server side applications. HTTP is not stateful so cookies maintain states for user sessions. But in the case of WSRF the server side is already stateful, so we do not need to worry about cookies. WS-Notification [116] provides a callback system for submitted jobs.

How to use shared memory for JS session?

If WSRF keeps the state on the server side, client would not need to worry about the session/shared memory ever. But the current GTLAB framework keeps many job submissions in the session and stores them in the hash tables within session memory.

How to extract and parse Grid tags from the JS page?

JS pages are HTML or XHTML pages. So they are structured. If the parse the whole page, we can easily extract the Grid tags out of the page. If possible, using XHTML helps to keep the pages simple and machine readable.

How to handle Web forms within JS?

This question has two parts. 1) How to submit Web forms? 2) How to handle user inputs to text areas?

In case of JS implementation of GTLAB there won't be any Backing bean implementation; rather all bean interfaces and capabilities are already discovered and implemented in Java. We need to implement these interfaces using the scripts. So the action scripts are bound to 'Submit' or action buttons that calls the right methods to access Grid services.

On the other hand, GTLAB manages user inputs through Resource bean. There won't be any bean in the scripts. But the same idea inspires the scripts. So we can implement get/set script methods to share common input values. These values are kept

in the session memory. Then the values are passed to the tag attributes. These are detailed implementation issues or details. We need to worry about the major architectural problems that are addressed in the first five questions.

5.19.2 Web gadgets

Gadgets can access to Standalone JSF code using HTTP object and performing POST operation. If you assume portlet pages there, you have to define a workflow to submit the job.

MyProxy → File transfer → Job Submit

The issue here is to store MyProxy in the session where gadget runs on. The other important aspect is to perform authentication thorough gadgets.

5.20 Summary

In this chapter, we have summarized the architecture of Grid tags and beans and we have reviewed the research issues that we raised. We have sought solutions to session management of requests in GTLAB that can handle requests. We consider our persistency mechanisms have also long term caching capabilities. We also handle synchronous request/response model in asynchronous fashion. We attempt to manage jobs and monitor the progression in timely manner. We have conducted experiments to show GTLAB does not add any overhead to existing servlet container. Our results are shown that the overhead is negligible for our architecture.

Chapter 6

Applying workflows to Grid portals

6.1 Foundations of scientific workflows

Scientific workflows compose, execute and monitor multiple jobs in a logical sequence. Scientific workflows symbolize sophisticated application patterns. These are usually typical scientific experiment analyzing procedures. In some fields the experimental research are based on legacy workflow applications. For instance, bioinformatics community uses Taverna tool to develop and enhance bioinformatics workflows.

There is lack of knowledge when developing scientific workflows. Some scientific fields already have constructed conventions and experience on building workflows. But most of the science communities are new to these emerging technologies. Even some of them have very limited knowledge and background on using computer-based

applications. On the other hand, many disciplines have stovepipe solutions to organize metadata for processing the simulations. This transition process for converting applications to computer-based workflows becomes a foundation for building workflows. A typical example of scientific workflow is to collect raw data from sensors, instruments and similar sources. Then store them in high capacity storages. Following by feeding application program to utilize the data and get some results. Results can be shown to scientists in certain ways: 1) Raw data 2) Plots 3) 2D images 4) 3D images 5) Animations. The form of showing the results of the applications is called visualization.

The applications can be characterized in very broad range in our study. Some applications are time sensitive, real time. Some of them controlled by the users and require human intervention. And most of the applications are enacted as "batch" process. The following is a list of application domains that we worked on.

1. Material science applications
2. Instrument and sensor based applications
3. Earthquake modeling applications
4. Molecular Science

Scientific workflows are available to use in certain scope of computer software: including operating system, application software, middleware applications. For instance, workflows are available for local clusters in operating system level. Scientists can facilitate these applications by using command-line scripts or desktop tools.

6.2 Importance of workflows in Grid portals

Grid environments and science gateways are utilizing applications run on different domains and on the Web. In order to support interoperability on the Web, we usually use Web services interfaces for legacy applications. Grid portals also provide portability of application processing and more importantly monitoring. Another aspect is to aggregate a lot of applications in well accessible platform. It is very similar to a shared desktop application that scientists can access through the Web.

Science applications run in a logical order, tasks depend on the previous task results. Grid portals aggregate science applications within a flow control. Hence, Grid portals collect all the features of workflow tools in a Web environment. The portlets are responsible for composing a workflow, executing it and keeping track of its progress. Portal events are part of Web application framework such as starting a workflow, stopping it, and resuming it. There are difficulties to maintain all the events occuring in the Web environment. Events are initiated by end users by submitting a Web form within a browser session. The portal server needs to handle requests from different users and keep track of the user sessions in asynchronous environment.

In a general approach Grid portals use portlets to split workflow processes into reasonable parts. For instance, composer could be a portlet, executer could be another portlet and monitor is also portlet. These three portlets can be grouped as in a Web application called workflow portlets.

Interoperability of portlets is another issue with portals. Portlets should talk to each other in a straight-forward fashion. The pushing of notifications within portlets allows workflow portlets to trigger a chain of events. Therefore, a workflow portlet can

interact with every other component within portal application. Such workflow can update calendars, drop messages to personal mailboxes etc.

6.3 Legacy workflows for Grid systems

There are numerous workflow frameworks exist in Grid systems. Most of the workflow frameworks utilize Web services to facilitate interoperability including Taverna, Kepler [117, 118], Business Process and Execution Language for Web Services (BPEL) [119], Pegasus, and Chimera [120]. We are going to evaluate Taverna and BPEL workflows in great detail in the following sections:

6.3.1 Taverna

The Taverna project has developed a tool for the composition and enactment of bioinformatics workflows for the life sciences community. The tool includes a workbench application which provides a graphical user interface for the composition of workflows. These workflows are written in a new language called the simple conceptual unified flow language (Scufl) [121], where by each step within a workflow represents one atomic task.

The new conceptual language is represented as XML based syntax that can be deployed as Web services. Bioinformatics requirements led to the specification of Scufl. This conceptual language process steps of the workflow that represents atomic tasks. A workflow in the Scufl language composed of three main elements:

(1) Processors

(2) Data links

(3) Coordination constraints

Processors are applications or filters that take inputs and generate outputs. Data links pipes the flow of data between processors. Coordination constraints restrict the execution between two processes where enforcing an order among processors.

Scufl is supported by the Freefluo [122] enactment engine where Freefluo executes the Scufl instances. Taverna system which includes Scufl, Taverna workbench and Freefluo is widely used in several Genome and Bioinformatics project and tested by their user community.

6.3.2 Kepler

Kepler [117] is a scientific workflow environment in which scientists compose, execute and control analytical procedures. Kepler provides Graphical User Interface (GUI) for design and execution tools to support actor-oriented modeling paradigm. Kepler workflows can be sketched in XML. Kepler is built upon Ptolemy [120] system that controls the execution model via directors. Workflow steps are described as actors that can utilize data sources, sinks, filters etc. Actors can have multiple input and output ports to direct the flows. Also parameters specify behaviors of the actors.

Actors can run on a local runtime environment as well as extend distributed execution via Grid and Web services. Kepler currently supports Java Native Interface (JNI) for different language platforms as well. Other workflows like Taverna are all based on a single dataflow execution model, while Kepler handles many.

6.3.3 Karajan

The Karajan [123] workflow framework provides access to Grid services by using an XML-based definition language. Karajan can be utilized in various platforms. Karajan

has its own parallel and structural language that is adopted for Grid services needs. Users can define jobs and their lifecycle management using the Karajan language. Karajan scripts are run by a Karajan engine, which may be embedded in a Karajan service.

Karajan service is a workflow engine that is accessible by several ways such as polling, call-backs, and persistent data retrieving. When users submit their workflows, Karajan service interprets these inputs through Karajan libraries. Then the engine creates client stubs for the tasks defined in Karajan script. These tasks are submitted to the Grid services using Java CoG abstractions.

6.4 Handling Directed Acyclic Graphs in GTLAB

GTLAB is designed to utilize several DAG frameworks in Grid computing including Globus [61] toolkit (by using Java CoG interface "taskgraph") and Condor DAGMan (by using the Birdbath [67] Web services interface). DAGs are built by application programmers and are embedded into JSF portal pages. Grid tags help to compose DAGs with dynamic parameters entered by end users within portlet pages. Grid tags are also responsible for executing workflow by initiating 'submit' tags. In Figure 5.4 the first job moves the input file from a remote host to the execution host. The second job runs a script on the execution host depending on completion of the first job. In other words, the script cannot run unless the input file is ready on the execution host. Finally, Grid tags allow users to keep track of the execution of the DAG by facilitating handler tags.

Java CoG Taskgraph: Java CoG encapsulates Grid services clients in abstract interfaces. In addition to Grid services, CoG interfaces also introduce a DAG expression capability, which is called *taskgraph*: pipelining a few Grid services calls as DAG. A CoG taskgraph is a DAG interface that is built using the Java CoG API. The CoG DAG builds workflows on top of Globus toolkit services such as GRAM and GridFtp. All service calls and their order and dependency relations are defined within taskgraph interface.

GTLAB implements a layer on CoG API that is encapsulated by XML tags. For instance, the *taskgraph* interface is used by <o:multitask> tag. These XML Grid tags are supported by Grid beans. Grid tags are injectors for Grid beans (using Inversion of Control design pattern [30]). They initialize beans and manage their lifecycles. A <o:multitask> tag can define attributes (see Figure 6.1) for taskgraph including *id, taskname, handler and persistent*. *multitask* also can contain dependent task objects are represented as sub tags including <o:myproxy>, <o:fileoperation>, <o:jobsubmit>, <o:filetransfer> and <o:dependency>.

Application developers compose their DAG scenario by using Grid tags and beans together within GTLAB framework. In this case, DAG attributes are filled by the developers. Some of the attribute values are application dependent and so they are static. For example, the Globus toolkit provider attribute can be set as *GT4* for entire portal. On the other hand, some parameters are provided by the end user through input forms. These attributes must bind HTML input text by using expression language semantics within JSF.

Condor DAGMan: Condor [34] is an environment for scheduling and executing applications on distributed networks of computers. DAGMan is a tool for describing complex application workflows to be executed on Condor in terms of directed acyclic graphs. In this case, GTLAB allows the user to prepare or transfer descriptions of Condor jobs or workflow scripts described with the DAGMan. End users can return later and monitor the progress of the jobs.

Condor manages job submissions to Globus-based Grids through Condor-G [69]. GTLAB provides a web application environment which can turn out to be a portlet for Condor DAGMan by introducing two additional JSF Grid tags: <o:condorDagman/> and <o:condorSubmit/>, which we describe below.

<o:condorSubmit/> is for single job submission to Condor-G resources. <o:condorDagman/> is used to describe composite DAGMan jobs and their dependencies along with a scripting file. Similar to our <o:multitask> tag, these tags provide access to Condor services in terms of using Condor beans. Our Condor beans have capabilities to prepare Condor jobs, submit jobs to Condor resources and manage the lifecycle of submitted jobs.

Condor has no equivalent Java client libraries that correspond to the Java CoG for the Globus toolkit. However, Condor provides Web services interface called Birdbath [67]. This provides an XML abstraction of the programming interfaces that can be bound to different languages such as (in our case) Java. Our Condor beans are built on top of Birdbath Web services clients. The Birdbath layer allows us to program Condor capabilities within Java Beans and associated Grid tags allow us to describe job

parameters by using dynamic Web interfaces. In this case, we use Web services clients to program Condor, instead of using command-line interface.

Birdbath client stubs are developed from the BirdBath Web service interface. We have packaged Birdbath clients as jar and added it to the library.

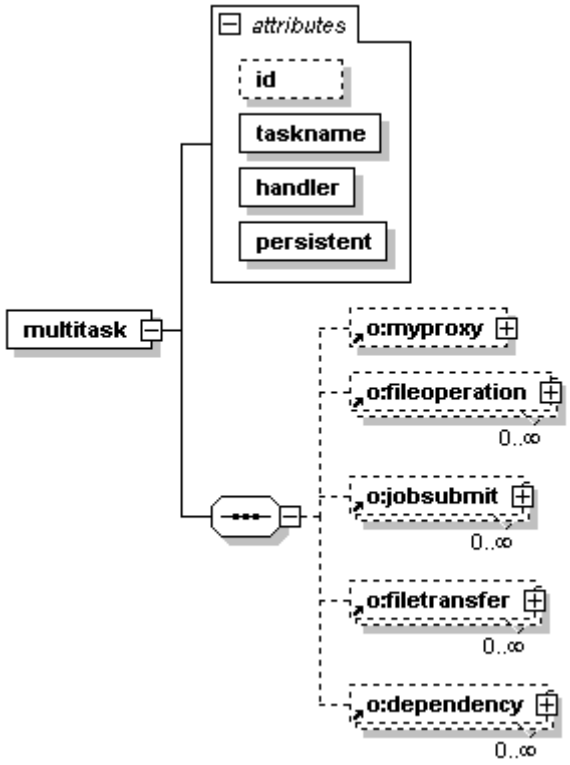


Figure 6.1 XML schema of multitask represents a DAG. It shows the relationship of Grid tags by defining dependency tag in GTLAB.

6.5 Design and Architecture of GTLAB Workflows

We consider in this section strategies for supporting more complicated workflows than can be represented by DAGs. Our goal in GTLAB is not to reproduce extensive pre-existing work in this field but to instead take advantage of it.

DAGs are very useful in case of simple workflows such as submitting a few tasks in a group. We have added new features to GTLAB such as the ability to build sub-graphs to allow partially ordered tasks. Partially ordered tasks can group the sequence of the tasks based on their dependency. But in case of enhanced workflows, DAGs are not sufficient. For example, if a user needs to run the simulation many times with a DAG, the DAG has to maintain loops. If an application portlet needs to provide dynamic flow control based on constraints, the DAG has to support conditional branches. Those features do not exist within DAGs. Thus, a scientific community has to facilitate these capabilities; they need to use workflows that cannot be expressed as simple DAGs. Directed graphs naturally do not handle this type of data structures. Our solution for supporting these more complicated workflows is described in this section.

Workflows are sophisticated flow control mechanisms of group of tasks. The foundations of Grid workflows are described in a special issue of *Concurrency and Computation* [39]. The tasks could be parallel, sequential, or concurrent. Workflows can handle loops, branches and conditional branches. Workflows can be overviewed in three main parts: 1) Composer, 2) Enactor, and 3) Monitor.

Composer: The composer is an essential part of the workflow representations. Workflows represent services as nodes and constraints as edges to the nodes. In this case, the top node is the starting point and intermediate nodes denote tasks and local filters. Edges denote dependencies. This structure could be a graph where nodes correspond to tasks and edges corresponds to relations. Also direction of the edges can limit the flow similar to flow charts.

Figure 6.2 shows a composition of three Grid tasks in order. Supporting Grid services within Taverna is an interesting approach. Taverna can utilize some services with local clients such as MyProxy. Although other Grid services already have Web services interfaces in GT4. These services can be scavenged to Taverna. Therefore, we are able to manage Grid services workflows through Taverna.

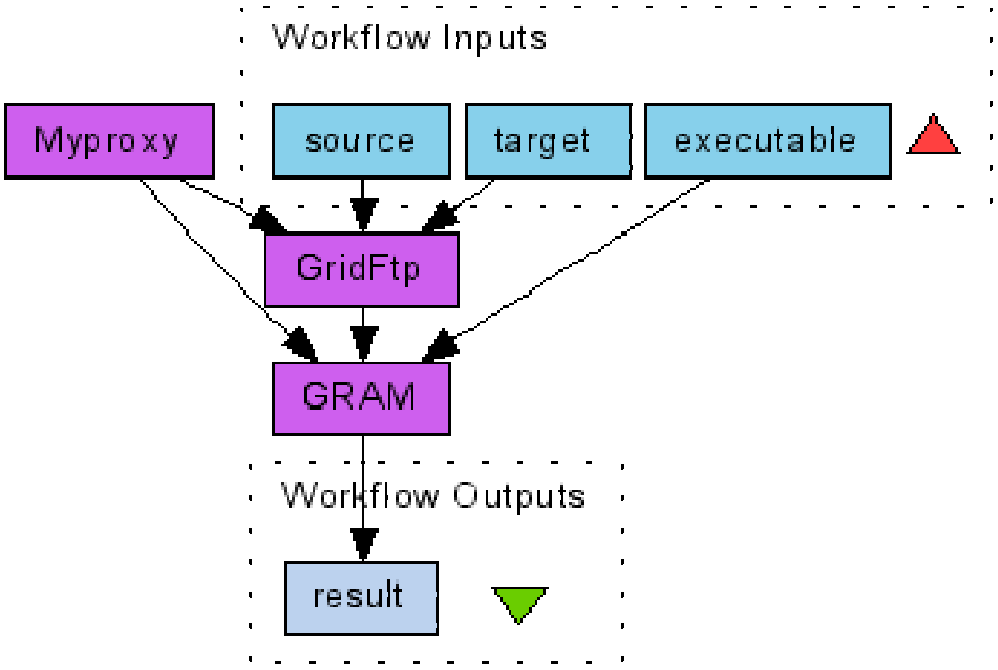


Figure 6.2 Taverna composition of three major Grid tasks in a workflow

Enactor: An enactor is a workflow engine that process nodes in the order determined by the composed graph. End users provide values for workflow inputs. Workflow processing results in with workflow outputs. An enactor can pipe inputs to one action

that is output of the previous one. An enactor also maintains constraints, branches, loops and parallelism.

Monitoring: Monitoring follows up the processing steps. It also manages lifecycle of the workflow. End users are able to interrupt the workflow to pause or cancel the execution of the workflow at each step.

Unlike DAG composition within GTLAB, workflow composition is out of scope of this work. In case of DAGs there is only a few tags definition exist such as *multitask, jobsubmit, myproxy, fileoperation* etc. GTLAB provides one-to-one mapping for each entity in DAG definition. But workflows are more comprehensive than DAGs, and there are many entities to define a complete workflow. Workflow policies are described by their own composition language. We either need to provide one-to-one mapping of each entity that exists in the workflow language, or Grid tags could import workflow policy as whole within GTLAB. We prefer the latter, to embed built-in workflow files into the enactor. The enactor takes the workflow description file as an input to start workflow engine for execution.

Our strategy for supporting workflows is as follows: GTLAB framework binds an enactor engine to a 'submit' button within a Web form on the portal page. Once the button is clicked by an end user, the enactor engine takes control of workflow along with the composition document. These workflow documents are already checked for validity. Workflow frameworks define their composition rules as explained in great detail in the next section. Finally, the engine starts running at the backend to process action steps.

GTLAB monitoring features are listed as status updating, cancelling, pausing, and resuming the jobs. GTLAB assigns unique handlers for all submitted workflows within the user session. These handlers are associated with 'handler' tags. The handler tag utilizes the capabilities of monitoring bean by using attributes and sub-tags.

6.6 Taverna Use Case

Taverna is workflow tool for composing and executing Web services. Its main target is bioinformatics applications, but it can in fact be applied to general workflow composition problems. Taverna includes a graphical user interface workbench that is used to formulate workflows. The Taverna workbench solves issues of complexity of the workflows by providing user friendly interface. The workbench facilitates diagrammatic and explorer representation of workflows. It allows users to compose their own workflows or to load previously designed workflows, such as may be obtained from a community repository with expert contributors. The workbench also lists the available resources (e.g. Web services) where the workflows can run. After the resources and enactor engine types are selected by a user, he or she can start the workflow and can monitor progression. The user can interrupt the workflow for cancelling a step or stopping the workflow.

The Taverna workbench relies on XML-based Scufl workflow scripting language. Scufl consists of a network of processors and links. In addition to basic entities, Scufl also can have input and output nodes and constraints for processors. The Scufl language primarily is designed for users who are familiar with Web forms and scripting

languages to use Web resources. Scufl is practical and is designed with extensibility features.

Workflow portlet: Generally a workflow portlet should contain these three major parts: 1) Defining workflow components and their relationships. 2) Executing the workflow: in case of Scufl, we use the Freefluo [122] enactor engine. 3) Monitoring execution flow and applying capabilities like resume checkpoint, cancel, remove, etc. Typically the first and third steps are tied to a strong graphical user interface such as the Taverna workbench.

Building a workflow composition environment with the graphical user interface features require many visual designs to accomplish with a success. The Taverna workbench is already available for composing workflows. Building a workflow composer out of Taverna is out of GTLAB's scope. But we can alternatively provide a text field to compose workflows in XML (e.g., Scufl) on the portlet page. However there are two drawbacks of this approach; 1) it is hard to catch syntax errors when composing a workflow, and 2) the Scufl document should be validated against Scufl schema. This process is offline and requires additional efforts. Scufl composition is out of scope for our current GTLAB work. However, it is common for Scufl-defined workflows to be reused and shared between developers, since many scientists are interested in the same basic workflow.

The workflow portlet application utilizes extended GTLAB features to submit Scufl workflows. This portlet loads a Scufl workflow file, collects input values from end users, submits the workflow on Taverna, and monitors the results inside the GTLAB session framework.

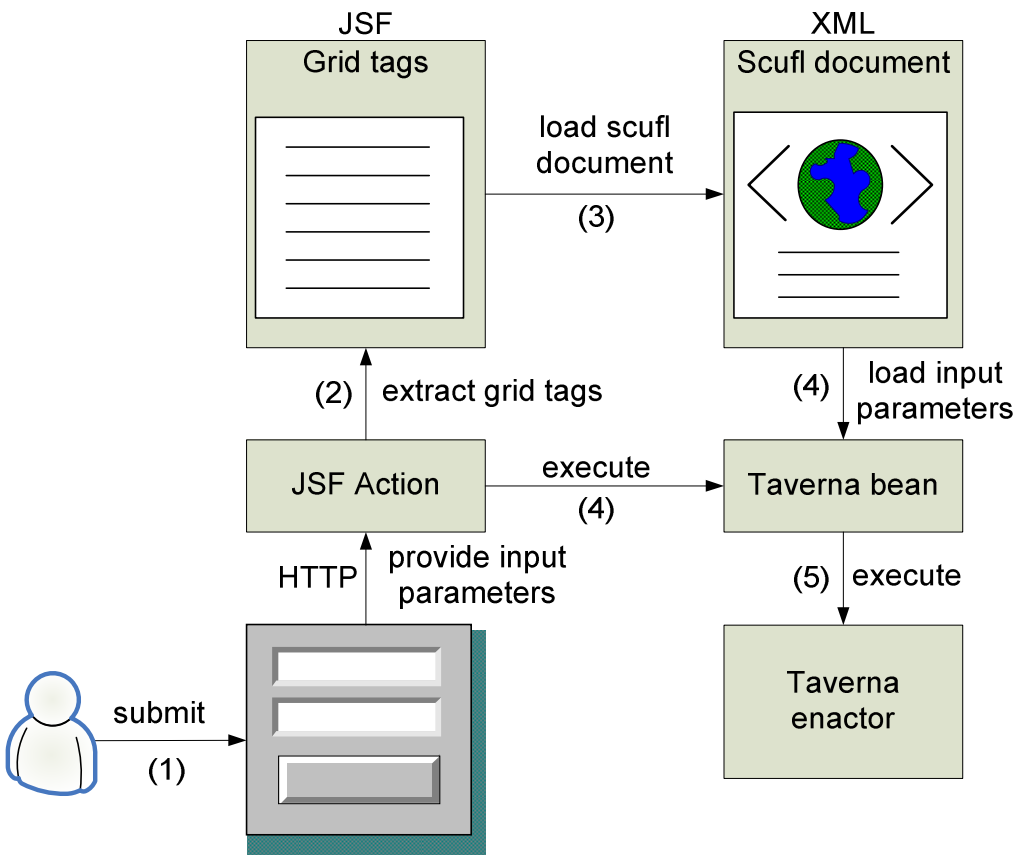


Figure 6.3 A user interacts with a workflow portlet to utilize Taverna enactor. User provides parameters by submitting a Web form that start the chain of events in order.

Figure 6.3 illustrates the handling of Taverna tags within GTLAB. In this case, Taverna tags are embedded into JSF portlet page integrated with a Web form. End users only see the Web form with a few text fields and submit button. They never see the Grid tags and JSF tags that build the portlet page. This is common for all Web applications. When the end user submits a Web form through the portlet page, JSF intercepts this request and calls the associated action methods of Grid beans. Next, Grid

beans load the appropriate Scufl document and input parameters to Taverna bean. Finally, the bean method starts execution of the workflow on Taverna enactor.

GTLAB assigns job handlers to each submitted workflow within the user session so that keeping track of the progression. In case of Taverna, the handlers synchronize with Taverna monitoring services to follow the workflow states.

Taverna Security: Taverna generally works in non-secure environments with Web services that can be used anonymously. The Taverna workbench uses local filters and scripts. The main concern of bioinformatics community is to process massive data by using complicated workflows. However, security is a critical issue in Grid services that rely on secure connections. Traditional Grid services apply GSI security [73] by using X.509 certificates. Since Taverna can facilitate emerging Web services technologies, we need to employ WSRF [60] and GT4 services within Taverna. Therefore, it is essential to address whether WS-Security [124] is applicable to Taverna processors. Similar problem has been addressed by [125] extending Taverna workbench adding new processors that support WS-Security.

Figure 6.4 shows an alternative approach to protect Taverna services by PERMIS system. In this case, we restrict general user accesses to Taverna services through portal. Therefore, the authorized users can only enact Taverna workflows within their portal user spaces.

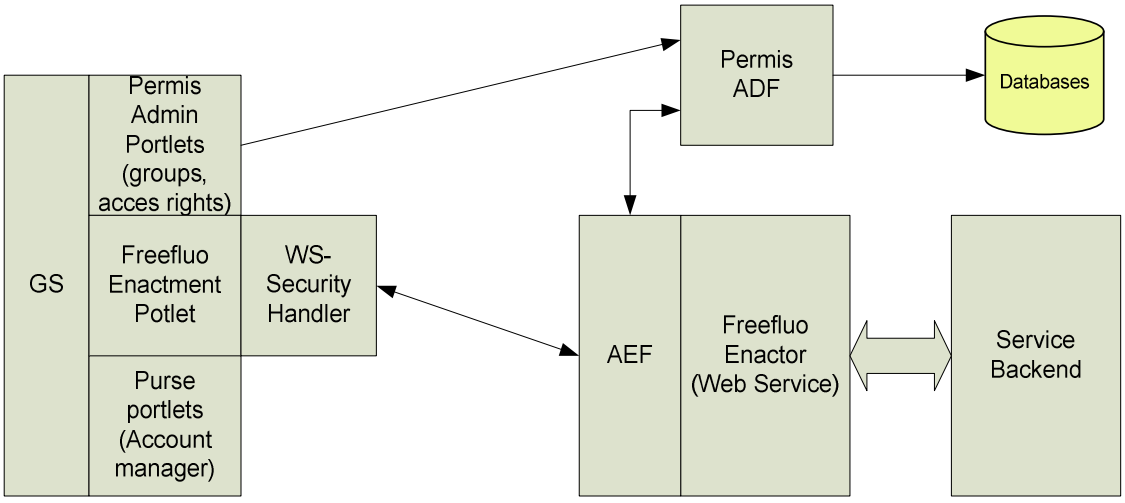


Figure 6.4 Grid portal support Taverna and PERMIS authorization schema.

6.7 Persistency issues of workflows within GTLAB

Grid workflow compositions are usually provided by community scientists. Workflows are well-studied experiments and they are reproduced by the users. As a result, keeping good workflows in a repository and then accessing with provenance is crucial for Grid portals. There are various ways to provide a well established persistent repository for workflows and data. i) File systems may be persistent repositories, ii) database access is another way of keeping resources persistent. iii) WS-Context service provides service based access to the repository. The advantage of using service based storage is to provide provenance and ontology.

6.8 Discussions and Conclusion

Workflow extensions to GTLAB increase the usability of Grid tags in wide area of scientific applications. Most of the science gateways are managing execution steps

intensively. Such a case is the VLab portal that facilitates simulation parameters and refines them within first round of iterations. In the next stage, application is started and results are shown in visualization environments. Similarly more complex science gateways can utilize GTLAB framework for their application systems.

Portal persistency mechanism which was used in VLab provides a repository of ready-to-use workflow compositions. The persistent storage can be accessed by metadata and provenance. Inexperienced users can first try the selection of good examples to exercise the best practices in that scientific arena while they are passing the learning curve.

Different workflow compositions are persisted and archived in the repository while their execution steps are closely watched and are reported. These stored steps of events can be use for future failures and enables the system to roll back to stable level and rerun form that point.

In this chapter, we have evaluated our initial VLab portal development work, which constructed workflows for Material Sciences that are based on DAGs. We provided support Globus toolkit by using Java CoG. Extending this initial work, we have added support for Condor DAGMan by using Birdbath services. We have also evaluated how to extend our architecture to support more complicated workflows and have implemented support for Taverna workflows. This allows us to deploy and manage more comprehensive workflows using Web services. We have designed additional Grid tags for Condor DAGMan and Taverna workflow. In conclusion, we showed that our GTLAB framework is extensible and applicable to different types of workflow frameworks.

In the next section we will discuss adapting GTLAB framework to work with BPEL and Kepler workflows.

6.8.1 Discussion of Kepler and BPEL extensions

Kepler actors can have multiple input and output ports to direct the flows. Also parameters specify behaviors of the actors. Actors can run on local runtime environment as well as extend distributed execution via Grid and Web services. Unlike Taverna that is based on a single dataflow execution model Kepler enables workflows to handle multiple flows.

GTLAB is trying to abstract all different workflow approaches in a portal container that users can customize application specific workflow mechanisms. For example, application users compose the generic workflows for specific tasks. Then in the usual case, end users enact these workflows by providing parameters while we provide additional parameters that end users can choose to run the experiments in different workflow like Taverna or Kepler. Although, the end users are technically able to run any workflow, they should also know whether their system can run this workflow. On the other hand generic workflow helps people from diverse disciplines. Generic workflow portlet can run on either Taverna or Kepler.

Chapter 7

Portlet Access Control Mechanisms

One of the benefits of Grid portals is to grant personal workspace and to provision computing resources in the Grid. Grid tag libraries are very useful to manage Grid backend services that are individually used by a specific user. GTLAB provides a workspace environment to each user that can manage jobs, credentials and metadata. On the other hand, not only scientists utilize the personal accounts, but also they collaborate on the observations and they exchange data. To this extend, we need to provide an access control mechanism to allow scientists to share data for collaboration.

In this chapter, we review CIMA portal to point out interesting access control problems exist in the current JSR 168 portlet specification. We also give detailed architecture of CIMA about how we solved these issues.

7.1 User account management in Grid portals

Portals aim to give different look and feel environments to the users. Users are able to customize their own browser windows with their selected applications. In a simple science gateway, there are many types of users can access to the gateway. Portal administrators are responsible for configuring and maintaining the portal resources. The scientists manage core applications and naïve users who browse the results or see the effects of the applications.

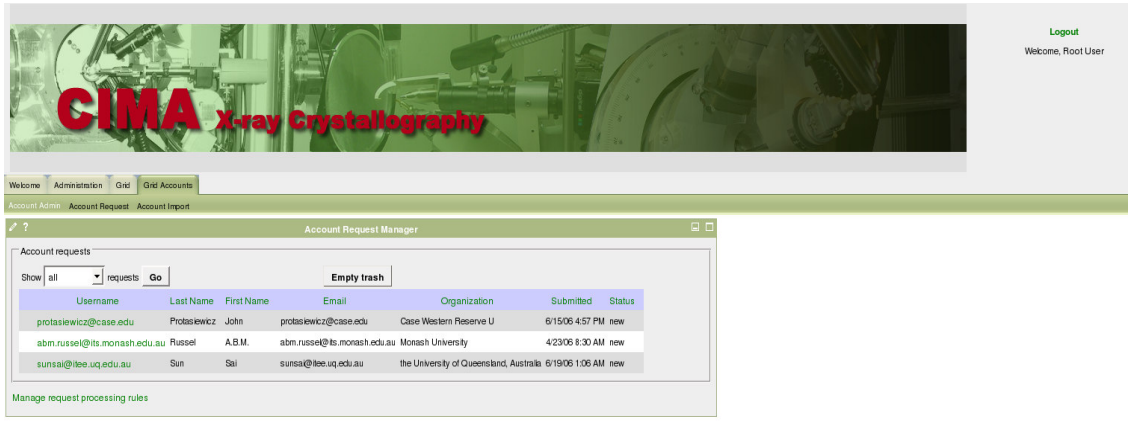
Portals have services for user management, authentication, authorization and groups. These concepts are not new in distributed systems, thus portals mimic these capabilities in higher level. Although portals provide services for user management and limited access to resources, portals interfaces are designed to serve end users in three tier architecture. However, in the middleware the services also require secure accesses to the Grid resources. Thus, account mapping among tiers is essential for Grid portals. There are certain account management techniques comply with these requirements including MyProxy, GAMA and Purse. These technologies are overviewed in the context of authentication.

7.1.1 Authentication

General portal page provide anonymous access to the public just like an open Web page. The first portal page is also gateway to the legitimate users. Authentication usually is provided by using username/password pairs. Credentials also can be used for authentication. After valid authentication, portal services load user's applications and environment settings such as color, style and font types. Although the user context is

setup, accessing to the services may require authentication tokens. Depending on the application type portal server can provide various authentication tokens such as username/password token or Grid credential.

Portal user tokens are mapped to Grid credential in several ways. One of the OGCE methods is to provide a MyProxy portlet [27] that gets proxy credential to the logged in portal users. In this case two-level authentication is required. The user first login to the portal and then gets the proxy credential respectively. During the portal session Grid credential proxy will be alive for two hours. The advantage of this method is that user keeps the Grid access under control. There is no way to portal proxy credential get user's behalf. On the other hand, there are administrative complexities such as users have to deal with two separate accounts, their applications and their maintenance.



July 13, 2007

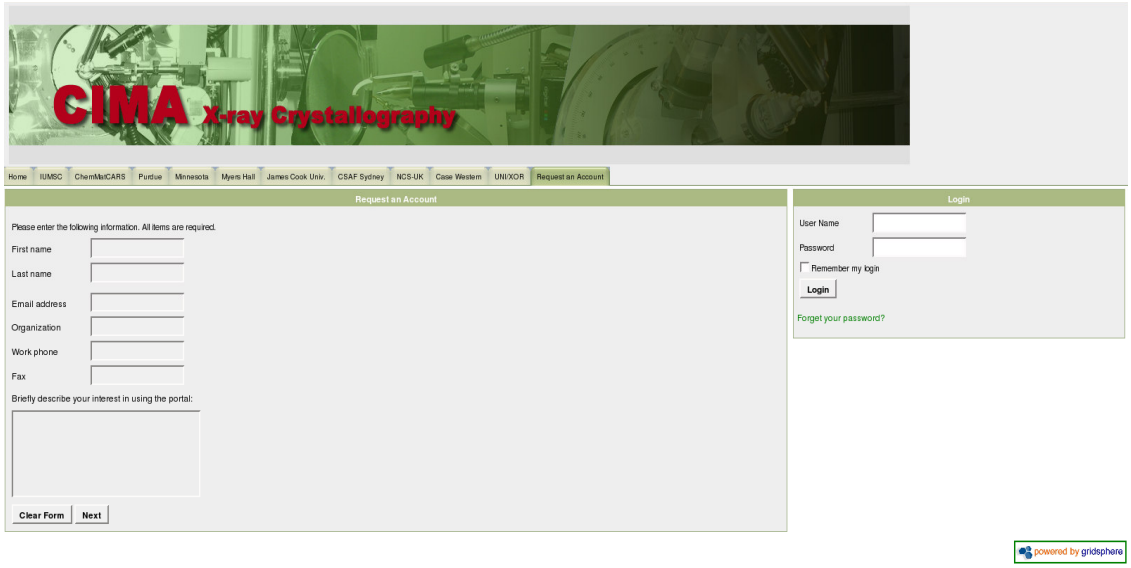


Figure 7.1 Snapshots of GAMA enabled CIMA portal

Grid Account Management Architecture (GAMA) [79] eliminates side effects of the MyProxy portlet by combining Grid accounts and portal accounts in a trusted server in Figure 7.1. Once a user logs in to the portal, the proxy credential will be set and ready to use. Also account creation is relatively easy too. The portal administrator creates portal accounts, generates Grid credentials and store them in the repository. GAMA user interfaces are packaged as JSR 168 portal and runs in GridSphere environment. We preferred to use GAMA in CIMA portal because of its JSR 168 portal support. GAMA is relatively easy for people from unrelated fields who do not need to know about underlying Grid services to issue the proxy credentials.

The Portal-based User Registration System (PURSe) [80] is another option to manage Grid accounts in Web applications. There was no portlet interface of PURSe when we choose to use GAMA. The difference is PURSe does not rely on a trusted

server. PURSe utilizes a certificate authority and MyProxy repository to create and issue Grid credentials. It also accepts existing Grid credentials. OGCE has PURSe portlets [126] that support JSR 168 compatible Grid portals now.

7.1.2 Authorization

Portals use authorization schemas to utilize fine-grained user access rights on the resources. Grid services and legacy applications also use authorization systems at service layer. The issue is to map the different level of authorizations without weakening the security. UNIX authorization framework is well known example of access control and group management on Unix file system.

We have overviewed the services have applied authorization schemas are Shibboleth and PERMIS.

Shibboleth [83]: Shibboleth provides single sign-on among domains while keeping user identities private. Shibboleth services are based on Security Assertion Markup Language (SAML) [84]. Shibboleth has two major parts, Identity provider and Service Provider. An identity provider creates, maintains the user accounts while service provider accesses to the resources. Shibboleth also provide a “Where are you from?” (WAYF) service for third party accesses decisions. WAYF service mediates among identity services of requester and service provider. The positive side of the Shibboleth is to federate portal instances among the organizational boundaries.

PERMIS [87]: Privilege and Role Management Infrastructure Standards Validation (PERMIS) provides role based authorization management among multiple domains. PERMIS uses XML based policies defining rules, specifying access control decisions. Roles are secured by X.509 certificates and stored in Lightweight Directory Access

Protocol (LDAP) repositories. Policies are enforced during service access by Authorization Enforcement Function (AEF) and then the service negotiates with Authorization Decision Function (ADF) to assert policies. The advantage of PERMIS architecture is to fit into role based portal architecture.

7.1.3 Portal Users and Groups

Portal users are defined as entities to access the resources with limited rights. Naturally some users would have access to all resources like super users and some end users can only access to resources with very limited capabilities. Current portal designers generalized the users with the roles. The most common roles are super, admin, user and guest where super and admin most privileged users.

Role Based Access Control (RBAC) [127] defines rules with set of permissions to the users. Users are assigned to a role or combination of roles. Most popular portal containers apply this schema such as Jetspeed, GridSphere and Sakai as essential part of the user management service.

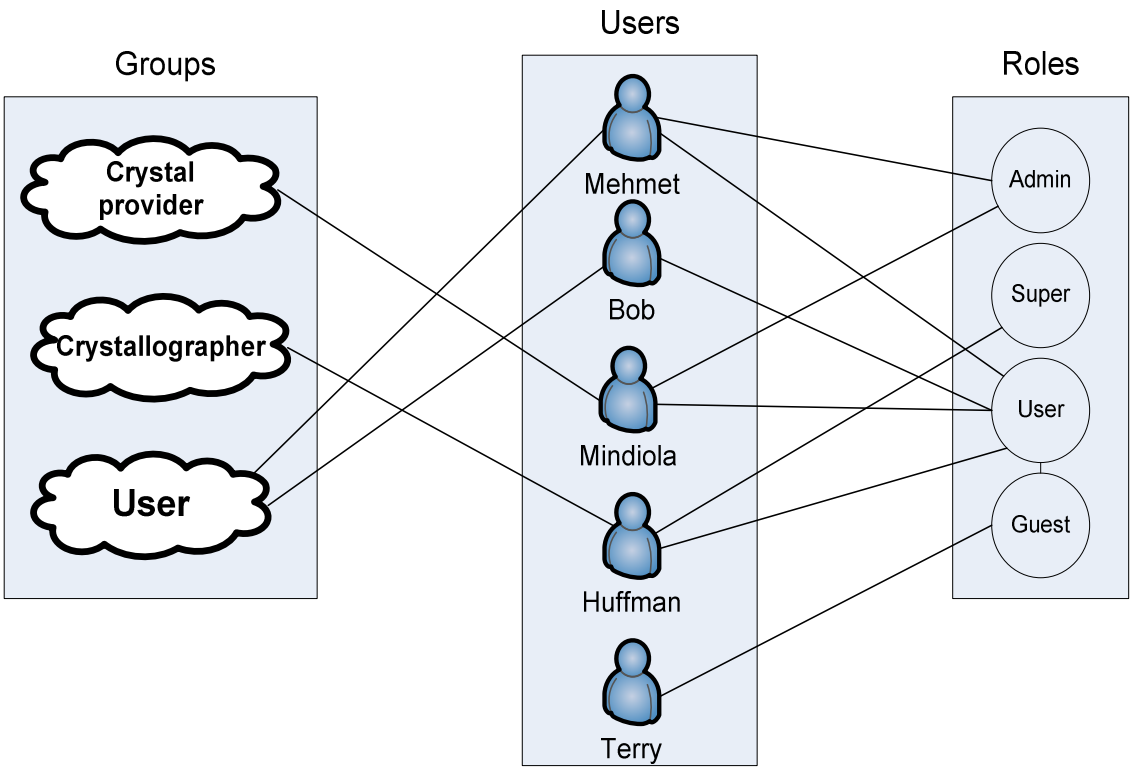


Figure 7.2 Relationships of CIMA portal roles, users and groups

Portal groups are composed of the portlets who are willing to access the same resources for collaboration such groups can be named as crystallographers in case of CIMA portal [47, 92]. Groups are functionally similar to Unix groups. Group member can share the common resources and they will have accessing and modifying rights on the shared resources.

Figure 7.2 depicts the relation of roles, users and groups among CIMA portal. Roles are defined in GridSphere portal framework and they are default values including super, admin, user, and guest. However, roles in GridSphere is fixed and difficult to extend.

Where super owns the portal resources, admin has right to create users and maintain portal resources, users have limited access to the portlets and guest is the anonymous portal page. There are CIMA groups including crystal provider, crystallographer and user. Where crystal provider owns the raw samples, crystallographer analyze and classify the samples, users can see their own samples and the other of samples unless they are private. CIMA users have roles when they created in first place and then admin assign them in relevant groups.

CIMA portal currently uses GridSphere framework and its own specific role, user and group schema within AccessControlManagerService. GridSphere support access control with Role Based Access Control (RBAC) schema. GridSphere RBAC schema has predefined roles. For example, GridSphere comes with four predefined roles:

- Super, admin, user, guest

CIMA Data Manager proposes roles as following:

- Provider, crystallographer, user and guest.

CIMA roles do not map with GridSphere roles, in contrast, GridSphere and its portlet container support users. In order to define the contents mapping to CIMA requirements, we need to describe groups within portlet contents. In contrast to GridSphere that can only group the portlet applications.

CIMA utilized additional database tables to support its own group and roles. CIMA portlets accessed to group tables with usernames and retrieved data from Data Manager. This CIMA specific portlets are only able to run with GridSphere user database. As a result CIMA portlets are tightly coupled with GridSphere.

CIMA portlet access control mechanism is based on a stovepipe solution. It does not apply any policy for portlet contents. CIMA requirements are applied in a hard-coded way. If they need to change the requirements and policies they need to elaborate the mechanism and re-implement the new approach. That is the drawback of CIMA access control mechanism. We need to abstract this approach to solve and adapt this solution.

7.1.4 CIMA Portlets for Partner Labs

CIMA portal provides portlets from each partner lab so that sharing and collaborating on samples of various partner labs. Current implementation of CIMA only allows adding portlets for corresponding partners. There is only one portal instance running at Indiana University Molecular Structure Center (IUMSC) [128] that can access all other resources using the portlets. This approach does not scale well and brings administrative burdens. Each lab should facilitate its own portal server with their administration staff. Shibboleth and PERMIS based authentication and authorization systems will be applied to federate CIMA portals.

7.2 Controlling Access to Grid portlet contents

We have overviewed user management of portal architectures so to narrow down the granularity problem within the portlets. JSR 168 portlet containers are designed to support portlet components as smallest part; additionally JSR 168 does not enforce any authorization mechanisms to portlet contents. JSR 168 is able to describe user names as subject, but there is no associated access control list. In order to solve this problem we have sketched the needs and our solution to portlet access rights and groups. We have

evaluated CIMA portal case in detail and implemented portlet groups on the GridSphere that utilizing CIMA samples.

Table 7.1 Sample features in CIMA portal

Sample No	Crystallographer	Crystal provider	Real Owner	Permission
00001	IUMSC	Dr. Chris	Chris	0
00002	IUMSC	Dr. Tall	Tall	0
00003	IUMSC	Dr. Graph	Graph	1
00004	IUMSC	Dr. CJ	-	-1

Table 7.1 Sample features in CIMA portal shows the sample information that includes sample number, crystallographer, crystal provider, owner and permission. Sample number is uniquely assigned once the provider places a crystal sample to the portal. Permissions to the raw samples are 0 that means sample owner can change metadata of the samples. Once the permission set to 1 then crystal owner cannot change anything with the sample. After that crystallographers can publish these samples as public or private. If it is private, only the crystal provider group that owns the sample can see that sample. Public samples are displayed anonymously.

7.3 Implementation of the CIMA Crystallography Portal[2]

7.3.1 Requirements

For the current work, a subset of requirements relating to user and administrative interaction with data was chosen. These include the following:

- Remote users and in-lab crystallographers must be able to monitor an experiment in progress, including viewing current and previously collected CCD frames and associated relevant environmental and technical parameters;
- All raw data is owned by the lab which performed the experiment and collected the data. In addition to the lab, represented by one or more lab administrators, individual users can view (but not modify or delete) their samples;
- Lab administrators must be able to control sample ownership and visibility;
- Because the notion of when an experiment ends is not clearly defined (*e.g.* experiments may be truncated after the fact or additional frames may be gathered based on evaluations made during a run), lab administrators should be able to set the end time of an experiment;
- Lab administrators must be able to add and remove users to an access control list for a sample;
- Users must be able to view their samples, including all files and sensor readings related to the experiment;
- Some sample data may be provided to the general public for educational or public science awareness purposes;
- Users must be able to view the current status of the lab as a whole;

[2] This section is taken from [92].

- Individual functions that are of general utility should be implemented in a reusable, pluggable, standards-based manner as portlets that can be added or removed by administrators or end-users as appropriate;
- The portlets must interact with a lab's data manager software via Web services calls;
- Users and groups will be managed by the portals container and access to all functions of the portal will be provided by a single sign-on through the portal.

A prototype implementation of the crystallography portal was completed using Jetspeed1 and CGI scripts. Although in the right direction, this implementation did not meet our modularization requirement and so with the ramp-up of the NSF middleware project and the availability of support from the OGCE group, we migrated to GridSphere and JSF-based portlet clients to CIMA services as a fully JSR 168 compliant portal container. This assures a degree of survivability and lateral flexibility to move the science process specific functionality to other containers if the need arises.

The requirements outlined above led us to develop the following portlets:

- A lab overview portlet that provides the current status of a facility and its instruments;
- An administrative *Admin* portlet to support management of sample ownership and other parameters related to individual experiments;
- A *PublicSample* portlet that provides sample data to all portal users and the general public;
- A *UserSample* portlet that shows a logged-in user their samples and other samples on whose access control lists they appear.

PublicSample and *UserSample* portlets also allow users to scan through all data objects in an experiment (Figure 7.3). Per design choice some functions of the portlets are made available as pop-up windows. These portlets provide all of the functionality listed in the requirements above. Extensions to the portal's basic group authorization mechanism provide an access control list associated with each data collection. Scientists can view and modify sample data from their X-ray diffraction crystallography experiments based on their roles in this project and can add users to the access control lists of their data sets. Nothing more than a Web browser is needed to interact with the system.

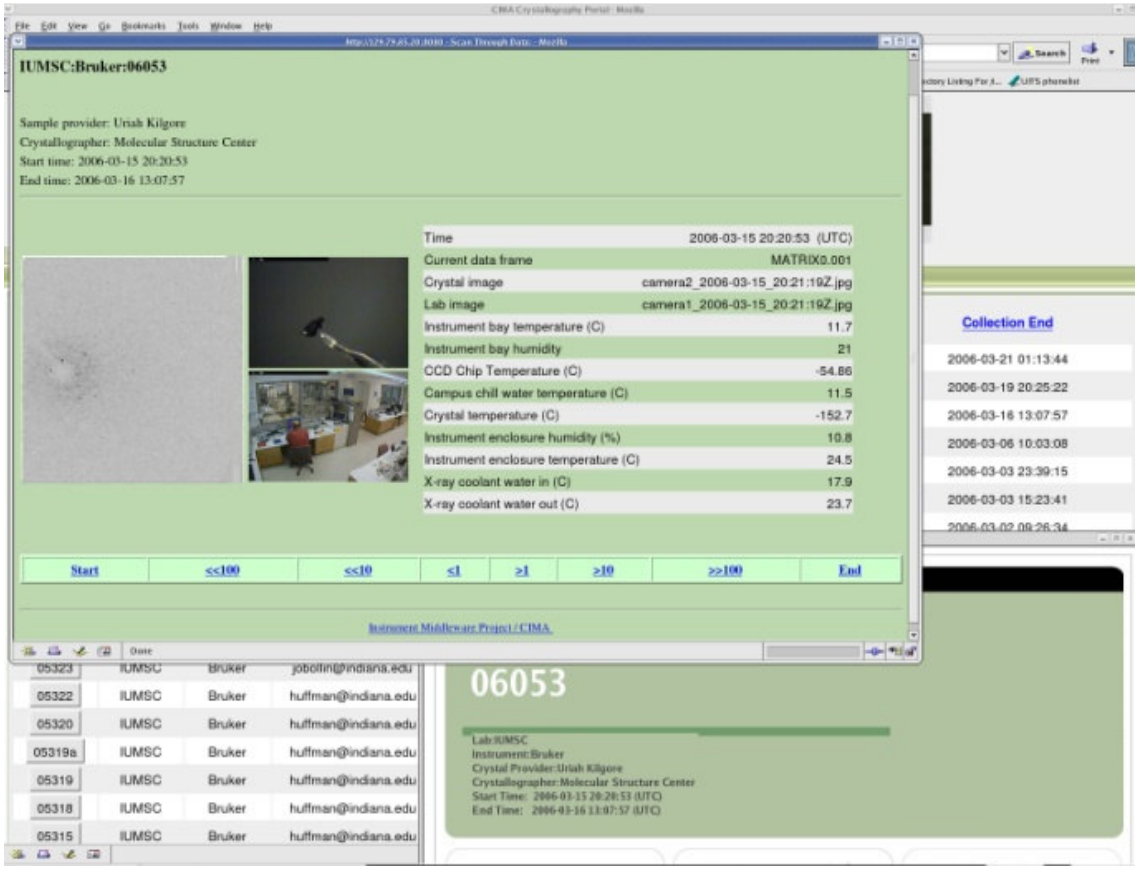


Figure 7.3 UserSample portlet that allows users to stepwise scan through an experiment.

7.3.2 Architecture of the CIMA Crystallography Portal

Developing JSF Web applications includes support for UI components, independent backing JavaBean code, and simplified management of HTTP request and session parameters. UI components handle the interaction with users and communicate with Web services through managed beans for such tasks as access to databases. Then we use portlets to wrap the Web application so that we can make use of the user, group and layout management from the portlet container (GridSphere), and also we can deploy these portlets with respective configuration for different laboratories within the same portal. Furthermore, since the relationship between our portal and data manager is loosely coupled by using Web services, we can effortlessly deploy the portal and Web services at diverse locations.

JSF Web applications handle the UI events and navigation rules to implement the application controller logic. Stubs are generated according to the WSDL of Web services via WSDL2Java tool provided by Apache Axis [76]. This allows access to the Perl-based data manager services. The managed beans in the JSF Web application invoke on the stubs to communicate with Web services and store the data returned by Web services. JSF provides a value binding mechanism to make it easy for Web applications to represent the data combined with managed beans to users.

These beans are populated with information from Web services calls to DM_WS. They are used to set up the model for sample data and related parameters. *WQuerySampleDataBean* and *SampleDataBean* acquire the basic information of sample data, such as sample number, laboratory, instrument and so on. *SampleInfoBean* and *TemperatureBean* obtain environmental conditions related to a specific sample,

like temperature and humidity. *SampleFilesBean* and *FilesBean* inquire CCD frames both in raw and jpeg format pertaining to a specific sample. *SampleCameraBean* and *CameraBean* query camera images of laboratory and crystal during the experimental period.

7.3.3 Identity Mapping between Portal and Data Manager Service

A significant problem faced in the design of the CIMA crystallography portal is the mapping of identities and associated privileges of portal users to identities associated with data sets gathered by the *My Manager* component.

The authorization model used by the portal container is that of users assigned to a limited set of four roles (Super, Admin, User, Guest), whereas the data manager uses an authorization model with users, groups and access control lists that can contain users and groups. Gridsphere does have a notion of groups but this is related to what users can access which portlets rather than a Unix-like notion of a general authorization mechanism for sets of users. Since the portal does not provide a flexible authorization scheme, a design choice was made to perform the mapping of portal users to data owners, groups and access control lists in the logic of the portlets used to access the data manager.

As mentioned above, there are three portlets, *PublicSample* portlet, *UserSample* portlet and *Admin* portlet. For *UserSample* and *Admin* portlets, users can only access the related data according to their roles. Thus, an approach is required to match user identities between portal and data manager service.

Then the *username* can be used to query the database combined with GridSphere to get the information of the user, such as user's full name, email address and groups, etc.

Because the current version of GridSphere doesn't support a hierarchy of groups containing users, we use GridSphere portal groups to control access to portlets that, in turn, control access to data objects. Several GridSphere portal groups are created: *"<lab>_admin"* represents the lab administrators who can access *Admin* portlet and are in charge of the ownership and access control of samples; *"<lab>_client"* represents portal users who are clients from a single lab and can access *UserSample* portlet. Access control lists are implemented as other groups titled as research group names, the members of which can view sample data from the group they belong to (non-public samples). The lab administrator can control the access list for a sample through *Admin* portlet according to users and groups mentioned above. The first two groups (*<lab>_admin* and *<lab>_client*) are used to separate users from different laboratories when there are multiple CIMA crystallography portals deployed in the same container. Gridsphere specific database tables are required to get user information. Finally, the user name and groups information are transferred as parameters to Web services to fetch sample data.

7.4 Summary

In this chapter we have summarized the access control methods of portlets within JSR 168 portlet containers. We have emphasized that fine-grained access control schemas are not described in the JSR 168 specification. The new specification (JSR 286) claims to cover these features. We have implemented and have shown a model of portlet access control through GridSphere portal framework. In general, accesses to the user management database are tightly coupled with GridSphere framework. Nowadays

open source portal community converging to standard solutions of user management. In the future, we can abstract these portlet group capabilities and synchronize our solution to the other portal frameworks.

Chapter 8

Conclusion and Future Works

Web portals have been gaining more attention and usage as the number of online users increases. In addition to current commercial Web portals, computing and Grid portals, which are more specific to their respective topics, will be more important for researchers and academia. A Grid portal would perfectly fit as an interface for Grid services. As such, links to other online applications for batch job execution, massive data transfer and information retrieval.

Despite the fact that the Web portals were initially intended to be used as aggregating content providers, its infrastructure has caused the rapid development of Grid environments through Web portals. The main characteristics of this Grid computing environment are the following:

- File and data management
- Scientific application management

- Access to user-centric metadata services
- Sharing computing and data resources via Grid
- Pervasive security environment to establish authentication and manage access rights to resources.

Grid portal technologies have been dramatically enhanced since last decade. However, instead of fully utilizing all the above important features, many of the initial approaches focused on specific Grid services. Most applications used standard open interfaces of the Grid tied to the middleware architectural paradigms. Grid portals are good representatives of this type of approach.

Since 2002, many science portals and gateways have been based on the reusable portlet component model. Although this model is an important foundation, in practice we have not seen the wide general adoption that was expected. Portlet repositories exist [26], but generally portlets are designed to encapsulate functionality at a coarser level than needed by most science portals. The problems are two-fold:

- Science portals have, for the most part, small communities of specialized users that do not scale. An RSS feed reader portlet (e.g. [129]), for example, would have thousands of feed sources that it could display. These could range from major news services such as Reuters to personal blog feeds. Such a component would be usefully shared among hundreds or thousands of portal installations (supporting campus and corporate intranets, for example). The total number users of all instances of this simple portlet could scale to millions. In contrast the total number of users of PWSCF would number at most in the hundreds or thousands, so a PWSCF portlet in the VLAB project is both much more

sophisticated and designed to support a much smaller community. We may design a reusable PWSCF portlet, but it is unlikely to find much usage by other science gateways than VLAB. Thus the design choices of the portlet API ultimately do not reflect the requirements of the science gateway community, as we discuss below.

- Science gateway portlets such as PWSCF do need to be based on reusable libraries and tools because they are difficult to build. The JSR 168 portlet API is only concerned with managing construction of HTML views and passing incoming HTML form request parameters to the appropriate code logic that must be written by developers. The reusability of the portlet action method's code logic is outside the scope of the portlet API. Likewise, the portlet API assumes portlets themselves are independent, so there is no way to take a collection of pre-developed portlets and express inter-dependencies between them when building a new science gateway.

Our research has taken up where the portlet API leaves off. To address the problems and limitations of the portlet component model for science portals, we have designed a fine-grained component model architecture that is implemented as an XML based, open, and applicable provided by the Grid Tag Libraries and Beans (GTLAB). GTLAB is intended to support Grid services and portal development frameworks, as well as the workflow engines such as Condor DAGMan and Taverna.

Our architecture consists of end user interfaces provided through Grid tag interfaces, Grid bean interfaces and libraries, and metadata repositories. Grid tag interfaces may be used by application developers to quickly assemble new user interfaces to science

applications. Grid beans are capable of communicating with Grid services as well as matching with Grid tags. They also are independent of implementation details. The metadata repositories, for example, may be independently designed either as file system repositories, as traditional relational databases or as service-oriented context repositories, which are connected later through bridges and adapters. The data of these metadata repositories are retrieved to access archives or for resubmitting similar tasks.

In GTLAB, we have designed each Grid service access point as an XML tag widget that corresponds to Grid beans. In this way, we showed an extensible architecture that can enable external Grid beans to GTLAB. In our design, we did not disregard the JSR 168 standard effort but, rather, searched for a way to adopt portlet contents as represented by smaller and modular components. We understand that Grid portlets will be utilizing workflows and component technologies, instead of defining each capability as separate portlet. Not only GTLAB providing modular Grid components, but also it shows a way to integrate and collect them in logical sequences such as directed acyclic graphs. Therefore, the emphasis of our research is to experiment with our approach on different science gateways.

Case studies lead us to identify and develop research issues. For the QuakeSim portal, we have remodeled its portal architecture to use Grid services by invoking QuakeSim applications that reside on TeraGrid nodes as opposed to previous Web service based approach. We also have built VLab portlets as the basis for GTLAB framework by dealing with complicated user interfaces creation and providing their transition to the Grid service clients, as well as managing metadata repositories.

Though GTLAB illustrated a number of important issues as presented throughout this thesis, it continues to be the source of interesting research problems: for example, providing workflow features of conditional branching and loops. As future work, we see that a more general and comprehensive implementation of GTLAB including Web 2.0 features may be necessary.

This future work addresses the issues of applying Grid tags to JavaScript (JS), AJAX or Web 2.0 tools. Grid tags should support client side implementations for browser clients. This will show that GTLAB architecture is portable among various technologies. The important aspect is to convert Grid beans as services that Web 2.0 gadgets and widgets can directly access to them and utilize these services through well known client tools such as AJAX. Security of user credentials should be provided by Web services to JS clients. Multiple job submission management and metadata tracking should be studied.

Appendix A

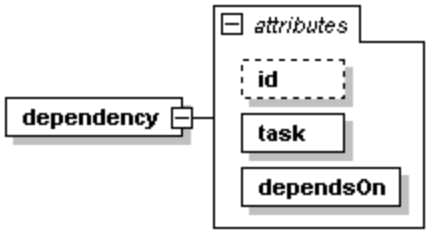
Schema **GridTagsXMLSchema.xsd**

schema location: C:\Documents and Settings\manacar\workspace\GridTagsBeans\GridTagsXMLSchema.xsd
attribute form default:
element form default:
targetNamespace: **http://www.ogce.org/gsf/task**

Elements
dependency
fileoperation
filetransfer
handler
jobsubmit
multitask
myproxy
root
submit

element **dependency**

diagram



namespace http://www.ogce.org/gsf/task

properties
- content complex

used by
- element **multitask**

attributes

Name	Type	Use	Default	Fixed	annotation
id	**xsd:string**	optional			
task	**xsd:string**	required			
dependsOn	**xsd:string**	required			

source

```
<xsd:element name="dependency">
  <xsd:complexType>
    <xsd:attribute name="id" type="xsd:string" use="optional"/>
    <xsd:attribute name="task" type="xsd:string" use="required"/>
    <xsd:attribute name="dependsOn" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:element>
```

attribute **dependency/@id**

type **xsd:string**

properties
isRef 0
use optional

source
```
<xsd:attribute name="id" type="xsd:string" use="optional"/>
```

attribute **dependency/@task**

type **xsd:string**

properties
isRef 0
use required

source
```
<xsd:attribute name="task" type="xsd:string" use="required"/>
```

attribute **dependency/@dependsOn**

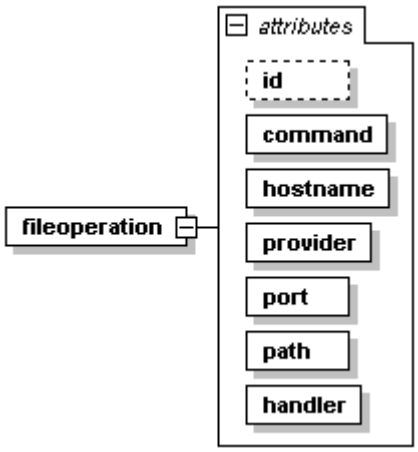
type **xsd:string**

properties
isRef 0
use required

source
```
<xsd:attribute name="dependsOn" type="xsd:string" use="required"/>
```

element **fileoperation**

diagram



namespace http://www.ogce.org/gsf/task

properties
content complex

used by
element **multitask**

attributes

Name	Type	Use	Default	Fixed	annotation
id	**xsd:string**	optional			
command	**xsd:string**	required			
hostname	**xsd:string**	required			
provider	**xsd:string**	required			
port	**xsd:string**	required			
path	**xsd:string**	required			
handler	**xsd:string**	required			

source
```
<xsd:element name="fileoperation">
 <xsd:complexType>
  <xsd:attribute name="id" type="xsd:string" use="optional"/>
  <xsd:attribute name="command" type="xsd:string" use="required"/>
  <xsd:attribute name="hostname" type="xsd:string" use="required"/>
  <xsd:attribute name="provider" type="xsd:string" use="required"/>
  <xsd:attribute name="port" type="xsd:string" use="required"/>
```

```
    <xsd:attribute name="path" type="xsd:string" use="required"/>
    <xsd:attribute name="handler" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:element>
```

attribute **fileoperation/@id**

type	**xsd:string**
properties	isRef 0 use optional
source	<xsd:attribute name="id" type="xsd:string" use="optional"/>

attribute **fileoperation/@command**

type	**xsd:string**
properties	isRef 0 use required
source	<xsd:attribute name="command" type="xsd:string" use="required"/>

attribute **fileoperation/@hostname**

type	**xsd:string**
properties	isRef 0 use required
source	<xsd:attribute name="hostname" type="xsd:string" use="required"/>

attribute **fileoperation/@provider**

type	**xsd:string**
properties	isRef 0 use required
source	<xsd:attribute name="provider" type="xsd:string" use="required"/>

attribute **fileoperation/@port**

type	**xsd:string**
properties	isRef 0 use required
source	<xsd:attribute name="port" type="xsd:string" use="required"/>

attribute **fileoperation/@path**

type	**xsd:string**
properties	isRef 0 use required
source	<xsd:attribute name="path" type="xsd:string" use="required"/>

attribute **fileoperation/@handler**

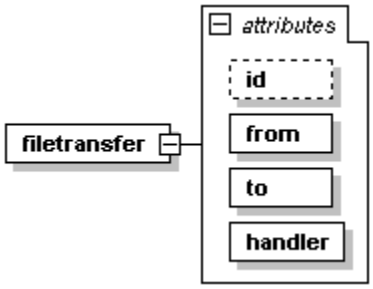
type	**xsd:string**

properties isRef 0
use required

source
```
<xsd:attribute name="handler" type="xsd:string" use="required"/>
```

element **filetransfer**

diagram



namespace http://www.ogce.org/gsf/task

properties content complex

used by element **multitask**

attributes

Name	Type	Use	Default	Fixed	annotation
id	**xsd:string**	optional			
from	**xsd:string**	required			
to	**xsd:string**	required			
handler	**xsd:string**	required			

source
```
<xsd:element name="filetransfer">
  <xsd:complexType>
    <xsd:attribute name="id" type="xsd:string" use="optional"/>
    <xsd:attribute name="from" type="xsd:string" use="required"/>
    <xsd:attribute name="to" type="xsd:string" use="required"/>
    <xsd:attribute name="handler" type="xsd:string" use="required"/>
  </xsd:complexType>
</xsd:element>
```

attribute **filetransfer/@id**

type **xsd:string**

properties isRef 0
use optional

source
```
<xsd:attribute name="id" type="xsd:string" use="optional"/>
```

attribute **filetransfer/@from**

type **xsd:string**

properties isRef 0
use required

source
```
<xsd:attribute name="from" type="xsd:string" use="required"/>
```

attribute **filetransfer/@to**

type **xsd:string**

properties isRef 0
use required

source
```
<xsd:attribute name="to" type="xsd:string" use="required"/>
```

attribute **filetransfer/@handler**

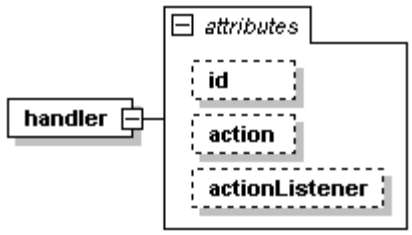
type **xsd:string**

properties
isRef 0
use required

source
```
<xsd:attribute name="handler" type="xsd:string" use="required"/>
```

element **handler**

diagram



namespace http://www.ogce.org/gsf/task

properties
content complex

attributes

Name	Type	Use	Default	Fixed	annotation
id	**xsd:string**	optional			
action	**xsd:string**				
actionListener	**xsd:string**				

source
```
<xsd:element name="handler">
  <xsd:complexType>
    <xsd:attribute name="id" type="xsd:string" use="optional"/>
    <xsd:attribute name="action" type="xsd:string"/>
    <xsd:attribute name="actionListener" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

attribute **handler/@id**

type **xsd:string**

properties
isRef 0
use optional

source
```
<xsd:attribute name="id" type="xsd:string" use="optional"/>
```

attribute **handler/@action**

type **xsd:string**

properties
isRef 0

source
```
<xsd:attribute name="action" type="xsd:string"/>
```

attribute **handler/@actionListener**

type **xsd:string**

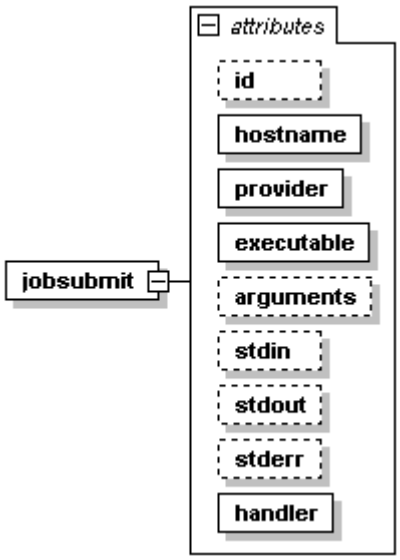
properties
isRef 0

source

```
<xsd:attribute name="actionListener" type="xsd:string"/>
```

element **jobsubmit**

diagram



namespace http://www.ogce.org/gsf/task

properties
- content complex

used by
- element **multitask**

attributes

Name	Type	Use	Default	Fixed	annotation
id	**xsd:string**	optional			
hostname	**xsd:string**	required			
provider	**xsd:string**	required			
executable	**xsd:string**	required			
arguments	**xsd:string**	optional			
stdin	**xsd:string**	optional			
stdout	**xsd:string**	optional			
stderr	**xsd:string**	optional			
handler	**xsd:string**	required			

source

```
<xsd:element name="jobsubmit">
 <xsd:complexType>
  <xsd:attribute name="id" type="xsd:string" use="optional"/>
  <xsd:attribute name="hostname" type="xsd:string" use="required"/>
  <xsd:attribute name="provider" type="xsd:string" use="required"/>
  <xsd:attribute name="executable" type="xsd:string" use="required"/>
  <xsd:attribute name="arguments" type="xsd:string" use="optional"/>
  <xsd:attribute name="stdin" type="xsd:string" use="optional"/>
  <xsd:attribute name="stdout" type="xsd:string" use="optional"/>
  <xsd:attribute name="stderr" type="xsd:string" use="optional"/>
  <xsd:attribute name="handler" type="xsd:string" use="required"/>
 </xsd:complexType>
</xsd:element>
```

attribute **jobsubmit/@id**

type **xsd:string**

properties
- isRef 0
- use optional

source <xsd:attribute name="id" type="xsd:string" use="optional"/>

attribute **jobsubmit/@hostname**

type **xsd:string**

properties isRef 0
use required

source <xsd:attribute name="hostname" type="xsd:string" use="required"/>

attribute **jobsubmit/@provider**

type **xsd:string**

properties isRef 0
use required

source <xsd:attribute name="provider" type="xsd:string" use="required"/>

attribute **jobsubmit/@executable**

type **xsd:string**

properties isRef 0
use required

source <xsd:attribute name="executable" type="xsd:string" use="required"/>

attribute **jobsubmit/@arguments**

type **xsd:string**

properties isRef 0
use optional

source <xsd:attribute name="arguments" type="xsd:string" use="optional"/>

attribute **jobsubmit/@stdin**

type **xsd:string**

properties isRef 0
use optional

source <xsd:attribute name="stdin" type="xsd:string" use="optional"/>

attribute **jobsubmit/@stdout**

type **xsd:string**

properties isRef 0
use optional

source <xsd:attribute name="stdout" type="xsd:string" use="optional"/>

attribute **jobsubmit/@stderr**

type **xsd:string**

properties isRef 0
use optional

source <xsd:attribute name="stderr" type="xsd:string" use="optional"/>

attribute **jobsubmit/@handler**

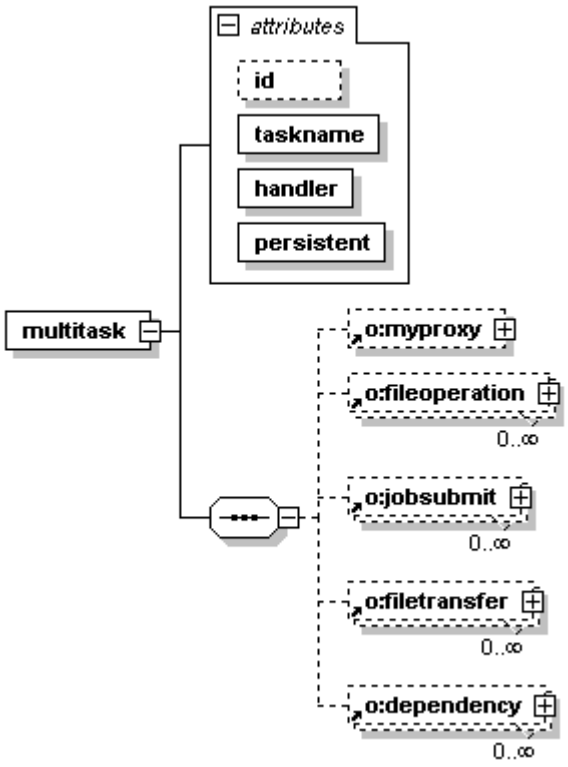
type **xsd:string**

properties
- isRef 0
- use required

source
```
<xsd:attribute name="handler" type="xsd:string" use="required"/>
```

element **multitask**

diagram



namespace http://www.ogce.org/gsf/task

properties
- content complex

children **o:myproxy** **o:fileoperation** **o:jobsubmit** **o:filetransfer** **o:dependency**

used by
- element **submit**

attributes

Name	Type	Use	Default	Fixed	annotation
id	**xsd:string**	optional			
taskname	**xsd:string**	required			
handler	**xsd:string**	required			
persistent	**xsd:boolean**	required			

source
```
<xsd:element name="multitask">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="o:myproxy" minOccurs="0"/>
   <xsd:element ref="o:fileoperation" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o:jobsubmit" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o:filetransfer" minOccurs="0" maxOccurs="unbounded"/>
   <xsd:element ref="o:dependency" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
```

```
    <xsd:attribute name="id" type="xsd:string" use="optional"/>
    <xsd:attribute name="taskname" type="xsd:string" use="required"/>
    <xsd:attribute name="handler" type="xsd:string" use="required"/>
    <xsd:attribute name="persistent" type="xsd:boolean" use="required"/>
   </xsd:complexType>
  </xsd:element>
```

attribute **multitask/@id**

type	**xsd:string**
properties	isRef 0 use optional
source	<xsd:attribute name="id" type="xsd:string" use="optional"/>

attribute **multitask/@taskname**

type	**xsd:string**
properties	isRef 0 use required
source	<xsd:attribute name="taskname" type="xsd:string" use="required"/>

attribute **multitask/@handler**

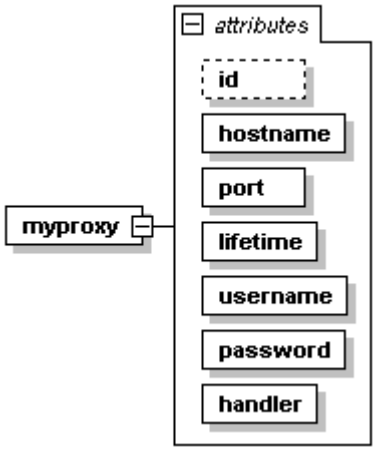
type	**xsd:string**
properties	isRef 0 use required
source	<xsd:attribute name="handler" type="xsd:string" use="required"/>

attribute **multitask/@persistent**

type	**xsd:boolean**
properties	isRef 0 use required
source	<xsd:attribute name="persistent" type="xsd:boolean" use="required"/>

element **myproxy**

diagram



namespace http://www.ogce.org/gsf/task

properties content complex

used by element **multitask**

attributes

Name	Type	Use	Default	Fixed	annotation
id	**xsd:string**	optional			
hostname	**xsd:string**	required			
port	**xsd:string**	required			
lifetime	**xsd:string**	required			
username	**xsd:string**	required			
password	**xsd:string**	required			
handler	**xsd:string**	required			

source

```
<xsd:element name="myproxy">
 <xsd:complexType>
  <xsd:attribute name="id" type="xsd:string" use="optional"/>
  <xsd:attribute name="hostname" type="xsd:string" use="required"/>
  <xsd:attribute name="port" type="xsd:string" use="required"/>
  <xsd:attribute name="lifetime" type="xsd:string" use="required"/>
  <xsd:attribute name="username" type="xsd:string" use="required"/>
  <xsd:attribute name="password" type="xsd:string" use="required"/>
  <xsd:attribute name="handler" type="xsd:string" use="required"/>
 </xsd:complexType>
</xsd:element>
```

attribute **myproxy**/@id

type **xsd:string**

properties isRef 0
use optional

source

```
<xsd:attribute name="id" type="xsd:string" use="optional"/>
```

attribute **myproxy**/@hostname

type **xsd:string**

properties isRef 0
use required

source

```
<xsd:attribute name="hostname" type="xsd:string" use="required"/>
```

attribute **myproxy**/@port

type **xsd:string**

properties isRef 0
use required

source

```
<xsd:attribute name="port" type="xsd:string" use="required"/>
```

attribute **myproxy**/@lifetime

type **xsd:string**

properties isRef 0
use required

source

```
<xsd:attribute name="lifetime" type="xsd:string" use="required"/>
```

attribute **myproxy/@username**

type **xsd:string**

properties
isRef 0
use required

source
```
<xsd:attribute name="username" type="xsd:string" use="required"/>
```

attribute **myproxy/@password**

type **xsd:string**

properties
isRef 0
use required

source
```
<xsd:attribute name="password" type="xsd:string" use="required"/>
```

attribute **myproxy/@handler**

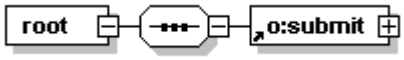
type **xsd:string**

properties
isRef 0
use required

source
```
<xsd:attribute name="handler" type="xsd:string" use="required"/>
```

element **root**

diagram



namespace http://www.ogce.org/gsf/task

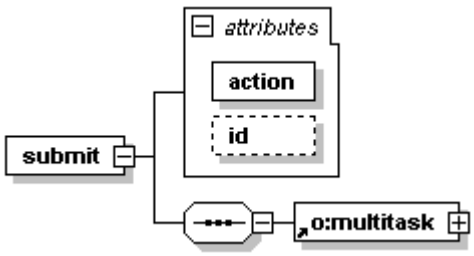
properties
content complex

children **o:submit**

source
```
<xsd:element name="root">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="o:submit"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

element **submit**

diagram



namespace http://www.ogce.org/gsf/task

properties
content complex

children **o:multitask**

used by
element **root**

attributes

Name	Type	Use	Default	Fixed	annotation
action	**xsd:string**	required			
id	**xsd:string**	optional			

source

```
<xsd:element name="submit">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element ref="o:multitask"/>
  </xsd:sequence>
  <xsd:attribute name="action" type="xsd:string" use="required"/>
  <xsd:attribute name="id" type="xsd:string" use="optional"/>
 </xsd:complexType>
</xsd:element>
```

attribute **submit/@action**

type **xsd:string**

properties
isRef 0
use required

source

```
<xsd:attribute name="action" type="xsd:string" use="required"/>
```

attribute **submit/@id**

type **xsd:string**

properties
isRef 0
use optional

source

```
<xsd:attribute name="id" type="xsd:string" use="optional"/>
```

Appendix B

Table 1. Attributes to dependency

Attribute name	Required	Description
id	yes	String: component id
task	yes	String: task component name
dependsOn	yes	String: task component that depends on

Table 2. Attributes to myproxy

Attribute name	Required	Description
id	yes	String: component id
hostname	yes	String: myproxy server name
port	yes	String: myproxy port number (default is 7512)
lifetime	yes	String: myproxy lifetime (default is 2 hours)
Username	yes	String: user name for stored credential
password	yes	String: password for stored credential
handler	no	String: defines bean method to submit this component

Table 3. Attributes to multitask

Attribute name	Required	Description
id	yes	String: component id
taskname	yes	String: task name is for multitask
persistent	no	String: stores task information
handler	no	String: defines bean method to submit this component

Table 4. Attributes to fileoperation

Attribute name	Required	Description
id	yes	String: component id
hostname	yes	String: gridftp server name
port	yes	String: gridftp port number (default is 2811)
provider	yes	String: gridftp provider name (default is ‘gridftp’)
path	yes	String: directory location on the local file system
command	yes	String: file operation (e.g. ls, mkdir)
handler	no	String: defines bean method to submit this component

Table 5. Attributes to jobsubmit

Attribute name	Required	Description
id	yes	String: component id
hostname	yes	String: GRAM server name
provider	yes	String: GT provider name (default is 'GT2')
executable	yes	String: executable command or script name
arguments	yes	String: arguments of the command or the script
stdin	yes	String: standard input file name for the command
stdout	yes	String: standard output file name for the command
stderr	yes	String: standard error file name for the command
handler	no	String: defines bean method to submit this component

Table 6. Attributes to filetransfer

Attribute name	Required	Description
id	yes	String: component id
from	yes	String: file location on source gridftp server
to	yes	String: file location on target gridftp server
handler	no	String: defines bean method to submit this component

Table 7. Attributes to submit

Attribute name	Required	Description
id	yes	String: component id
action	yes	String: defines bean method to submit this component
actionListener	yes	String: defines action listener method

Table 8. Attributes to handler

Attribute name	Required	Description
id	yes	String: component id
action	yes	String: defines bean method to submit this component
actionListener	yes	String: defines action listener method

Bibliography

[1] *National e-Science centre for UK*. [cited; Available from: http://www.nesc.ac.uk/.

[2] Atkinson, M., et al., *Web Service Grids: an evolutionary approach.* Concurrency and Computation: Practice and Experience, 2005. **17**(2-4): p. 377-389.

[3] Foster, I., et al., *The Physiology of the Grid*, in *Grid Computing*, G.F.T.H. Fran Berman, Editor. 2003. p. 217-249.

[4] *TeraGrid*. [cited; Available from: http://www.teragrid.org/.

[5] *Open Science Grid*. [cited; Available from: http://www.opensciencegrid.org/.

[6] *Enabling Grids for E-sciencE (EGEE)*. [cited; Available from: http://www.eu-egee.org/.

[7] *Coordinated TeraGrid Software and Services (CTSS)*. [cited; Available from: http://www.teragrid.org/userinfo/software/ctss.php.

[8] *The Virtual Data Toolkit*. [cited; Available from: http://vdt.cs.wisc.edu/.

[9] *gLite* [cited; Available from: http://glite.web.cern.ch/glite/default.asp.

[10] Fox, G., D. Gannon, and M. Thomas, *Special Issue: Grid Computing Environments.* Concurrency and Computation: Practice and Experience, 2002. **14**(13-15): p. 1035-1044.

[11] Wilkins-Diehr, N., *Special Issue: Science Gateways - Common Community Interfaces to Grid Resources.* Concurrency and Computation: Practice and Experience, 2006. **19**(6): p. 743-749.

[12] Droegemeier, K.K., et al., *Linked environments for atmospheric discovery (LEAD): A cyberinfrastructure for mesoscale meteorology research and education.* 20th Conf. on Interactive Information Processing Systems for Meteorology, Oceanography, and Hydrology, 2004.

[13] Ewa, D., et al., *Grid-Based Galaxy Morphology Analysis for the National Virtual Observatory*, in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*. 2003, IEEE Computer Society.

[14] *QuakeSim portal*. [cited; Available from: http://quakesim.jpl.nasa.gov/.

[15] Nacar, M.A., et al., *VLab: Collaborative Grid Services and Portals to Support Computational Material Science* Concurrency and Computation: Practice and Experience, 2007. **19**(12): p. 1717-1728.

[16] Abdelnur, A., E. Chien, and S. and Hepper, (eds.) *Portlet Specification 1.0*. 2003 [cited; Available from: http://www.jcp.org/en/jsr/detail?id=168.

[17] Novotny, J., M. Russell, and O. Wehrens, *GridSphere: a portal framework for building collaborations.* Concurrency - Practice and Experience, 2004. **16**(5): p. 503-513.

[18] *GridSphere*. [cited; Available from: http://www.gridsphere.org.

[19] *uPortal* [cited; Available from: http://www.uportal.org/.

[20] *Pluto*. [cited; Available from: http://portals.apache.org/pluto/.

[21] *Jetspeed*. [cited; Available from: http://portals.apache.org/jetspeed-1/.

[22] *Sakai*. [cited; Available from: http://sakaiproject.org/.

[23] *Liferay*. [cited; Available from: http://www.liferay.com/web/guest/home.

[24] *JBoss*. [cited; Available from: http://labs.jboss.com/.

[25] *Exo Portal*. [cited; Available from: http://www.exoplatform.com/portal/public/site/.

[26] *OGCE web site*. [cited; Available from: http://www.collab-ogce.org/ogce2/.

[27] Alameda, J., et al., *The Open Grid Computing Environments collaboration: portlets and services for science gateways.* Concurrency and Computation: Practice and Experience, 2007. **19**(6): p. 22.

[28] Russell, M., J. Novotny, and O. Wehrens, *The Grid Portlets Web Application: A Grid Portal Framework*. Parallel Processing and Applied Mathematics. 2006. 691-698.

[29] von Laszewski, G., et al., *A Java commodity grid kit.* Concurrency and Computation Practice and Experience, 2001. **13**(8-9): p. 645-662.

[30] Johnson, R., *Expert One-On-One J2EE Design and Development*. Wrox. 2003, Indianapolis, IN, USA: Wiley Publishing, Inc.

[31] Singh, I., *Designing Enterprise Applications with the J2ee (tm) Platform*. 2002: Addison-Wesley Professional.

[32] Gannon, D., et al., *Grid portals: A scientist's access point for grid services (draft 1).* Global Grid Forum, 2003.

[33] *Globus toolkit*. [cited; Available from: http://www.globus.org.

[34] Thain, D., T. Tannenbaum, and M. Livny, *Condor and the Grid*, in *Grid Computing: Making The Global Infrastructure a Reality*, A.J.G.H. Fran Berman, Geoffrey Fox, Editor. 2003, John Wiley.

[35] Erwin, D.W., *UNICORE—a Grid computing environment.* Concurrency and Computation: Practice and Experience, 2002. **14**(13-15): p. 1395-1410.

[36] Sirvent, R., et al., *GRID superscalar and SAGA: forming a high-level and platform-independent Grid programming environment.* CoreGRID Integration WorkShop, 2005. **2005**.

[37] Russell, M. *Vine project*. [cited; Available from: http://gforge.man.poznan.pl/gf/project/vine/.

[38] Nacar, M.A., et al., *Building QuakeSim Portlets with GTLAB*, in *GCE 07 at SC 07*. 2007: Reno, NV.

[39] Fox, G.C. and D. Gannon, *Special Issue: Workflow in Grid Systems.* Concurrency and Computation: Practice and Experience, 2006. **18**(10): p. 1009-1019.

[40] *Java Server Pages (JSP).* [cited; Available from: http://java.sun.com/products/jsp/.

[41] *Java Server Faces (JSF).* [cited; Available from: http://java.sun.com/javaee/javaserverfaces/.

[42] McClanahan, C., E. Burns, and R. Kitain, *Java Server Faces Specification. Version 1.1*.

[43] *Extensible Markup Language (XML).* [cited; Available from: Extensible Markup Language (XML).

[44] *Java Servlet Technology*. [cited; Available from: http://java.sun.com/products/servlet/.

[45] Oinn, T., et al., *Taverna: lessons in creating a workflow environment for the life sciences.* Concurrency and Computation: Practice and Experience, 2006. **18**(10): p. 1067-1100.

[46] Goble, C.A. and D.C. De Roure, *myExperiment: social networking for workflow-using e-scientists*, in *Proceedings of the 2nd workshop on Workflows in support of large-scale science*. 2007, ACM: Monterey, California, USA.

[47] *CIMA portal*. [cited; Available from: http://cimaportal.indiana.edu:8080/gridsphere.

[48] Nacar, M., et al., *Designing Grid Tag Libraries and Grid Beans*, in *Second International Workshop on Grid Computing Environments GCE06 at SC06*. 2006: Tampa, FL.

[49] Okada, Y., *Surface Deformation Due to Shear and Tensile Faults in a Half-Space*. BSSA, 1985. **75**(4): p. 1135-1154.

[50] Parker, J.W., Donnellan, A., Lyzenga, G., Rundle, J.B., and Tullis, T. *Performance Modeling Codes for the QuakeSim Problem Solving Environment*. in *Proceedings of the International Conference on Computational Science (Part III)*. 2003: Springer-Verlag, Berlin.

[51] Booth, D., et al., *Web Service Architecture.* W3C Working Group Note, 2004. **11**.

[52] Nacar, M.A., M. Pierce, and G.C. Fox, *Developing a secure grid computing environment shell engine: containers and services.* Neural, Parallel & Scientific Computations, 2004. **12**(3): p. 379-390.

[53] Kernighan, B.W. and R. Pike, *The UNIX Programming Environment*. 1984: Prentice Hall Professional Technical Reference.

[54] Novotny, J., *Grid Portal Development Toolkit (GPDK). Accepted for publication in.* Concurrency and Computation: Practice and Experience, Special Edition on Grid Computing Environments, 2002. **14**(13-15): p. 1129-1144.

[55] Pierce, M.E., C. Youn, and G.C. Fox, *The Gateway computational Web portal.* Concurrency and Computation: Practice and Experience, 2002. **14**(13-15): p. 1411-1426.

[56] Karl, C., et al., *A Resource Management Architecture for Metacomputing Systems*. : Job Scheduling Strategies for Parallel Processing. 1998. 62.

[57] Allcock, B., et al. *Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing*. in *IEEE Mass Storage Conference*. 2001.

[58] Foster, I., *A Globus Toolkit Primer.* 2005.

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

[60] Czajkowski, K., Ferguson, D., Foster, I., Frey, J.,, S. Graham, Sedukhin, I., Snelling, D., Tuecke, S.,, and W. Vambenepe, *The WS-Resource Framework*. 2004.

[61] Foster, I., C. Kesselman, and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* International Journal of High Performance Computing Applications, 2001. **15**(3): p. 200-222.

[62] Kaizar, A., et al. *Abstracting the Grid*. in *Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing*. 2004.

[63] Scandolo, S., et al., *First-principles codes for computational crystallography in the Quantum-ESPRESSO package.* Zeitschrift für Kristallographie, 2005. **vol:220**: p. 574-579.

[64] *iGoogle*. [cited; Available from: http://www.google.com/ig.

[65] *Netvibes*. [cited; Available from: http://www.netvibes.com/.

[66] Laszewski, G.v., et al., *A Java commodity grid kit.* Concurrency and Computation: Practice and Experience, 2001. **13**(8-9): p. 645-662.

[67] Chapman, C., et al. *Condor Birdbath: Web Service interfaces to condor*. in *Proc. UK e-Science All Hands Meeting*. 2005. Nottingham UK.

[68] Box, D., et al., *Simple Object Access Protocol (SOAP) 1.1*. 2000, May.

[69] Frey, J., et al., *Condor-G: A Computation Management Agent for Multi-Institutional Grids.* Cluster Computing, 2002. **5**(3): p. 237-246.

[70] Barrett, D.J. and R.E. Silverman, *SSH, the Secure Shell: The Definitive Guide*. 2001: O'Reilly.

[71] Dahan, M. and E. Roberts, *TeraGrid User Portal v1.0: Architecture, Design, and Technologies*, in *Second International Workshop on Grid Computing Environments GCE06 at SC06*. 2006: Tampa, FL.

[72] *OASIS: Organization for the Advancement of Structured Information Standards*. [cited; Available from: http://www.oasis-open.org/home/index.php.

[73] Ian, F., et al., *A security architecture for computational grids*, in *Proceedings of the 5th ACM conference on Computer and communications security*. 1998, ACM Press: San Francisco, California, United States.

[74] Christensen, E., et al., *Web Services Description Language (WSDL) 1.1*. 2001.

[75] Uddi, O.R.G., *Universal Description Discovery and Integration (UDDI)*. 2002.

[76] *Apache Axis*. [cited; Available from: http://ws.apache.org/axis

[77] Mestrallet, B., *Exo: New Open Source JSR 168 Compliant Portal and More*.

[78] Kropp, A., C. Leue, and R. Thompson. *Web Services for Remote Portlets (WSRP)*. [cited; Available from: http://www.oasis-open.org.

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

[80] *PURSe*. [cited; Available from: http://www.grids-center.org/solutions/purse/.

[81] *GridShib*. [cited; Available from: http://gridshib.globus.org/.

[82] Barton, T., et al., *Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, GridShib, and MyProxy*. 5th Annual PKI R&D Workshop, April, 2006.

[83] *Shibboleth*. [cited; Available from: http://shibboleth.internet2.edu/.

[84] Scavo, T. and S. Cantor, *Shibboleth Architecture: Technical Overview*. Working Draft. **1**.

[85] Hughes, J. and E. Maler, *Technical Overview of the OASIS Security Assertion Markup Language (SAML) V1. 1*, in *OASIS, May*. 2004.

[86] *Central Authentication Service (CAS)*. [cited; Available from: http://www.ja-sig.org/products/cas/.

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

[88] *World Wide Web Consortium (W3C)*. [cited; Available from: http://www.w3.org/.

[89] Bodoff, S., *The J2EE Tutorial*. 2002: Addison-Wesley Professional.

[90] *Apache portal bridges*. [cited; Available from: http://portals.apache.org/bridges/.

[91] *MyFaces*. [cited; Available from: http://myfaces.apache.org/.

[92] Yin, H., et al., *Providing Portlet-Based Client Access to CIMA-Enabled Crystallographic Instruments, Sensors, and Data*, in *7th IEEE/ACM International Conference on Grid Computing (GRID 2006)*. 2006: Barcelona, Spain.

[93] Nacar, M.A., et al. *Building a Grid Portal for Teragrid's Big Red*. in *TeraGrid 2007*. 2007. Madison, WI.

[94] Bollig, E.F., et al., *VLAB: Web Services, Portlets, and Workflows for Enabling Cyber-infrastructure in Computational Mineral Physics.* Physics of The Earth and Planetary Interiors, 2007. **163**(1-4): p. 333-346.

[95] Pallickara, S. and G. Fox, *NaradaBrokering: A Distributed Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids*. Middleware 2003. 2003. 41-61.

[96] Donnellan, A., et al., *QuakeSim and the Solid Earth Research Virtual Observatory.* Pure and Applied Geophysics, 2006. **163**(11): p. 2263-2279.

[97] *Apache Ant*. [cited; Available from: http://ant.apache.org/.

[98] McMullen, D.F. and K. Huffman, *Connecting Users to Instruments and Sensors: Portals as Multi-user GUIs for Instrument and Sensor Facilities*

Concurrency and Computation: Practice and Experience, 2007(Special issue on Grid portals).

[99] McMullen, D., T. Devadithya, and K. Chiu. *Integrating Instruments and Sensors into the Grid with CIMA Web Services*. in *Proc. of 3rd APAC Conference on Advanced Computing, Grid Applications and e-Research (APAC'05)*. 2005.

[100] Devadithya, T., et al. *The Common Instrument Middleware Architecture: Overview of Goals and Implementation*. in *e-Science and Grid Computing, First International Conference on e-Science and Grid Computing (e-Science 2005)*. 2005.

[101] Giacovazzo, C., *Fundamentals of Crystallography*. 2002: Oxford Univ Pr.

[102] *SAINT*. [cited; Available from: http://xray.utmb.edu/saint.html.

[103] *The MEME/MAST System*. [cited; Available from: http://meme.sdsc.edu/meme/intro.html.

[104] Oinn, T., et al., *Taverna: a tool for the composition and enactment of bioinformatics workflows.* Bioinformatics, 2004. **20**(17): p. 3045-3054.

[105] *How to write your own JSF components*. [cited; Available from: http://www.exadel.com/tutorial/jsf/HowToWriteYourOwnJSFComponents.pdf.

[106] *GTLAB Schemas*. [cited; Available from: http://grids.ucs.indiana.edu/users/manacar/GridTags/GridTagsInterface/GridTagsXMLSchema.xsd.

[107] Tanenbaum, A.S. and M.v. Steen, *Distributed Systems: Principles and Paradigms*. 2002: Prentice Hall. 803.

[108] Rajasekar, A., et al., *Storage Resource Broker-Managing Distributed Data in a Grid.* Computer Society of India Journal, Special Issue on SAN, 2003. **33**(4): p. 42-54.

[109] Schwidder, J., T. Talbott, and J. Myers, *Bootstrapping to a semantic grid.* Cluster Computing and the Grid, 2005. CCGrid 2005. IEEE International Symposium on, 2005. **1**.

[110] Bunting, B., et al. *Web Services Context (WS-Context).* 2003 [cited; Available from: http://docs.oasis-open.org/ws-caf/ws-context/v1.0/wsctx.html.

[111] Aktas, M.S., G.C. Fox, and M. Pierce, *Managing Dynamic Metadata as Context*, in *The 2005 Istanbul International Computational Science and Engineering Conference (ICCSE2005), Istanbul, Turkey*. 2005.

[112] *HttpClient*. [cited; Available from: http://jakarta.apache.org/httpcomponents/httpclient-3.x.

[113] *Google Web Toolkit (GWT).* [cited; Available from: http://code.google.com/webtoolkit/.

[114] *Direct Web Remoting (DWR).* [cited; Available from: http://getahead.org/dwr.

[115] Garrett, J.J., *Ajax: A New Approach to Web Applications.* Adaptive Path, 2005. **18**.

[116] Humphrey, M., et al., *State and events for web services: a comparison of five WS-resource framework and WS-notification implementations.* High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on, 2005: p. 3-13.

[117] Altintas, I., et al. *Kepler: an extensible system for design and execution of scientific workflows*. in *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*. 2004.

[118] Ludäscher, B., et al., *Scientific workflow management and the Kepler system.* Concurrency and Computation: Practice and Experience, 2006. **18**(10): p. 1039-1065.

[119] Andrews, T., et al., *Business Process Execution Language for Web Services Version 1.1*. 2003.

[120] Altintas, I., et al. *A Modeling and Execution Environment for Distributed Scientific Workflows*. in *Proceedings of the 15th International Conference on Scientific and Statistical Database Management (SSDBM)*. 2003. Boston, MA.

[121] Tom, O., et al., *Delivering web service coordination capability to users*, in *Proceedings of the 13th international World Wide Web conference on Alternate track papers \& posters*. 2004, ACM Press: New York, NY, USA.

[122] *Freefluo workflow enactment engine*. [cited; Available from: http://freefluo.sourceforge.net.

[123] Laszewski, G.v., M. Hategan, and D. Kodeboyina, *Work Coordination for Grid Computing*. 2007.

[124] Atkinson, B., et al. (2004) *Web services security (ws-security).* **Volume**,

[125] Perera, S. and D. Gannon, *Enabling Web Service Extensions for Scientific Workflows*, in *HPDC2006 Workshop on Workflows in Support of Large-Scale Science (WORKS06)*. 2006: Paris, France.

[126] *Purse portlets*. [cited; Available from: http://www.collab-ogce.org/ogce2/purse-portlets.html.

[127] Sandhu, R., D. Ferraiolo, and R. Kuhn, *The NIST model for role-based access control: towards a unified standard.* Proceedings of the fifth ACM workshop on Role-based access control, 2000: p. 47-63.

[128] *Indiana University Molecular Structure Center (IUMSC).* [cited; Available from: http://www.iumsc.indiana.edu/.

[129] *RSS portlet*. [cited; Available from: http://sourceforge.net/projects/rssportlet.

Vitae

NAME of AUTHOR:	Mehmet Akif Nacar
DATE OF BIRTH:	14 November 1972
PLACE OF BIRTH:	Sanliurfa, TURKEY

EDUCATION

MARCH 2008	Ph.D. in Computer Science Department of Computer Science Indiana University, Bloomington, IN, U.S.A.
DECEMBER 2000	M.S. in Computer Science Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, U.S.A.
DECEMBER 1998	M.S. in Computer Education Gazi University, Ankara, TURKEY
JUNE 1995	B.S. in Computer Engineering Trakya University, Edirne, TURKEY