CLAS-NOTE 2004-014

# "Grid Computing for Physics Environments,"

J. Hone, L. Dennis, P. Eugenio, and G. Riccardi

Florida State University

April 22, 2004

# ACKNOWLEDGMENTS

The author wishes to thank Larry Dennis, Mark Ito, Paul Eugenio, Chip Watson, and Greg Riccardi for substantial assistance and guidance during the completion of this project. Also, thanks to the thesis committee consisting of Larry, Susan Blessing, Jorge Piekarewicz, and Dan Schwartz. Last but not least my wife Meghan as she helped during the entire process.

TABLE OF CONTENTS

# LIST OF FIGURES

# GLOSSARY

**Apache HTTP Server**: A very popular, fast, and highly functional open-source web server hosted by the Apache Software Foundation. It is now included in the latest versions of Red Hat Linux, and can be used to turn any box with a stable IP into a web server in several easy steps.

**Apache Software Foundation**: A host of many open-source projects, including most of the ones I use in constructing this project. Jetspeed, the Apache HTTP Server, and Tomcat are just a few of these projects. Apache is supported by IBM.

**Axis**: Apache's web service server.

**DOM**: The Document Object Model. This is a way to have an XML document, entire or fragment, loaded into one object for easy manipulation. DOM is much slower than its alternative, SAX.

**DTD**: Short for Data Type Definition, this is one way to specify the structure and content of a new markup language by declarations of markup elements such as tags, attributes, and character data.

**Grid**: A set of connected resources and services.

**IBM**: Supporter of the Apache Software Foundation. Many of its programmers are involved in Jakarta projects.

**Jakarta**: A collection of open-source Java projects supported under Apache. These projects tend to generally support new functionality for web applications.

**Java**: A platform-independent programming language which is the most popular in open-source projects because of its focus around object-orientated design. It achieves platform independence by using its own compiler and runtime execution machine, meaning that any Java program is compiled to the same machine code and run with the same environment, limited only by Java version.

**Jetspeed**: A Jakarta-hosted portal project.

**Open-source**: Programming projects which are intended to be built by collaboration of distributed programmers have this name, referring to the fact that the source code is open to the public. Contributors focus on providing Application-Programmer Interfaces (API) which ease programmer use of pluggable open-source pieces in an application's design.

**Portal**: A customizable, personalized web application which offers persistent storage for user data. A common professional example is My Yahoo.

**Portlet**: The basic unit of content in a portal. Each page in a portal consists of multiple portlets. For example, a home page may contain an email portlet, a weather portlet, and a news feed portlet.

**SAX**: The Simple API for XML, this is a way to process XML by writing code to parse different XML events, such as encountering the start or end of a tag, or an attribute. Several layers of filters can be constructed which process the same set of events in different ways.

**SOAP**: An acronym meaning Simple Object Access Protocol. SOAP is the language of web services, describing the messages web services and their clients pass back and forth to each other.

**Sun**: The software company which created Java and hosts different implementations of W3 standards as new open-source projects. Most of its standard implementations such as the Java API for XML Processing are replaced by competing Apache projects such as Xerces or Xalan in this application.

**Tomcat**: A Jakarta-hosted web application container.

**Turbine**: A Jakarta-hosted web application framework. Turbine is intended to be a kind of skeleton which allows different web applications to be built around and on top of its different structures and services.

**Velocity**: A Jakarta-hosted templating service. A Velocity template contains markup and references to objects placed in code associated with the template.

The Velocity engine which renders the final markup is in this application controlled by the portal.

**Web Application Container:** An engine used to administer multiple web applications and contain their operation under supervising processes. For example, the servlet context, logging, and security can all be provided for an application deployed inside a web application container.

**Web Service**: A new technology designed to allow complete interoperability between any service and/or resource an application may need. Complete interoperability reflects the ability to seamlessly connect deployed services into an application without knowing the mechanisms of how those services operate. All a programmer knows is an interface for interacting with and communicating with the service.

**WSDL**: The Web Services Description Language. These are XML documents conforming to the WSDL schema that define web services, from the parameters they accept to the data they return. They are intended to be parsed by client programs which can determine from the WSDL content what service the program will provide, without any foreknowledge about the service itself.

**W3**: Otherwise known as WWWC, or the World Wide Web Consortium, this is a group of leading scientists who publish standards for new technologies to ensure cohesiveness among different projects with the same goal. For example, they create a DOM standard, and projects wishing to implement the DOM, such as JAXP or Xerces, must adhere to the specifications set forth in the standard.

**Xalan**: An Apache project focused on XML translation via XSL.

**Xerces**: An Apache project focused on XML transformation and manipulation via SAX or DOM.

**XML**: The eXtensible Markup Language. XML in structure is an abstraction of markup languages like HTML, and can be used to create other markup languages

by defining data types and structure via XML Schemas and DTDs. These created markup languages represent new encodings of data into information.

**XML Schema**: An XML-oriented way to specify XML structure. This surpasses DTD functionality in many ways, including allowing character data to be validated against regular expressions and other data types. Inheritance of data types is heavily used in this project as well.

**XSL**: The eXtensible Stylesheet Language. A stylesheet or set of stylesheets specify rules dictating how to translate input markup to output, be it XML, HTML, or text.

## ABSTRACT

Compute grids for physics applications have the potential to solve significant issues which contribute to low data processing and publication rates. This potential includes an ability to transparently manage large amounts of data and to greatly enhance project computing resources. This project proposes construction of a structured computing grid from the combination of several open-source Java web application systems, with the main components being the web service server called Axis and the portal called Jetspeed. This thesis outlines the main motivations, the system design requirements to answer these motivations, the details of the system implementation, the underlying technology, and the first working compute grid application, running at the Hadronic Nuclear Physics cluster to provide simulation and analysis services to Jefferson Lab (JLAB) Hall B users. This effort works in conjunction with data grid services provided by the Storage Resource Manager which allow access to CLAS data on tape at JLAB. Technical detail on how to install, troubleshoot, upgrade, start up, navigate, and deploy physics applications to the system is also provided.

*Chapter 1*

# INTRODUCTION

In this paper I discuss a project to implement a highly extensible solution to use portal technology to provide structure for grid computing. The main motivation for this is to provide an example of how these technologies can help meet the needs of the physics community. The physics community is always striving for improvements in its ability to understand and measure natural phenomena. Often this requires dramatic increases in data processing rates and data volume. Experimental and theoretical facilities expand as fast as possible, taking advantage of advancing technology to increase their raw capability to collect data. One consequence is that the amount of data the physics community wants to process has grown monotonically with time, while the number of people available to process and understand this data remains relatively flat, making improvements in methods and efficiency necessary conditions for capability growth.

For example, Thomas Jefferson National Lab[1] plans an upgrade in facilities for the beam energy to 12 GeV from 6 GeV. They also plan to build a fourth experimental hall, Hall D. Current data collection rates are about 130 terabytes per year, while the rates after the expansions are projected to be over a petabyte per year. In fact, few facilities in the world match the data collection rate common at Hall B[2] as it stands currently. However, at current data rates the management of the bulk of the data is problematic. This manifests itself as a low

[1] www.jlab.org

[2] http://www.jlab.org/Hall-B/

physics publication rate and/or long periods of time between data acquisition and publication.

To address this problem, Jefferson Lab added to its existing data storage system of one tape silo containing 120 tapes at 6GB per tape a second as recently as the winter of 2001. This second tape silo relieved demand for one year, and subsequently reached capacity so the lab is now in the same situation as before. Clearly, when the data not yet analyzed comes in faster than data whose analysis is finished, accumulation will occur, bogging down the publication process. Some individual collaborators on an experiment may speed up their part of the analysis by gaining higher priority on available resources than other collaborators, as occurs with "fast-tracked" papers or simple batch priority tricks. However, this does not increase the speed of analysis of the collaboration as a whole until the total amount of resources is increased. The resulting backlog cannot be eliminated without more efficient processes.

Jefferson Lab is also moving toward grid use for data replication. Experiments usually perform such compute-intensive tasks as simulations off-site, at member university facilities. However, using a grid to replicate Jefferson Lab data at other sites would allow other major tasks, such as track reconstruction, analysis, and real-time calibration, to occur with the same ease of movement and access that would occur on-site at the Lab itself. Grids provide the tools to coordinate the movement and tracking of multiple gigabytes of physics data, creating automatic paths to and from major computing facilities.

Grids and portals are the two main tools used in realizing the solutions discussed here. A grid can be defined as an abstract collection of computing and data resources whose functionality is accessed by a collection of programs and packages called web services. A grid is like an abstract batch computing system where clients sign on to use services already deployed to the grid. They have no involvement in any other system details, such as cluster membership, security, data management, logging, accounting, or error handling. Since a grid is meant to

be an abstract gateway to services deployed, published, and hosted at arbitrary locations, a portal, with its ability to encapsulate a wide array of functionality, is a natural choice in which to embed a highly extensible grid. A portal can be defined as an application built for the web which has the ability to hold customizable, personalized information and content in a persistent, consistent manner for a community of client users. For example, the portal can easily remember information such as a history of a user's last "N" job submissions to provide a user with a job cache as well as easy resubmission capabilities. The basic unit of content in a portal is a portlet. Portlets are added to individual pages within a user's portal account space and are meant to be easy to customize to reflect a user's preferences. Therefore, grid applications deployed as portlets enjoy the flexibility, extensibility, and added functionality that both new technologies provide. A more comprehensive treatment of how this combination satisfies design goals to answer the main project motivations can be found below.

A further motivation was to design a product that fits the application domain of a physics environment. Physics grid computing needs are among the most cycle-intensive and data-intensive in the world. The system designed needed to embody a grid which would be most useful to fit these needs while still remaining flexible. Two kinds of grid paradigms are available, structured and unstructured. The difference between the two is that a structured grid uses fixed destinations or endpoints for services while an unstructured grid uses dynamic destinations which it will search for at runtime. An example of an unstructured grid is the SETI@home[3] project, hosted by the University of California at Berkeley. SETI@home is a program which, once downloaded and installed, runs during idle computer time as a screen saver. While the screen saver is active, SETI@home will download files of astronomical data from a remote server and analyze it with the packages that also come with SETI@home. Finished data is

[3] http://setiathome.ssl.berkeley.edu/

then sent back to the original server and if possible the process begins again. It is unstructured since services are performed at arbitrary locations, and any machine that asks for data is given it. A structured grid such as the system described in this paper knows both client and server beforehand, and our reasons for choosing this paradigm are included below.

Another way this system fits the needs of the physics community involves the projected use of the system in practice. While the possibilities are endless, two separate ways are discussed here, which focus on the traits that make a physics environment distinct from others in which grids may be employed. The first deals with the fashion in which the system will be used by clients.

There remains the most relevant technological distinction for physics community grids: that the need for specificity in grid computing applications is so high that the usual grid model of searching for a service to provide needed data is insufficient. This is in sharp contrast to the envisioned business application of searching for the best price on a given service. The following three examples set the general pattern. In one case, grid applications would be deployed corresponding to theoretical or phenomenological models. Someone wishing to run these models may subscribe to a specific portal for this purpose. The second example applies to collaborating scientists who need to run packages or programs installed outside of their normal working group resources, such as at different labs or universities. A third example involves the use of roles in the system. A portal makes use of roles in defining the members of its community, and this built-in feature eases the job of defining clear distinctions between and among physicists and computer scientist grid administrators. It is given that as much of the maintenance and use of new technology as possible should be delegated to grid administrators, while physicists spend as much time as possible developing new physics.

The second way that this is an appropriate solution for the physics community concerns the resources the grid computing environment consumes.

The batch farm is the most popular and most capable resource unit in solving physics data processing problems today. Physics is a field where batch computing is a necessity in many fields, and so our system should and does encourage an optimized use of these resources. Our premise is that as grid computing centers develop, they will employ batch systems that focus on providing one type of job, so that each system can be customized to perform its task efficiently. It is unlikely that a batch system which handles every type of job will be most efficient, so examples such as the following will be most prevalent. For instance, there could be a simulations cluster at FSU, an analysis cluster at JLab, and a calibrations cluster at Old Dominion, as opposed to each of the three centers doing all three jobs. This would improve performance by allowing hardware and scheduling schemes to be customized for the jobs at the different sites. Also addressed is the bandwidth problem involved in transporting hundreds of gigabytes of data, which becomes easier to solve by pre-staging the transfer, so that as one large job is running, another is loading. Then, efficient scheduling becomes a tractable problem, fewer cycles are wasted, efficiency improves, and costs go down.

The following list contains design goals to direct this project to address the motivations already presented.

- The system should reflect a structured grid computing solution in as transparent a manner as possible. Physicists should not have to study a large amount of new technology to use, build, or extend the system.
- The system should allow the appropriate level of modularized control of security and resource utilization. For example, every grid application should be able to be deployed by a physicist but managed by an administrator. Additionally, rules for cluster use should be determined by the cluster owners. Here, deployment refers to the creation of the necessary files which fully describe the application and any interaction with it using the expertise of the physicist and any other interested parties.

The grid administrator can work with the physicist to publish the application in the portal, shifting publication management duties and issues to the realm of the grid administrator.

- The system should be highly extensible and flexible. Future improvements should have a mechanism to easily plug into the existing system, and as many kinds of grid applications should be available as possible.
- The system should provide a physicist deployer with full control over the application deployed. The application's user interaction, documentation, interface, and data validity should be specified as far as possible with as many options as possible by the deployer.
- The system should be useful to the physics community outside of its use for grid computing.

This thesis has eight more sections. In the next section I will discuss system design requirements and the solutions employed in generic and technical detail, followed by a chapter on system implementation. A specific physics application using the completed system is then discussed, which is a particular implementation of this technology to create a simulations, post-processing, and analysis service available to members of the portal. Jefferson Lab (JLAB) users will be able to move data to and from this cluster at Florida State University (FSU) for the purpose of running GSIM jobs just using the capabilities provided by the portal. A conclusion provides results and how well the project motivations are answered. Four appendices then document how to install and run the system; how to navigate around the portal; how to deploy an application, including examples of the various necessary files for different situations; and a project timeline.

*Chapter 2*

# SYSTEM DESIGN

**Introduction:** In this chapter I first discuss the system design requirements which served as guidelines for the development of this project. Following that, a detailed explanation of each requirement is given. Many new technological ideas are employed by different projects which are incorporated into this system. Therefore, an introduction of these new ideas and the implementing technology used to meet these requirements is made, with direction on how the ideas and technology are used to fulfill the design requirements. These sections are called Grids and Portals, XML, Web Services, and Templates.

**Requirements:** These design requirements were intended to implement the design goals discussed in the Introduction in order to achieve answers to the main motivations for this project. Each item in the list is described in comprehensive detail below.

- Educational without loss of professional advantage.
- Flexibility to work with a broad range of applications while supporting the same level of control and specification of interaction, as well as extensibility to ease entry of new applications and upgrades to new versions of old applications.
- Self-documentation to reduce physicist documentation workload and increase deployed application lifetime.
- Short system-use learning curve.
- Clear separation and appropriate delegation of system responsibilities.

The first requirement demands a system which can, depending on the role and sophistication of the user, be as verbose as necessary or as sleek and solely utilitarian as the user can handle. Some physics applications have very high

learning curves. While a particular interface to an application may be easy to learn, understanding the physics the application is employing and using the application to produce contributions to research work is the more difficult task in most cases. The expertise a physicist must have to use an application effectively depends on whether he is using the application, extending it, or developing it. The intention was to provide for personal expert assistance, enabling the system to serve as an educational tool when that functionality is called for and to simply perform a job without any extra trappings when it is not.

The ability to determine role and sophistication of a user by an HTML client is just one feature among many, such as security, session management, logging, and other aspects of web application infrastructure, which can be provided by existing open-source systems. It was clear from the beginning that it was essential to build this system on the work of others, so that web application specialists could contribute in their area of expertise to enhance the base system while physics customization remained the main focus of the work. This required an extensive search of existing technology to find the appropriate software platform. Factors such as the robustness of features this system provided, the community support among users and developers of the system, the future direction and motivations of the developers of the system, and the difficulties associated with any barriers in beginning use of the featured system were considered.

High flexibility involved searching for a method to simply deploy wide ranges of applications. This required searching for a solution in describing physics applications which would be able to cover the broad spectrum used in compute-intensive research. This must combine with the requirement for a low system learning curve since a greater level of flexibility usually means more options for the system user and can require a more complicated mechanism for system use. Also required was a system with low barriers to the integration of new technology to upgrade or initialize system functionality. In one sense this

applies to the work of the application deployer, who should have a quick, clear means to convey upgrades in functionality in the application they deployed. In another sense, this means that upgrades to the system's component technologies should be as easy to introduce as possible, since patching a system is not always transparent and since new functionality will be desired over the lifetime of the system.

Self-documentation is important for this system partly because of the central role that graduate students play in the maintenance and advancement of physics technology. These students are dedicated but transitory, so that their systems, applications, and code must be able to be maintained after they have moved on to other work. Physics as a field in general has a problem with lack of documentation and other utilities concerning software used in research which makes reusing, expanding, or patching old code more difficult than necessary.

A short system-use learning curve ensures that the system can be taught to physics contributors without the need to introduce most of the new technology employed. Web service technology is still developing, and can be quite complicated, especially to scientists outside of the computer science research field. A broad range of integrated technologies is required to create a viable grid application, and so it is conceivable that a community of physics users could be disinclined to work with grid solutions as long as other solutions to the common heavy computing requirements were available. If possible, a system that requires no code to be written by any application deployer would provide the lowest barriers to entry of a particular compute grid application into the grid. A low learning curve would also encourage more rapid development of compute grid applications and help to create and deploy systems of immediate impact and use to the physics

community.

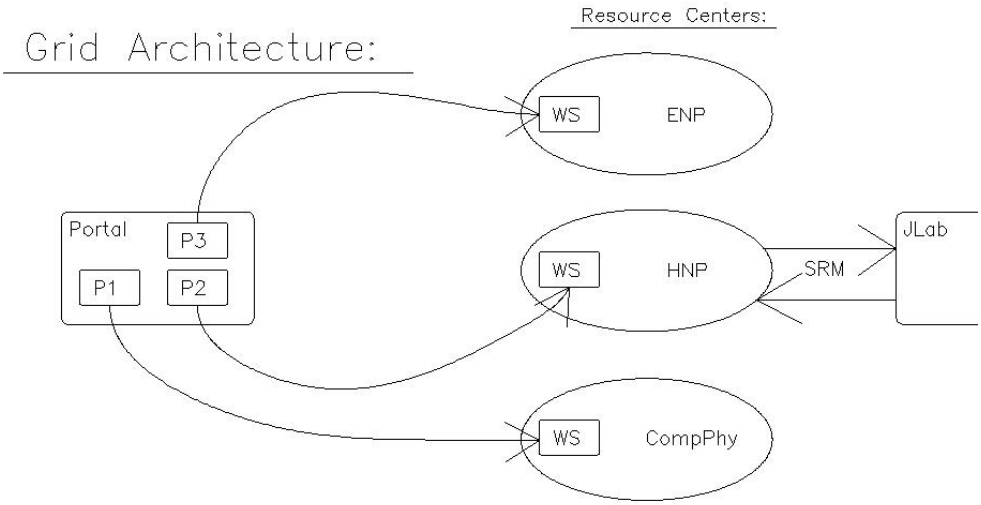


*Figure 2.1 Sample grid architecture.*

The last requirement for clear separation and delegation of responsibilities asks that every agent's role in deploying an application to the grid be well defined. More than one person should be able to fill a role at one time. This emphasizes the community nature of creating a compute grid, since multiple interested parties may want to be involved in every step of the process. This also requires defining a sensible process for describing, deploying, and publishing physics applications to the grid which takes these considerations into account.

**Grids and Portals:** To fulfill the requirements in this list, we sought to rely solely on open-source Java,[4] XML[5], Web Service[6], and web application technology since it can be redistributed at no cost, it is widely available, and enjoys a high rate of advancement in functionality over time from the open-source community.

[4] http://java.sun.com

[5] http://www.w3.org/XML/

[6] http://www.w3.org/2002/ws/

Grid and portal interaction is the basis for much of this functionality. A grid can be defined as a collection of services which can satisfy a client's request for one or more of those services without a client having knowledge of the deployment of the service. Servlets, which also provide remote service performance, require links in client and servlet code which generally increase in complexity as functionality increases. Grids are a way for a service provider to publish a service and a client to consume the service without prior collaboration between the two.

A portal is a customizable personalized web application that remembers private data about each individual user, maintaining persistent, consistent sessions of member use. Portals are HTML clients based usually on servlets and are becoming increasingly popular among e-businesses as ways to more closely connect with their community of customers. A portal's unique interface lowers our system's learning curve significantly, and we receive the added benefit that many portal systems have low barriers for the entry of new technology to provide new functionality. Moreover, a portal can naturally proffer such member traits as user role or group to portlet applications, which enhances our capability to separate and delegate system responsibilities. The two applications, grids and portals, can combine to provide an environment for different grid services to have access to a wide array of useful portal tools and personal user information. So, the portal provides a toolkit to encapsulate grid functionality for higher user or client accessibility. For instance, no grid naturally logs its activities or errors and no grid naturally provides highly useful interfaces for its specific services. A grid is simply a network of connected resources and services. The contention here is that other web applications which are created to specialize in other tasks can and should be integrated with grids.

**XML:** XML is the eXtensible Markup Language, and it is a meta-markup language that is used to define new markup languages which encode and convey information. XML documents are determined to be valid and well-formed by

restrictions placed in files called Document Type Definition[7] documents (DTDs) or XML Schema[8] documents. Either kind can define the rules for content appearing in an XML document. DTDs specify the allowed order, name, and structure of elements, attributes, and other nodes of XML content. Schemas go one step further by allowing a user to restrict the type of character data that is contained in the XML document by checking the content against regular expressions and some basic built-in types. Schemas also allow inheritance, so that types of data structures can be defined from combinations of other basic types. XSL[9] (eXtensible Stylesheet Language) stylesheets define the translation of XML documents into other documents, such as XML, HTML, or even plain text, through stylesheet rules contained in XSL documents.

Existing XML technologies provide the foundation for satisfying almost all of the design requirements. The first requirement under consideration is flexibility and extensibility. XML structures can be defined by anyone, and so this freedom to create complete structured languages and to design numerous ways to translate them provides our grid computing system with tools to deploy any physics application to the web that can be described by any sort of language which XML can encode. Closely following this is self-documentation. XML can define custom languages in a highly self-describing manner, and it gives physics applications described in this way a layer of natural documentation, so that someone reading XML documents can by the text in the markup and structure of elements and attributes gain as much information as the application deployer is willing to provide. Also, using XML languages to describe physics applications can mean that physicists do not need to be conversant with the numerous technologies which can provide web service and grid functionality. They only need to know one of the ways to create and edit XML documents. Writing what

[7] http://www.w3.org/TR/REC-xml

[8] http://www.w3.org/XML/Schema

[9] http://www.w3.org/Style/XSL/

amounts to validated configuration files as opposed to writing, compiling, and integrating code shortens the learning curve significantly. Finally, the division of XML technology into instance document, schema, and stylesheet provides a ready means of dividing system responsibilities. The identities of each of these types of documents are already well-defined by the XML community, and so each document's role in an XML information system is well established, accepted, and supported. One advantage which comes from this is the development of XML, XSL, and Schema editors which can ease the task of knowing three new languages and speed up application deployment. This system only assigns to each document type a job as a step in the process of creating and using the physics grid and its applications.

**Web Services:** A web service is the basic tool used to provide functionality to grids, and SOAP[10] is the XML language that these web services use in messages to each other. Web service is the name for the technology which aims to supply complete interoperability between any services provided by applications accessible over the web. Web services can be described by XML documents written in an XML language called the Web Service Description Language[11] (WSDL) for the purpose of dynamic service location and invocation. The Simple Object Access Protocol, or SOAP, is itself an XML language designed to hold information passed to and from web services and clients. A SOAP message consists of a SOAP Header, Body, and any Attachments inside an enclosing SOAP Envelope. The Header contains XML element children known as Header Blocks. Each Header Block represents an intermediate destination for the message as it travels to its ultimate destination, and the XML inside each Header Block must be understood by any intermediate destination for which the Header Block is targeted. A similar requirement exists for the Body Blocks of the SOAP Body, except only at the ultimate destination. If any destination does not

[10] http://www.w3.org/2000/xp/Group/

understand the XML in any Block targeted for it, or any other type of error, such as transport, occurs, a SOAP Fault block is added to the SOAP Body and the message is returned to the client. SOAP Attachments are usually large blocks of data, such as image data, which is simply transferred as is and is not combined with the SOAP XML Envelope. Our main SOAP tool is the Apache-supported web service server called Axis[12]. A web service server is a server application servicing clients which call asking for access to locally deployed web services. Requests are routed to invoke the appropriate service and responses are returned to the client containing the resulting information. Axis is the most complete web service server, supporting all of the above SOAP functionality and including many other useful features, such as a web client for remote deployment and management of web services.

Axis supports both structured and unstructured grid applications, but we chose a structured grid for reasons based on a cost-benefit analysis of the advantages and disadvantages of each. An unstructured grid has the advantage of tremendous flexibility and potentially unlimited resources which arise from the open-ended service provider inherent in its design. An unstructured grid can plug into as many resource sites which normal technological limits such as bandwidth allow. However, the disadvantages are that the grid's benefactors cannot always trust the information that is returned, due to the anonymous nature of the information generators, and also that jobs are not available on demand. In an unstructured grid, scheduling and timetables for the benefit of the client are not available. A structured grid answers the concerns of the unstructured grid without giving up much flexibility. In particular, this structured grid utilizes endpoints for services which are fully deployed and maintained by professional members of the physics community. This allows specialization in tools for both clients and servers. Client utilities can include rough estimates on

[11] http://www.w3.org/2002/ws/desc/

completion of services and guarantees of authenticity and security of the service information. Server utilities can include specialization of resource sites to specific services for better efficiency and more optimized scheduling. All are examples of the advantages provided by a structured grid. Security is an especially important consideration since the resources a physics grid can provide are likely to include very expensive clusters of grant-funded computers and data management systems.

This system does not make use of WSDLs for any purpose, due to the particular nature of the physics community in which the system will be deployed. Our main reason is that the WSDL features of dynamic location and invocation of services are not desirable in this context. The contract a web service provides via its WSDL document concerning performance of its service consists of a method name, a parameter list, and any XML documentation the WSDL can contain. It is unlikely that these features can be sufficient to describe the many facets of a physics application, especially since the scale the two systems were meant to address is completely different. Web services were meant to provide small pluggable pieces which together combine to create an application, with the links between functionality provided by WSDLs. This achieves the main goal of web services, complete and full interoperability between applications and functionality in clients and servers across the web. However, physics applications are often several-million-line programs, usually with features provided in multiple programming languages in patches and extensions that are intended to be a conglomerate of functionality. Therefore, describing one in terms of the other is inappropriate.

With this system, we simply take advantage of the fact that the large physics applications must have some method for user interaction, and use a grid computing environment as a uniform way to encode and describe this interaction.

---

[12] http://xml.apache.org/axis/

While one can imagine a sequence of such applications (web services) the practicalities of bandwidth limitations generally do not encourage the random transmission of large data files from one service to another.

**Templates:** Another new area of technological enhancement employed in this system is templating using the Velocity[13] template engine. A template engine creates applications by the combination of two files: pseudo-HTML templates and Action classes. The templates contain HTML laced with names representing objects. These names will refer to actual objects, so method calls and data members can be inserted which adhere to Java syntax. The Action class contains code with many pieces which are conditionally executed depending on the state of the application. This code inserts objects into an intermediate staging area, where the engine matches objects identified by names given in the Action class with object names appearing in the template. Once the objects are matched by name to their template references, object methods and data members can be resolved and executed in order to deposit relevant information into the text of the resulting HTML.

One large advantage of a template engine system for web application design is that this technology utilizes the Model-View-Controller (MVC) design paradigm. MVC is a way to combine results from the different jobs in web development, allowing programmers to solely write code while interface designers solely write HTML and Javascript. This makes application development easier by enabling a complete separation of these responsibilities and lowering the need for developer interaction with domains out of their main specialty. Our choice of Velocity over other main templating engines such as Java Servlet Pages[14] (JSPs) developed by Sun[15] or PHP[16] reflects Velocity's low learning curve, high portal

[13] http://jakarta.apache.org/velocity/

[14] http://java.sun.com/products/jsp/

[15] http://www.sun.com/

[16] http://www.php.net/

and developer community support, excellent MVC realization, and continuing open-source development.

An example of how templates can be used to meet some of the design requirements is contained in the following. In most template systems, different templates can be designed for the same application based on a user's level of expertise. One template for beginners could be very verbose in its explanation of the application, while another template, for experts, can be terse and streamlined for fast and easy submission. The two can even be combined in one template. In this case, role information provided by the portal can be interpreted by Velocity and the template engine filters the template and renders the appropriate content. Other examples arise when different code needs to be executed depending on certain parameters like user role. The expert may wish to monitor the progress of each beginner, but the template may stay the same for users in both roles. Yet each role needs different programs to execute, one which monitors the beginner and one that does not, since the expert has no need to be watched. In each case, portlet configuration and other information can match one template to one program, so that the appropriate matches, and hence the appropriate functionality and data, can naturally serve situations with very different user needs and results. In this way the portal can be educational without loss of professional advantages such as quick submission in a page not cluttered with lengthy descriptions, images, and other such documentation aids.

*Chapter 3*

# SYSTEM IMPLEMENTATION

**Introduction:** This chapter gives details on the implementation of the system design into a running compute grid hosted by a portal. It begins with an overview of the system implementation and then describes the features the system supports. Component technologies of which the system is comprised are described in each section detailing the feature they support, and the reasons for selecting the technologies along with their features and advantages are presented. The system features described include Job Submission, Job Monitoring, Workflow, and Job History.

**Overview:** The Apache Software Foundation[17] is the source of support for most of the open source component projects which make up this system. The particular category they are developed under is called the Jakarta Project[18], which produces systems aimed at resolving the various concerns and responsibilities associated with creating and maintaining server-side applications which support e-business on the web. Integrating many Jakarta subprojects together can provide a powerful application with broad functionality, and our system attempted to implement such a solution.

The system macroscopically consists of a Jetspeed[19] portal running inside a Tomcat[20] web application container which plugs in to the Apache web server[21]. The Tomcat web application container hooks into the more familiar Apache

[17] http://www.apache.org/

[18] http://jakarta.apache.org/

[19] http://jakarta.apache.org/jetspeed/

[20] http://jakarta.apache.org/tomcat/

[21] http://httpd.apache.org/

HTTP web server via a module called mod_webapp.so. This module, once deployed to Apache's code library, allows a connection to be created in the Apache HTTP web server's configuration file between the web server and the web application container. Pseudo-addresses are then created for web applications residing in the web application container, allowing a unique name and address to be assigned to applications deployed to each web application container. Once the Apache HTTP web server is running, it will try to connect to the web application container when it needs to service HTTP requests of pages from the paths of the deployed applications by connecting to the Tomcat web application container. The Tomcat engine is configured to serve requests on a specific port only through Apache connections, and provides several highly reliable container-level utilities, including session management, logical security, logging, page and servlet caching, and remote application management.

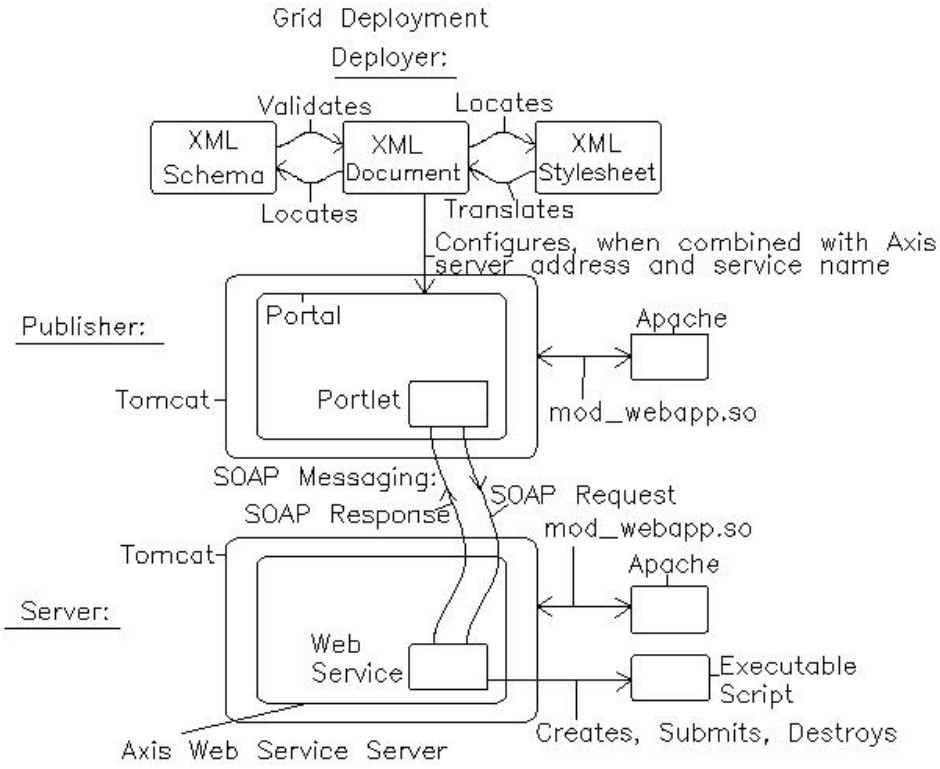


*Figure 3.1 System Architecture.*

Jetspeed is based on a general web application framework called Turbine[22]. Turbine is intended to be a skeleton which allows developers to easily build custom applications around the existing base infrastructure, while providing some unique advantages to the created application itself. Turbine fully employs the concept of pluggable services, in order to separate different jobs required in a web application and allocate those jobs to specific services only by configuration. A pluggable service is a service provided to a web application which is used without knowing what code will provide the service at compile time, so that multiple services can be plugged in to the application to fill the role. One common illustration occurs since Turbine recognizes the need for most applications to have a logging service, but does not provide any code to do that job. It merely provides a placeholder class for the logging service throughout its main code. Whenever the logging service is required during runtime execution of the application, Turbine checks its configuration to see if an appropriate application has been deployed to perform logging duties. If so, then the placeholder class is replaced by the plugged class by the runtime environment. The default logger in most cases is supplied by Log4J[23], the Jakarta Logging for Java project, but an application could use its own logger. In this case Turbine would search through the application's configuration files and replace all class names and parameters appropriate for the default, Log4J, with custom values. This concept is employed below in one of the batch monitoring tools which utilizes a Turbine service to filter an XML description of jobs running on a batch queue. Information is filtered by a search string and the results are displayed via a Velocity template and action class combination. Velocity is also plugged into Jetspeed as a template service, but it is not the only one available in Jetspeed. JSP is also available as a templating service, and both are plugged and called as is appropriate.

---

[22] http://jakarta.apache.org/turbine/

The Jetspeed portal itself has many unique features. It allows a user to create many portal pages, and to select which portlets appear on their pages. It also presents several layouts, such as two or three column, for the user to choose from to arrange their portlets. Appendix B discusses the interface which enables this functionality and how a new user may navigate it to start using these features. A user may also select multiple color schemes, or skins, from among those available to apply to individual portlets or whole pages. All this information is stored in the portal as a set of files for each user in an XML language called Portal Storage Markup Language (PSML). Alternatively, PSML data for each user can be stored in a database instead of directly on the file system. PSML files provide transparent persistent storage of the user's customized skin, content, and layout information, as well as all portlet parameter values. Security is maintained in the portal by helpful administrator level portlets. One portlet lists all current and potential portal members and allows an administrator to alter security roles and groups for any individual, as well as approve or deny membership or place an account on probation. From this area one can also create new security roles and groups as well as search for particular users based on simple strings or regular expressions. Each portlet has associated with it certain permissions, such as the permission to maximize, minimize, customize, close, or simply view. Permissions for specific portlets become attached to roles so that a role may grant a particular permission on a certain portlet but not grant that permission globally to all portlets. Other administration level portlets survey the status of the portal and monitor such system information as the current Java runtime or anything flagged by either the Bad URL Manager daemon, the disk cache daemon, or the XML feed daemon.

**Job Submission:** To a regular portal user, the job submission portlet appears to be only an application description followed by a table containing batch and

[23] http://jakarta.apache.org/log4j/docs/index.html

application parameter names, descriptions, examples, and empty boxes for the values. Pushing the submit button at the bottom will submit the job, returning a success message at the top of the portlet if the job is successfully submitted. This information is all that a user who only wants to run jobs will ever need to know. However, there are other layers to this portlet, and those who wish to take advantage of other features like creating new grid applications will need to know a little more.

Portlet instances of the job submission portlet deployed to the Jetspeed portal contain distributed grid services which have been cast as job submission services by XML, XSL, and XML Schema files which are authored by the deployers of the application. The job requests are submitted by constructing a SOAP message and sending it to a web service running inside an Axis web service server. This web service then translates the Body Elements of the SOAP message into a batch submission script via the XSL stylesheets whose addresses are specified in the SOAP message. So, a many-to-many relationship is constructed. One portal can send messages to many Axis servers running at different resource sites, and one server can receive requests from many client job-requesting portals. Once a user has been given a portal account, he can use the job submission feature by adding a job submission portlet to one of his pages.

This mode of conveying information means that the web service uses SOAP messaging for its main communication. One key decision to achieve a highly extensible structured grid was to employ a generic web service which is customized by the SOAP messages it receives. For example, our job submission service builds a script based on a transformation of the content of the input SOAP message, but since the service is merely translating the message, nothing specific to any particular grid application is required beyond the restrictions of XML input form found in Appendix C. In particular no extra code is required, since standard XML and web service tools provide all of the required functionality. The application and batch system are described by the SOAP

message, and this description is translated by the web service according to information given in the SOAP message.

This allows a physicist to deploy an application to a grid simply by the following three steps:

⇒ First, write an XML instance document describing the interaction a user should have with the application and conforming to the rules of all relevant Schemas, both system-specific and custom.

⇒ Second, write a Schema for the dual purpose of validating the structure of the XML message and the form of the data. A logical level of security is then naturally enforced by the Xerces[24] validator and is combined with any wire security in place at the HTTP server or web application container level.

⇒ Third, write an XSL stylesheet to transform the message to its final form.

This way, the application deployer does not have to learn or write any web service or other code, but simply writes these files which amount to XML configuration files.

Once the portal and web service servers are successfully installed, it then becomes the portal administrator's job to publish and maintain these applications for the users via configuration of new portlets, thus creating a clear definition and separation of roles between an application publisher and an application deployer. However, other layers beyond this are possible and even desirable. One application that shall be discussed in more detail later is the deployment of a GEANT[25] simulations cluster via this portal. This system used XSLT to transform the input XML messages into Perl scripts that enforce certain rules specific to the example without any alteration of the system's code. Assuming the SOAP Request to the web service, whose address and method are found from

[24] http://xml.apache.org/xerces2-j/

[25] http://wwwasd.web.cern.ch/wwwasd/geant/

portlet configuration, is completed successfully, the web service will extract the XML document from the SOAP Body and attempt to translate the different parts of the message into a job submission script according to XSL stylesheets. These stylesheets are web-accessible documents found at URLs specified by information within the instance XML document. The translation engine is the open-source Apache project Xalan[26], chosen because of its excellent W3 standard implementation and useful API. It provides all of the required functionality.

To determine what form the interface will take within the portlet, the portlet will first read an XML document whose location is specified in the portlet configuration. To create the required XML document, the application deployer must first decide what information is necessary to run the application or batch system being deployed. Once the structure of the information has been decided, the deployers will add to the XML document markup belonging to namespaces created as part of this project. This will give the portal cues on how to use the information structure contained in the instance document to create interaction between the deployed application and the end user. The portlet's associated Velocity Action class will parse the XML input document using Xerces and return enough information to the template engine to create the HTML interaction interface. This template is intended to serve as a generic job submission template and convey any job submission request by reflecting as many application and batch structures as possible. Also in this interface, examples and documentation can be removed by customization to save space on the screen. Once the submission has been completed with no errors, a success statement appears and the grid computer can check any job monitoring portlets for status of the newly submitted job.

The input XML document provides many features to the application deployer. After deciding the unique structure of the information to be conveyed,

[26] http://xml.apache.org/xalan-j/

use can be made of the markup from the portal-specific namespaces to tell which tags contain parameters to display in the portlet. Once this determination has been completed, the portlet will look for documentation for each parameter contained in markup from an interaction namespace. Then the portal will look for any child tags which are marked to be values that it should provide to the interface. It will also support parameters appearing inside other parameters, so that logical hierarchies can be created. While the use of the documentation feature may be straightforward, the others may require further illustration. An application deployer can use this feature to tell a client portal which application and batch parameters a user should have access to and which they should not. One example of this use is a set of batch parameters that describe the Portable Batch System, or PBS. It might be reasonable that the person who deploys a certain batch system to the compute grid may want to restrict the parameters that a job submitter may set. The batch deployer may want to disallow the ability to set the parameters associated with error and output streams, since a grid user would probably not be a member on the computing cluster who could gather any information from those streams. Instead the batch deployer wants to set the PBS parameters to direct all error and output streams to an email address provided by the user, so they markup their XML instance document to disallow interactivity with certain parameters and simply provide appropriate defaults instead. These defaults will automatically be conveyed to the web service since no interactivity with those parameters has been approved by the deployer. In this way, certain rules can be set and enforced, and further examples appear in our own grid computing center description in Chapter 4.

This input document will also point to fully valid schemas to validate its structure and element content. Upon submission of the job from the portlet, the portlet will first validate the XML which it is about to send out to the web service. If the character data added to the elements is invalid according to the rules created by the grid application deployer in the Schemas, the request to

submit the job will not be made and the user will be informed of his error as far as possible. If everything is valid, then the input XML document is loaded with the values given from the input parameters and appended as one large SOAP Body Element to an outgoing SOAP Request. This feature also helps to provide logical security as well as natural error recovery for such situations as "parameter=value | rm *". Thus, the features provided to the application deployers by use of Schemas include not only the ability to create well-defined, self-policing, and self-documenting structures and layers of structures, but also inheritance hierarchies for types of data and restrictions on the character data as well according to regular expressions and other built-in basic data types. Reuse of popular data types through invocation of data type libraries could potentially become quite common as well.

Third, stylesheets are created by the application deployers which represent the final layer of control that this system allows a deployer to have over his application. This level of control, even more so than the other layers, allows the stylesheets that application deployers write to reflect the composite work of multiple interested parties. The stylesheets determine how the XML layers which the deployers created become job submission scripts. A good example which will be demonstrated later in Chapter 4 occurs when the first layer is only a logical ordering of input information created by the scientist who deploys the application. Further layers can be added around this logical translation to enforce certain resource system rules which reflect site-specific policies on accounting or security. This example explicitly shows how this method grants the opportunity to use the stylesheets like configuration files for different systems. In another example, a stylesheet describing PBS parameters can have different values from site to site, system to system, in its rule for processing a node that describes the location of the error stream. So, different stylesheets can describe the same markup but contain site-specific information as the input XML is parsed. The system allows for this generality by transforming input according to the URL

passed in the input itself. The web service simply recognizes XML attributes placed in certain tags in layers created as part of this project, all described in Appendix C.

**Job Monitoring:** Three different kinds of portlets have been created as part of this project to facilitate monitoring of jobs submitted from the job submission portlet. Two of these portlets are XSL portlets, which read an input XML document and transform it according to XSL stylesheets into readable HTML. The third is a Velocity portlet which uses a Turbine service to search for specific strings within user names on running jobs to provide a report on a certain user's current activities. All three portlets are dependent on reading XML streams produced by the batch monitoring system called JPortal[27]. JPortal was developed at JLab and can be installed on a machine running a PBS server to produce output XML streams available over HTTP. JPortal provides details on the activities occurring within the batch system. Once JPortal is correctly installed and running, the three portlets can be configured to read in the XML documents that JPortal publishes on the web and then display results which inform job submitters of the status of their jobs on the system where they are running.

The first of these portlets is an XSL portlet which reads the XML stream and uses a stylesheet to create a graphic which depicts the capacity of each PBS server. An HTML table is created by the stylesheet containing one column and several rows. The first row displays the name of the server. The second contains a number showing the percentage of the available nodes in the server which are occupied, according to an XPath[28] calculation done in the stylesheet. The next two rows hold different images which are stretched based on the percentages, so that the result looks like a status bar, with parts of the bar appearing full or empty depending on the batch server's state. The last row holds the date and time which the XML that provided all this information was generated.

---

[27] http://www.jlab.org/hpc/ClusterInACan/index.html

The second XSL portlet operates much like the first. While the first was a way to determine at a glance the availability of a batch server to accept jobs, the second is a way for a batch administrator to determine at a glance if the PBS server is reporting any nodes as down, offline, or unknown. The stylesheet transforms the same XML stream into an HTML table titled after the name of the batch server. The table has two columns containing the node name and its state.

The third portlet creates a table of jobs owned by a user whose name has a string matching a search value submitted by the user. Technically, this portlet uses Velocity to combine an Action class and a template. The Action class invokes the Turbine Job Report Service I created to filter the XML looking for the search string and create matching Job objects which are returned to the template engine. The information in the Job objects is then inserted into a HTML table which displays the job name, owner, state, and PBS server ID. Due to the flexibility of the search, this portlet can provide information on any job owner, not just the portal user's jobs.

These portlets are simple for a portal administrator to create and also for a user to customize within the portal. Recall that a portlet is created by adding an entry to an XML registry of portlets in a portal configuration file, most commonly %JetspeedRoot%/WEB-INF/conf/local-portlets.xreg. After a valid portlet entry is added, users can see the new portlet and add it to their pages within their own portal accounts. Most portlets can be created by following the patterns of other portlets in the same category. For example, XSL portlets are all configured as child portlets of a generic parent XSL portlet whose configuration details the class the portal should use when building any child portlets. The generic definition of an XSL portlet and other types of portlets can be found in %JetspeedRoot%/WEB-INF/conf/portets.xreg. Jetspeed configuration of XSL

---

[28] http://www.w3.org/TR/xpath.html

portlets requires several parameters to be set, some specific to this type and some not.

Every portlet needs a parent and a unique name, as well as other attributes and elements that need not have any value at all, such as a URL tag, a media-type tag or a category tag. In many cases the portal will provide suitable defaults, and in others no default is necessary. For instance, only one item specific to the XSL portlet, the stylesheet, needs to be provided. The value given can be a relative path or a URL to a stylesheet. The URL tag is used to point to the XML document which the portlet should transform, and its value can be an absolute URL or a relative path in the file system on which the portal is deployed.

Once created, a user can customize any of the portlet's parameters to which their portal roles allow access, since security restrictions based on role or even user name can be placed on any parameter in a portlet's configuration. Also, any documentation that the portlet creator wishes to place at a portlet or parameter level will be available in most customize templates.

The third portlet can be created as a child of the CustomizerVelocity type, which offers three parameters to be customized. The first two are common to any child of the CustomizerVelocity type, and those are the Action class name and the template name. An additional parameter has been added to this type for this particular class so that the default search string will never be null. Anyone can add parameters to a portlet configuration, and in this case that ability was very helpful since it gave the Action class direct access to persistent storage and easy, transparent retrieval of the search string. There are currently no options for the user to customize this portlet since the portlet is quite one-dimensional, and a user cannot be expected to provide valid values for a parameter like a path to a template or a new Action class name.

**Workflow:** The concept of workflow is important when one is faced with the problem of constructing a grid. For the physics domain, this idea's implementation should prove central in the future's most advanced grid

applications. Workflow is the idea that steps in a work process, designed to accomplish some job which normally relies on human input, can be codified so that a machine can understand subsequent steps and take appropriate action accordingly. For physicists, this has a few important examples. The first is in the area of calibrations. Typically, a few percent of the experimental data are analyzed to calibrate the conditions of the experiment. However, these procedures are almost completely uniform in that they happen the same way each time there is data to be analyzed for a particular experimental setup. One workflow example would automatically perform these steps and present the results to an appropriate expert, who could approve or disapprove of the results, and then automatically initiate either recalibration or a massive analysis of the acceptably calibrated data. One suggestion even proposes to take data as it comes off the detector wires and send it directly to a calibration process and eventually on to the reconstruction analysis process. This idea of using the physicist to monitor the computing without having to initiate the computing is not new, but as data volumes grow is must become more pervasive. The benefits of correct implementation of workflow would lower significantly the burden of managing the ever increasing data volume to its analyzers for every step which can be handed to a workflow manager, and give scientists more time to do science.

A second example arises for a much more general case. As has already been stated, one contention of ours is that different sites will specialize in different types of jobs. So, simulations data may need to move from the site which specialized in simulations to other specialty sites for post-processing, analysis, or reconstruction. This would require some higher overseer for the job process which not only knows the appropriate times and places for a job to pass to its next step, but also knows about the data that is important for the next step and coordinates its transport as well. The general process is that a physics application may be composed of many steps which may need to utilize many resource centers, taking and storing the correct data at every step along the way.

Since data grids are usually developed as separate applications, much like this compute grid has been developed to be separate from other parts of the grid, the integrated workflow application would almost certainly involve a dedicated workflow application, XML language, or other suitable tool. Indeed several are already underway, including some complete products.

One important point to illustrate how this system utilizes workflow can be raised by illustrating the application structure as described so far. The system has one small bit of built-in workflow in that batch system parameters are processed before application parameters. This essentially allows the job submission environment to be configured before application executables are typically invoked. Therefore, an application deployer has to include information about the batch system the job will run on as well as any application parameters as the instance XML document is built. However, and this is especially true for a compute grid, it does not follow that someone able to deploy one set of these parameters is able to describe and deploy the other kind in the same way. So, the only options are to have the application deployers write original schemas, XML documents, and stylesheets for specific combinations of application and batch parameters, or to have them cut and paste examples of unfamiliar systems already written or in place in other deployments into documents dealing with the deployers' realm of specialization. The solution to this problem can be provided both by the web service style utilized and advancing web service technology. The messaging style of web service makes it easier and more transparent to add intermediate destinations to the path of the SOAP message. A SOAP message operating under the web service protocol of Remote Procedure Call (RPC) follows a request-response pattern under which it calls the web service and then returns. With messaging, multiple message patterns can be specified, including intermediate destinations designated by SOAP Header elements in the SOAP Request. The transport details are also transparent to the programmer, since Axis handles routing of the message. Advancing web service technology will provide

this system with a more useful tool than the usual intermediate destination functions such as logging or database entry: grid scheduling. An intermediate grid scheduler should be able to receive a set of application parameters and find a batch site for the job to run. It can then retrieve an XML instance document describing batch parameters for return to the portal. This batch document will be complete with schema and stylesheet, so that the role of an application deployer is purely concerned with communicating their area of expertise to the grid. This grid scheduler may also facilitate other features such as knowledge and permanence of job submission state to further enhance the utility of the compute grid.

In much the same way, we postulate that executing workflow is not an appropriate job for a compute grid. Rather, we hypothesize that any useful compute grid should be fully pluggable with regards to a workflow engine, so that incorporation can be achieved without the benefit of explicit inclusion. Web service architecture can help to achieve this, by either adding the workflow engine as an intermediate destination in a SOAP Header or by making the workflow engine or an engine which can rely upon it the ultimate message destination. In the latter case the engine could simply redirect the appropriate message pieces to the correct web service resource site. A separate workflow engine appears to be a viable solution in part because other tasks; such as grid scheduling, grid logging, and grid data transactions; are all postulated to be developed as separate pluggable pieces, for the good reason that the separate functionality should stay separate for simplicity's sake. As an example, one compute grid portlet could submit information describing one particular physics application to the grid. The grid scheduler searches for a site which can download the required data and execute the job in the shortest amount of time and returns the batch information the user needs to fill out to complete the request. This method enables the compute grid to accurately reflect the current state of the submission, and for the other pieces

to maintain the proper place in the workflow chain without the compute grid taking on that extra job.

Workflow in this system is utilized in a simple way. Jobs that need to be run concurrently can be simply described in the same XML instance document and translated into concurrent jobs by use of a script or other means. This solution is employed in Chapter 4's deployment of a simulations job which can conditionally be combined with a post-processor and then an analysis package. Input data is moved from tape at Jefferson Lab and put through a simulations program. Post processing may or may not be applied to the data, followed perhaps by the analysis package. The stylesheets simply create a Perl script which passes the filename of one program's output into the parameter specifying the next program's input depending on which of the conditional steps have been taken, achieving a kind of trivial workflow. That is, this workflow only passes data from task to task while residing on one site and in one step of computing, execution of the script. This workflow occurs implicitly, since the machine is only following the order of commands given. It does not understand them as separate steps in a process, which could enable it to find better sites to perform these steps, or facilitate any of the other benefits workflow can provide.

*Chapter 4*

# THE HADRONIC NUCLEAR PHYSICS GSIM SIMULATIONS CLUSTER AT FLORIDA STATE UNIVERSITY

**Introduction:** This section details the implementation of this system installed at the Hadronic Nuclear Physics computing cluster at Florida State University. First we describe the cluster, the data grid, and the physics applications which were published to the grid. Next, the steps which were necessary to install the system, populate it with users, and create, deploy, and publish physics functionality are detailed. This includes an example of the situation-specific rules and policies which were built into this application to illustrate a general capacity to create and enforce rules on any system. The last section details how the system is being utilized and what areas may be improved in the future.

**Batch System and Application Description:** The Hadronic Nuclear Physics cluster comprises 20 commodity computing nodes running under the control of a PBS batch system on Linux operating systems. A RAID array is available for data storage and file staging to handle the large amounts of data involved in running physics applications. The main node is also a grid node, or a node which has been enabled as part of the DOE Science Grid[29] project to take part in a data grid application known as SRM[30]. SRM is the Storage Resource Manager, and it can securely move data via restrictions on user credentials which are supplied by the DOESG in the form of certificates and proxies. This application was installed by Bryan Hess of Jefferson Lab and moves data from tape at Jefferson Lab to the grid node and also from the grid node to tape. The particular physics applications which were deployed to this compute grid during the trial run were

[29] http://doesciencegrid.org/

[30] http://sdm.lbl.gov/srm-wg/

GSIM, GPP, and A1C[31]. GSIM is a specific casting of GEANT[32] for the detector geometry in Jefferson Lab's Experimental Hall B, host of the CLAS[33] detector. GEANT was developed by CERN to simulate the response of detectors to elementary particle and nuclear physics reaction products. The common setup in nuclear physics experiments in Hall B is a high energy beam of electrons or photons which collide with heavier matter, producing other elementary particles after possibly passing through interesting intermediate states. GEANT simulates the interactions with detectors to produce signals of the same form as those produced by the actual experiment, so that the two sets of events can be compared to evaluate the appropriate error in the calculated result. GPP was developed by Kyungseon Joo to be run after GSIM to correct for temporary imperfections in the detector. For example, GPP uses GSIM output to remove dead detector wires and burnt tubes as well as to simulate noise and background effects. A1C is an analysis package run on Hall B data to reconstruct the reaction information (particles, momenta, energies) from the detector responses.

**Installation and Use:** This system was first installed following the instructions of Appendix A in January 2003. An upgrade in Jetspeed version was performed in March and was necessary so that the same portlet could be added multiple times to a page, among other features. The batch system where the web service server resided runs behind a firewall, and so only connections from the machine where the portal ran, outside the firewall, were allowed in order to maintain the security. Root runs and owns the portal on the machine where it is running, and a special user was created to run the web service application inside the firewall. This was necessary because all scripts are executed under the ownership of the process which created them and root cannot run jobs on PBS.

---

[31] http://clasweb.jlab.org/offline/utilities/a1/a1_docs.html

[32] http://wwwasd.web.cern.ch/wwwasd/geant/

[33] http://www.jlab.org/Hall-B

The first jobs were submitted in February, while the first useful jobs incorporating the data grid application SRM were submitted in May. GSIM, SRM, GPP, and A1C were all described with Schemas, XML Instance documents, and XSL stylesheets and deployed to the portal following the steps given in Appendix C. The stylesheets were designed in two ways to show the flexibility of the approach. One way simply creates a PBS batch script full of PBS parameters and the application command, allowing the web service to directly submit the script with the PBS command qsub. The second is more complicated, since the batch system administrator wanted to enforce a rule to load and boost all data from the web using only one node, leaving the other nodes hidden safely behind the firewall. In this case the XSL stylesheets were written to compose a Perl script which created a batch script, loaded all data onto the batch system's RAID array, submitted the job once the data was ready, moved the resulting data back to tape at Jefferson Lab after waiting for the job to be done, and cleaned up all created files so that the file system would not be cluttered after the job was done. The web service executed the Perl script in this case. Since the RAID disks were visible to all batch nodes, input card and monte carlo data files were easily accessed by the running jobs.

The text output method which allows the web service to generate scripts ensures that many different types of system rules can be created to enhance and protect grid functionality. XSL can simply serve as a pipeline for transporting information encoded in XML documents into scripts designed by batch system administrators to enforce rules which protect the security of the systems they employ. Other administrators may wish to enforce rules which allocate only a certain percentage of their deployed resource to grid users, while leaving the rest for local use. Situations where information can be shared also exemplify important applications of physics data grids, illustrating rules that can be made about not only bringing the data back to a user, but shipping it to other interested parties, or any other use to which a local data grid can be applied.

**System Use and Future Improvements:** The system has been deployed to the physics community at Jefferson Lab which is interested in using the FSU cluster for simulations. The Jefferson Lab cluster discourages simulations jobs since they usually take up to eight hours to complete, in addition to usually requiring the simultaneous completion of many jobs at one time. For these jobs a batch system's resources become tied up quickly. However, the grid resources at FSU can be employed in a completely transparent manner as a substitute for running jobs on the JLAB cluster. FSU currently offers simulations service to approved JLAB grid users, so that the batch system is rarely idle while serving the simulations needs of the CLAS community. Since the resource is open to serve the simulations needs of the CLAS community, and job monitoring tools help scientists to visualize the availability of these resources, the deployed resource system efficiency is raised by grid users requesting as many jobs as the resource can service.

*Chapter 5*

# CONCLUSIONS

The general problems of high data volume management and the associated computing jobs involved in processing and analyzing this data contribute much to the complexity of the physics research environment. It was shown above that a significant solution for these problems comprises compute and data grids when combined with portals. In the system design, an open source Java portal hosted an application which communicated with a web service server, incorporating many other open-source projects, to form a highly functional compute grid. To deploy physics applications to this grid, the system consumes XML, Schema, and XSL Stylesheet documents written by physics experts and administrators, providing both logical consistent structure and several levels of security. This project created the first data and compute grid between FSU and JLAB, a system which is being utilized by CLAS research groups for CLAS nuclear physics simulations, post-processing, and particle track reconstruction analysis, showing that the system can be used for a broad range of applications. This system is also one of the first applications of a structured grid to be deployed to a physics environment, and the advantages of this approach were developed and studied. A timeline of my activities in carrying out this project is available in Appendix D.

Each project design goal has been addressed and solved in some way. The first requirement focused on building a structured grid with low barriers for a physicist to use, build, and extend. To solve it, the combination of grids and portals was examined and determined to have great potential. Portals are easy to learn to use and are highly conducive to features such as personalization that make each use easier than the last. Grids allow access to resources and services in

an open and transparent manner impossible to achieve by any other means, in line with the general grid and web service goal of achieving complete interoperability among applications over the web. Heavy use of open source web application technology such as the portal and its many component technologies enabled a system to be created with many layers of functionality which can both evolve over time and extend the lifespan of the original system.

The thrust of the second goal was to provide a way to modularize system control of security and resource utilization, so that physicists can deploy grid applications while administrators publish them to the community of users. This concern was answered by the creation of portlets whose configuration and individual parameters create addresses to be used by web services, so that a structured grid is maintained by portal administrators who also are able to help in writing the XML files which deploy physics applications to this compute grid. The concept of a structured grid was applied and incorporated successfully into the design of this system through this feature. Portal administrators are also for the most part responsible for publishing these services within the portal, so that physicists from the deployment stage only need to provide input if they have any specific concerns. The open source systems employed, from main applications to minor services, are continually being developed, so that their functionality can be enhanced and reincorporated with a minimum of physics resource investment.

The third design goal requested both a broad range of applications and high application flexibility to operate within the grid. Incorporation of and the central role given to XML and its related technologies provide the flexibility to create languages of well-defined meaning to both man and machine, increasing the lifetime of both this system and its deployed applications. The system is not specific to any area of physics, only those which require prohibitively large amounts of computing resources for completion. When the applications are upgraded or otherwise changed, these changes may be made to the application-defining documents at any time without the system missing a beat. Further,

upgrades or enhancements to the web service do not for the most part affect its availability to the system. Thanks to the web service server, large extensions to the service may be developed without alteration to the underlying framework.

The fourth goal carried an obligation to enable as much control over the deployed application as possible. In much the same manner as in the above goal, XML technologies related to the regulation and validity of XML related documents helped achieve the scope of control currently available. The deployers have full control over the applications and batch systems deployed by controlling the data structure, information content and validity, and literal translation to scripts at the end of the process. In addition to this, the portal has several built-in means of control, including the use of existing portal tools such as security roles for a user's group and permissions to further secure the deployed applications.

The multiple tools that are being developed both for this specific application by computational physics groups at FSU and for the general environment and tools which form this system's base by the open source community ensure not only long-term viability but the enhancement of physics community activity outside of this grid, encapsulating the fifth design goal as well. This project will be presented at the upcoming CLAS Collaboration meeting at Jefferson Lab, where we will introduce this project to a much larger community of users. I am working with Dr. Riccardi on a job submission history feature which will operate based on web service messages to a database. This feature should be able to support applications such as a portlet job submission history, in which the last X jobs of a certain type are displayed, and other applications which draw on the messages, documents, and scripts which this database will store. One application would be for all job state information to be kept in this database, so that a workflow engine could simply use it as a client to keep track of what phase each job currently occupies. Other computational physics groups, like the one headed by Paul Eugenio, will continue to develop the project in directions

that will be of most use to physicists, with a focus on quickly creating functionality focused on serving the needs of the physics community.

A working system implementation of this structured grid was installed at the Hadronic Nuclear Physics computing cluster at Florida State University in connection with users and data from JLAB's CLAS community. A twenty-node batch processing farm was securely made available for use first in simulations and later in post-processing and track reconstruction analysis of nuclear physics data. A data grid in use between JLAB and FSU was employed to complete the simulations deployment. The system was installed on a Physics Department personal computer using the version of Apache which came with its Linux distribution. Situation-specific rules and policies were embedded into this application to keep all batch nodes but one closed to HTTP while still allowing the data grid to bring physics event files from JLAB over HTTP to each job. This shows a general capacity to create and enforce many rules on a generic system.

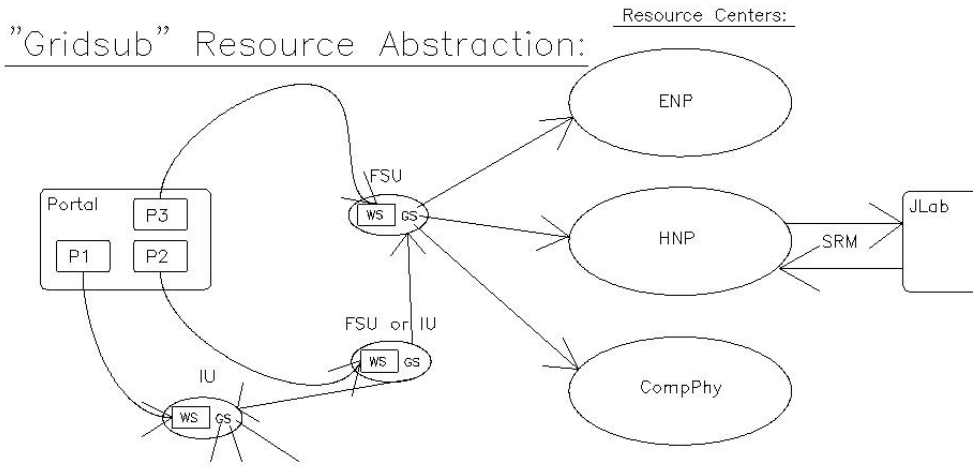


*Figure 5.1 Gridsub.*

*Appendix A*

# INSTALLATION, START UP, TROUBLESHOOTING, AND UPGRADES

**Introduction:** This Appendix will first show how to install both sets of software, the client and the server, and then provide instructions on how to start things up and troubleshoot some problems to keep things running smoothly. A tentative how-to on performing a Jetspeed upgrade is also included.

**Installation:** Both systems follow the same installation procedure up to a point. Begin by downloading an Apache HTTP server. This may the easiest step since most new Linux distributions come with it already installed. However, there are different versions of Apache and also different install procedures for Windows operating systems than for Linux. I could never get the Windows way to work properly, so in what follows I will only treat the Linux case. In this case the Apache HTTP daemon is invoked usually by the command /etc/init.d/httpd plus an argument like stop, start, or restart, and the configuration files reside in /etc/httpd/conf. Refer to Apache documentation to correctly configure the web server for your domain and IP, and then the process of connecting to Tomcat can begin. The first thing to do is to find the library in this software distribution which allows the connections and then deploy it to the modules directory in the Apache root directory. You can find the file mod_webapp.so inside the Tomcat home directory of this system's distribution. The next step is to reconfigure the Apache configuration file to recognize the new library and make connections between Apache and the Tomcat engine. Add a line like this at the end of the section in %ApacheRoot%/conf/httpd.conf where there are several LoadModule statements clustered together:

LoadModule webapp_module modules/mod_webapp.so

Again where there are many AddModule statements, add:

AddModule mod_webapp.c

Then at the bottom of the file the connections can be created like the following:

WebAppConnection *connection_name* warp *server_name:port*

Following this several lines which deploy individual applications can be made like:

WebAppDeploy *application_name connection_name path_name*

In the WebAppDeploy lines the connection name must be the value created in the WebAppConnection line. Warp in the Connection line refers to the type of connector bridging together Tomcat and Apache, while the server name and port is the name and port that Tomcat can be configured to listen for as it waits for requests to be forwarded its way. In the Deploy lines, the application name refers to the name of a web application in Tomcat's webapps folder, located at %TomcatRoot%/webapps. The path name refers to the path part of the URL that Apache will serve. This allows the actual file system to be masked to help divert hacker attacks. At this point, Apache is aware of web applications contained by a Tomcat container and has the code needed to make connections to them. Tomcat configuration is the next step and fairly straightforward.

Suitable Tomcat distributions are included as part of the system distribution, and the Axis server and portal server are already in place within their own Tomcat distribution. Before they can start to run they need a version of Java installed on the system. Any version will be fine, especially the latest ones since none of the ones I have tried have broken the system in any respect. Once they are installed, point to the root directory of the installation in the file %TomcatHome%/.bashrc to set the environment. Of course there are other ways to set the environment, and so long as the variable names stay the same they are equivalent for the purposes of this install. The Tomcat configuration files have already been altered from their original state so that only minor amounts of information need to be changed to allow the system to run on the new site. All XML belonging to the localhost setup has been removed from the configuration

file, %TomcatRoot%/conf/server.xml, and only that which corresponds to the Apache connection remains. In all this XML, only one item needs to be changed to just get the system up and running, the port number in the Connector element under the Service element registering Tomcat-Apache. The port number to be used corresponds to the port number written into the Apache httpd.conf file in the WebAppConnection line. This port is used as an internal redirection port, where Apache will forward requests it receives for the applications named in WebAppDeploy lines to the port where, by virtue of its configuration, Tomcat will be listening. Tomcat will then call the appropriate application, service the request, and return pages to Apache. At this point, Tomcat is fully configured for its new site and ready to start running.

**Startup:** Starting both of these engines is a fairly simple process once you are assured the installation is completed correctly. Simply set the environment variables with the scripts provided or some other method, enter the %TomcatRoot%/bin directory and run startup.sh, and then restart the Apache server with the command /etc/init.d/httpd restart. This ensures immediate pickup of Tomcat by Apache. Both Tomcat engines containing the applications are started the same way. One caveat can be offered about startup, which is to pay attention to which user is starting up these machines. Since the Axis server is submitting scripts that start jobs, you may not want to have root do this. PBS for example does not allow root to submit jobs. However, there is most likely nothing wrong with root owning and starting the portal, and that is the policy I employed at the FSU site. This is just an extra caution which should also be taken into consideration if the startup of these engines is written in to the computer's initialization routines which run as its power is turned on and the operating system starts up. Finally, at this point each Tomcat engine is running and has initialized all applications in its webapps folder. Only the applications deployed in the httpd.conf file are available for service, since Tomcat's own excellent standalone HTTP server has been disabled by removing it from the

server.xml configuration file. The Axis server has no properties that care about any site specific information at its new home. However, the portal still needs further configuration to enable some features and services at its particular host.

**Additional Considerations:** The Jetspeed directory %JetspeedRoot%/WEB-INF/conf holds many configuration files. One is for Turbine properties, one for Jetspeed properties, and one for properties which affect this particular application. The ones that need to be changed are mostly just server names. In TurbineResources.properties, the property mail.smtp.from should be changed to an appropriate email address for a person like an administrator who can answer concerns about email the portal sends out. At the bottom of this file are several include statements for other property files. If you wish to add your own property file which overrides any other properties set in any of the others, simply add the file to this list. The rest of the properties in this file deal with system policies and service configurations and as such they are all very important to running a good system and deserve careful attention. A good example is the properties on lifetime of certain portlets and other resources in portal cache, which can determine how fresh the portal's content stays. However, these properties do not affect the main features this system provides for physics grid computing. The excellent Jetspeed and Turbine online documentation and community support can supply explanations to these and many other concerns dealing with the general setup of the system. The next file to consider, JetspeedResources.properties, must be customized in much the same way.

This file contains properties which apply to Jetspeed as its own application. Remember that Jetspeed is built on top of the Turbine framework, so some Turbine properties may be superceded by Jetspeed properties, and some Jetspeed applications may not be mentioned at all by any corresponding Turbine instructions provided in the documentation in these configuration files. To override these properties I created another file I called JLabJR.properties which listed some of the same properties as JetspeedResources.properties but had

appropriate values for my particular system. The file begins with some system properties on how often the disk cache daemon should check for updates to files in its cache and similar concerns. Following that is a configuration of the Jetspeed email system which enables the features of email account requests, account acceptance or rejection notification, and password reminders. These set the templates to use for each type of email which is exchanged with the user and other properties like the applicable return address and title of the administrator, as well as the server name running the email daemon to which Jetspeed will try to connect to conduct all this email traffic. The last set of parameters specifies exclusively parameters for web services and physics grid computing. After setting the new application name under the property webservice.application.name, others may be changed much later as the new administrator gains understanding of the ways and means by which the system works. The two most important are the namespaces for interactivity and job submission. The portal depends on understanding XML from these namespaces to provide its main functionality, but these parameters are included so that other namespaces which extend the main namespaces can be used. Simply put, the portal looks for certain information in the form of specific tags and attributes and if it does not find them it does nothing. However, this basic framework can always be extended by adding new information in new structures as long as new code is added, and these parameters provide for that situation.

Other files of note which are involved in the cultivation of a customized application include the data sources, security definitions, and so on. Every file in the conf directory allows for custom structures to be built and seamlessly integrated into the portal, from new skins and security roles to lists of XML streams which may be sources of information for particular portlets an administrator may create. Consulting the mainstream Jetspeed documentation and the extremely helpful development community will prove invaluable to any effort in developing a particular portal's identity.

**Troubleshooting:** Sometimes the portal completely crashes and the reason why it happened may not be immediately obvious. Perhaps you were testing some new code, perhaps there was a database error during an upgrade, or maybe there is an even more subtle answer. The way to get around this is to try to return to a state where things were working correctly, and be sure to clean out Tomcat's %TomcatRoot%/work/Apache/*server_name* directory. This stores versions of the servlets in persistent storage at the container level, and if it is not erased then the old broken servlet contexts will keep breaking your application, even if you have fixed the problem that broke everything in the first place.

**Upgrading:** Performing a Jetspeed upgrade is a complicated procedure. This is one area where it definitely helps to have a separate development environment from the main production environment, so that if things break during a test upgrade the system in use does not also implode. The high impact newly integrated features of these new versions make the trouble worth experiencing more often than not. One step is to move all files from one version to another. This includes template files in WEB-INF/templates, configuration files in WEB-INF/conf, class and library files in WEB-INF/classes and WEB-INF/lib, XSL stylesheets in WEB-INF/xsl, XML and GIF/image files in %JetspeedRoot%/xml and %JetspeedRoot%/images and PSML files in WEB-INF/psml. Every file that customizes your particular application should be transported. The harder part of this is how to move the users from the user database in one version to the next. The procedure that usually works for me is to start up the new version without the new users first. Then shut it down, leaving the sign-in screen open in a browser window. Next, open the file WEB-INF/db/jetspeed.script in the new version and try to introduce all the information from the file in the old version, following any syntax changes that seem to have taken place. This script starts up a database of Jetspeed objects and is based on Turbine notions of users and other system objects. This database method also has its roots in a corresponding Turbine method. Once the script

looks complete, start the Tomcat engine up and sign in using the existing browser window as any user who existed before the upgrade, preferably an administrator. Then you should be able to sign out and sign back in as the new users. If anything goes wrong, you are likely to receive a Turbine Horrible Exception, and you would need to follow the procedure outlined in the troubleshooting section to recover and start over.

Upgrading the other components is less complicated. Tomcat upgrades are usually unnecessary since we only expect Tomcat to perform a limited range of functionality and are not currently interested in complicated applications of its more advanced features. New versions of Axis should be pluggable in Tomcat in place of the old, requiring only redeployment of the web service to its new server. Many services in Jetspeed are actually provided by pluggable separate open-source projects, and it is recommended that their manual upgrade is not attempted because of the existing complex portal infrastructure. New versions of significant difference can be incorporated by Jetspeed developers during milestone builds, so changes to these should be reserved for experts only.

*Appendix B*

# PORTAL INTERFACE HOW-TO

**Introduction:** One of the first questions a new subscriber to a portal may ask is simply how to navigate in the new environment. Since portals specialize, among other things, in providing members with high content and layout flexibility, one of the first subjects to which a new user should be introduced is how to take advantage of these features. During the initial deployment of the HNP portal, this was one of the first areas of improvement to be suggested. Many excellent Jetspeed tutorials on these and other topics, including building custom applications, can be found, starting from the official Jetspeed website itself. However, much of what has been written assumes a more expert web application background than should be expected of a new user, so this appendix is dedicated to providing a simple explanation on basic portal actions.

**How to Create a New Account**: The first page a user sees in Jetspeed is the page belonging to the anonymous user, created to show a glimpse of the portal without allowing any permissions on that content. The top right corner of the screen displays two blanks to sign in with username and password, and below it are two links which allow someone to create a new account and be reminded of their password. To create a new account, first follow that link and fill out the fields in the resulting form. If the portal email features have been configured correctly, submission of this form will send an email to a portal administrator at an address determined in portal configuration. The email will contain a link that the portal administrator can use to bring up the page which displays all users, both existing and pending, once the administrator has signed in. This portlet can be found under the security tab of the portal administrator's main account. The first portlet displayed is the User Browser, which holds a list of all users with

potential users highlighted in yellow. To approve a user, click the Edit link by their name and check the approve box. To deny, check the other box. A user can be removed by following the remove link located across from the name in the User Browser, but denying membership does not remove the user, so that admittance may be granted after membership has been denied. Acceptance will generate an email to the address provided in the account request and the new user may subsequently sign in. Reminder of password may be achieved by following the other link at the top right of the anonymous page and typing the account's email address in the field before submission of that form.

**How to Create a Personalized Page:** Once a new user signs in, they see a set of default pages which have been created by a portal administrator. These pages are set apart by tabs at the top of the screen. To simply create a new page, click the pencil icon at the far right of the tabs, and hit the Add Pane button in the resulting page. Type in a name for the new pane and hit the apply button at the bottom of the next two pages. Click on the link the new name appears in and a blank page should appear.

Adding content requires that much of the same procedure be followed. The pencil icon on the tab of the new pane will display the current content along with its layout, which should be blank for the new page. The Add Portlet button at the top of this page should display a list of portlets with checkboxes. Click a box to add a portlet to the page. Thanks to the last Jetspeed upgrade, an individual portlet can be added to the same page many times. If a portlet is already on the page, a blue checkmark will appear next to the portlet title. After making a selection of portlets, click the Apply button and the layout page will be presented again. The arrows in the upper right corner of each portlet, represented by a box, will move the content around the page. Click Apply once more and the new page will have new content arranged to the preferences of the user.

**Additional Features:** Once these basic actions have been mastered, a new user has the ability to move on to more advanced portal use. This is recommended, since these features enable interaction and customization which are entirely new to web applications. Available features include skin (color and presentation scheme) customization, security settings, role and group creation and permission association, portlet groupings and layout personalization. Whether a portal administrator, a regular user, or both, the advanced tutorials provide a wealth of information for those interested in enhancing a particular application for personal and community use.

*Appendix C*

# HOW TO DEPLOY AN APPLICATION

**Introduction:** An application may be deployed to this physics grid by following the steps outlined above in Chapter 3. However, the XML documents whose creation comprise most of the deployment require additional explanation, since they have particular structure and meaning to this compute grid. In the following I first provide an overview of the design together with the motivation for following through with this particular solution to the problem. Then I describe the structure and functionality of the XML I created which addresses the above concerns. This description contains a working example as shown in the figures which is offered in addition to the examples which come packaged with the distribution of the code. Since I offer writing these XML files as substitutions to writing web service code, any interest in deploying an application to the grid must first begin with an introduction to XML and its related technologies, Schema and XSL. This tutorial is not intended to substitute for the many excellent books which can be found by now at any bookstore on these topics, and a basic understanding is recommended before proceeding. However, I have also provided examples of these files which are capable of servicing a wide range of physics applications after only introducing minor semantic permutations for the purpose of getting a new user quickly started.

**Overview:** In order to create meaningful interaction between the portal interface and the job submission web service, I created XML languages to describe each aspect of the job submission process. These layers of XML are interpreted by the code which runs each part of the compute grid, and so they dictate the functionality which will be employed each time the system is used. As I describe the XML layers and elements implemented in the portal and web service in detail,

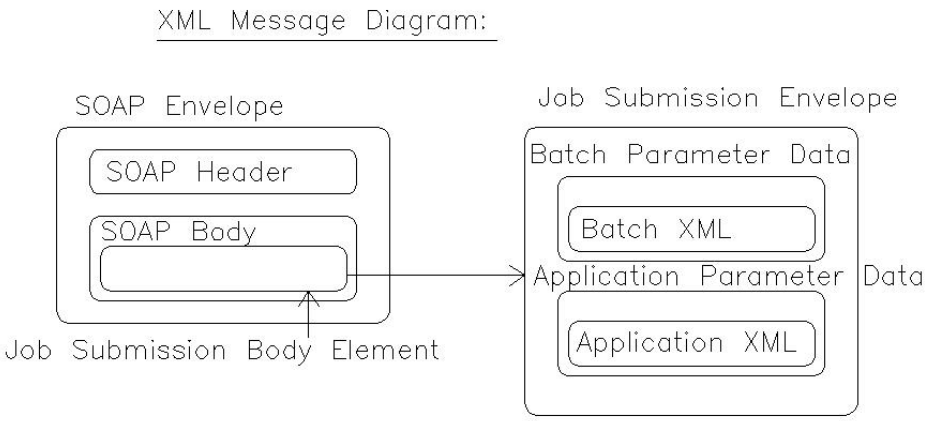


*Figure C.1 XML Message Diagram.*

I would like to first note the motivation for this design. Simply put, these layers I created act as very general flags to the application, which looks for specific tags and attributes to describe job parameters and construct interactivity. A concern may be expressed that it would be preferable to allow any markup to perform these roles I have described above, rather than only markup from one of my namespaces which matches in form the elements I created. However, steps have been taken to ensure that the framework is flexible for extension in functionality and form, by following the general model set by SOAP XML and WSDL XML. These widely accepted languages provide a framework for their respective purposes while allowing as much XML content, and therefore as much information in as many ways, as possible. Also, any content with one of my namespaces that matches in form can be substituted without the application losing any functionality, since the same namespace can be attached to different Schemas.

**Job Submission XML:** Each instance XML document contains a root element of <jobSubmission> subsequent to the XML declaration. The root element contains two child elements: <batch> and <application>. These sections

contain respectively batch and application parameters encoded in any XML language which has been attached to a namespace in the root element. Each of the two child tags has a required stylesheet attribute whose value is a URL where the stylesheet which translates each fragment of XML resides. Additionally, the batch element has a command attribute whose value is the command that the web service should use to execute the script on the file system where it resides.

**Job Interactivity:** The second namespace I created is the interactivity namespace. This namespace is intended to hold XML that describes to the portal how an interface describing any physics application deployed to the grid should be structured. Interaction can be described with <examples> and <description> tags placed as appropriate with respect to the schema regulating the instance document. Also, Boolean attributes such as isParameter can be placed on elements which are children of <batch> and <application>. Once the portal application discovers a tag with isParameter="true", it looks for child tags with the attribute isValue. If that value is set to true, the application looks for a parameter name contained by the same tag in the attribute valueName. It will take the value of valueName and create a parameter in the portlet interface with that name. Whatever value is entered into the field of this parameter in the portlet interface will then become the character data of the element with isValue="true". Note that one tag with isParameter="true" can have both multiple value children and also contain isParameter="true" children, allowing for subtrees of parameters to be described.

A sample XML instance document fragment looks like the following. The instance document defines namespaces and specifies their location in the root element, and then goes on to display a simple structure defining a few parameters for the PBS batch submission system and the common physics application GEANT. In both the schemas and the stylesheets, the ability to use hierarchy to define data types and rules for processing elements adds to the robustness of description and functionality of the system. Data type libraries can

be constructed for different kinds of parameters and layers of stylesheets can be developed for different methods of script construction that can significantly reduce the complication in deploying new applications by building on the past work of others. Schemas describing both custom and generic layers can be found at the addresses presented in Figure C.1, as well as stylesheets which translate them according to the appropriate attributes. These Schemas and stylesheets were written according to the rules and examples which are widely distributed across many bookstores and the Web. The simplest way to change these documents to reflect a particular application is to avoid changing the document's logical structure. Copy and paste interesting and useful definitions and rules and then change element and attribute names. More complicated applications can be created after more careful study of the capabilities of these XML technologies, but the preceding instructions are sufficient for beginners to use in deploying physics applications to this compute grid.

```
<?xml version="1.0" ?>
- <js:jobSubmission xmlns:js="http://www.jlab.org/~hone/jobSubmission"
    xmlns:interact="http://www.jlab.org/~hone/jobInteractivity" xmlns:history="http://www.jlab.org/~hone/jobHistory"
    xmlns:pbsParams="http://www.jlab.org/~hone/demo/pbs" xmlns:demoParams="http://www.jlab.org/~hone/demo/demo"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.jlab.org/~hone/jobInteractivity
    http://www.jlab.org/~hone/jobInteractivity.xsd http://www.jlab.org/~hone/jobSubmission
    http://www.jlab.org/~hone/jobSubmission.xsd http://www.jlab.org/~hone/jobHistory
    http://www.jlab.org/~hone/jobHistory.xsd http://www.jlab.org/~hone/demo/pbs
    http://www.jlab.org/~hone/demo/pbs.xsd http://www.jlab.org/~hone/demo/demo
    http://www.jlab.org/~hone/demo/demo.xsd">
  - <js:batch js:stylesheet="http://www.jlab.org/~hone/demo/PBS.xsl" js:userID="" js:command="qsub">
    - <pbsParams:mail_options interact:isParameter="false">
        <interact:examples>n or a, ab, e, abe...</interact:examples>
        <interact:description>Options associated with job notification email. n=none,a=on job abort, b=on job beginning, e=on
          job ending.</interact:description>
        <pbsParams:mailOptionsData interact:isValue="true" interact:valueName="Job Email
          Options">eab</pbsParams:mailOptionsData>
      </pbsParams:mail_options>
    - <pbsParams:mail_list interact:isParameter="true">
        <interact:description>This is a comma-separated list of email addresses. These email addresses will be notified upon job
          completion.</interact:description>
        <interact:examples>joshhone@yahoo.com,hone@jlab.org</interact:examples>
        <pbsParams:mailListData interact:isValue="true" interact:valueName="Email Address List" />
      </pbsParams:mail_list>
    </js:batch>
  - <js:application js:stylesheet="http://www.jlab.org/~hone/demo/Demo.xsl">
    - <demoParams:command interact:isParameter="false">
        <interact:examples>printenv</interact:examples>
        <interact:description>The only command that is valid is "printenv"</interact:description>
        <demoParams:commandData interact:isValue="true" interact:valueName="Command to be
          Executed">printenv</demoParams:commandData>
      </demoParams:command>
    </js:application>
  </js:jobSubmission>
```

*Figure C.2 Demo XML Deployment file.*

```
<?xml version="1.0" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.jlab.org/~hone/demo/demo"
    xmlns:tns="http://www.jlab.org/~hone/demo/demo" xmlns:jobParam="http://www.jlab.org/~hone/jobParameter"
    xmlns:paramTypes="http://www.jlab.org/~hone/jobParameterTypes"
    xmlns:interact="http://www.jlab.org/~hone/jobInteractivity" elementFormDefault="qualified" attributeFormDefault="qualified">
  - <xs:annotation>
      <xs:documentation xml:lang="en">Job submission schema for FSU, JLab, PPDG, etc.</xs:documentation>
    </xs:annotation>
    <xs:import namespace="http://www.jlab.org/~hone/jobParameter"
      schemaLocation="http://www.jlab.org/~hone/jobParameter.xsd" />
    <xs:import namespace="http://www.jlab.org/~hone/jobInteractivity"
      schemaLocation="http://www.jlab.org/~hone/jobInteractivity.xsd" />
  - <xs:complexType name="DemoParamDataType" abstract="true">
    - <xs:simpleContent>
      - <xs:extension base="xs:anySimpleType">
          <xs:attribute ref="interact:isValue" use="required" />
          <xs:attribute ref="interact:valueName" use="required" />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
    <xs:element name="command" type="tns:CommandType" />
  - <xs:complexType name="CommandType">
    - <xs:complexContent>
      - <xs:extension base="jobParam:JobParameterType">
        - <xs:sequence>
            <xs:element ref="tns:commandData" minOccurs="1" maxOccurs="1" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:element name="commandData" type="tns:CommandDataType" fixed="printenv" />
  - <xs:complexType name="CommandDataType">
    - <xs:simpleContent>
      - <xs:restriction base="tns:DemoParamDataType">
        - <xs:simpleType>
          - <xs:restriction base="xs:string">
              <xs:pattern value="[\S]*" />
            </xs:restriction>
          </xs:simpleType>
        </xs:restriction>
      </xs:simpleContent>
    </xs:complexType>
  </xs:schema>
```

*Figure C.3 Application Schema.*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:js="http://www.jlab.org/~hone/jobSubmission" xmlns:interact="http://www.jlab.org/~hone/jobInteractivity"
    xmlns:demo="http://www.jlab.org/~hone/demo/demo" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.jlab.org/~hone/jobInteractivity http://www.jlab.org/~hone/jobInteractivity.xsd
    http://www.jlab.org/~hone/jobSubmission http://www.jlab.org/~hone/jobSubmission.xsd
    http://www.jlab.org/~hone/demo/demo http://www.jlab.org/~hone/demo/demo.xsd">
    <xsl:output method="text" indent="yes" omit-xml-declaration="yes" media-type="text/plain" />
    <xsl:strip-space elements="js:application" />
    <xsl:include href="http://www.jlab.org/~hone/grid/webservices/xsl/JobInteractivity.xsl" />
  - <xsl:template match="/">
      <xsl:apply-templates select="child::js:application" />
    </xsl:template>
    <!--  Here are the rules that describe how the script is to be formed from input XML.    -->
  - <xsl:template match="js:application">
      <xsl:value-of select="child::demo:command/child::demo:commandData" />
    </xsl:template>
  </xsl:stylesheet>
```

*Figure C.4 Application Stylesheet.*

```
<?xml version="1.0" ?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.jlab.org/~hone/demo/pbs"
    xmlns:tns="http://www.jlab.org/~hone/demo/pbs" xmlns:jobParam="http://www.jlab.org/~hone/jobParameter"
    xmlns:paramTypes="http://www.jlab.org/~hone/jobParameterTypes"
    xmlns:interact="http://www.jlab.org/~hone/jobInteractivity" elementFormDefault="qualified" attributeFormDefault="qualified">
  - <xs:annotation>
      <xs:documentation xml:lang="en">Schema for PBS parameters. Valid data types are required for job
        submission.</xs:documentation>
    </xs:annotation>
    <xs:import namespace="http://www.jlab.org/~hone/jobParameter"
      schemaLocation="http://www.jlab.org/~hone/jobParameter.xsd" />
    <xs:import namespace="http://www.jlab.org/~hone/jobInteractivity"
      schemaLocation="http://www.jlab.org/~hone/jobInteractivity.xsd" />
  - <xs:complexType name="PbsParamDataType" abstract="true">
    - <xs:simpleContent>
      - <xs:extension base="xs:anySimpleType">
          <xs:attribute ref="interact:isValue" use="required" />
          <xs:attribute ref="interact:valueName" use="required" />
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
    <xs:element name="mail_options" type="tns:MailOptionsType" />
  - <xs:complexType name="MailOptionsType">
    - <xs:complexContent>
      - <xs:extension base="jobParam:JobParameterType">
        - <xs:sequence>
            <xs:element ref="tns:mailOptionsData" minOccurs="1" maxOccurs="1" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:element name="mailOptionsData" type="tns:MailOptionsDataType" default="n" />
  - <xs:complexType name="MailOptionsDataType">
    - <xs:simpleContent>
      - <xs:restriction base="tns:PbsParamDataType">
        - <xs:simpleType>
          - <xs:restriction base="xs:string">
              <xs:pattern value="(n|[abe]{1,3}){1}" />
            </xs:restriction>
          </xs:simpleType>
        </xs:restriction>
      </xs:simpleContent>
    </xs:complexType>
    <xs:element name="mail_list" type="tns:MailListType" />
  - <xs:complexType name="MailListType">
    - <xs:complexContent>
      - <xs:extension base="jobParam:JobParameterType">
        - <xs:sequence>
            <xs:element ref="tns:mailListData" minOccurs="1" maxOccurs="1" />
          </xs:sequence>
        </xs:extension>
      </xs:complexContent>
    </xs:complexType>
    <xs:element name="mailListData" type="tns:MailListDataType" />
  - <xs:complexType name="MailListDataType">
    - <xs:simpleContent>
      - <xs:restriction base="tns:PbsParamDataType">
        - <xs:simpleType>
          - <xs:restriction base="xs:string">
              <xs:pattern value="([^\s]*@[^\s]*)?" />
            </xs:restriction>
          </xs:simpleType>
        </xs:restriction>
      </xs:simpleContent>
    </xs:complexType>
  </xs:schema>
```

*Figure C.5 Batch Schema.*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
- <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:js="http://www.jlab.org/~hone/jobSubmission" xmlns:interact="http://www.jlab.org/~hone/jobInteractivity"
    xmlns:pbsParams="http://www.jlab.org/~hone/demo/pbs" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.jlab.org/~hone/jobInteractivity http://www.jlab.org/~hone/jobInteractivity.xsd
    http://www.jlab.org/~hone/jobSubmission http://www.jlab.org/~hone/jobSubmission.xsd
    http://www.jlab.org/~hone/demo/pbs http://www.jlab.org/~hone/demo/pbs.xsd">
    <xsl:output method="text" indent="yes" omit-xml-declaration="yes" media-type="text/plain" />
    <xsl:strip-space elements="js:batch" />
    <xsl:include href="http://www.jlab.org/~hone/grid/webservices/xsl/JobInteractivity.xsl" />
  - <xsl:template match="/">
      <xsl:apply-templates select="child::js:batch" />
    </xsl:template>
    <!--  Here are the rules that describe how the script is to be formed from input XML.    -->
  - <xsl:template match="js:batch">
      <xsl:apply-templates />
    </xsl:template>
  - <xsl:template match="pbsParams:mail_options">
      <xsl:apply-templates />
    </xsl:template>
  - <xsl:template match="pbsParams:mailOptionsData">
    - <xsl:if test="not(.='')">
        #PBS -m
        <xsl:value-of select="." />
        <xsl:text />
      </xsl:if>
    </xsl:template>
  - <xsl:template match="pbsParams:mail_list">
      <xsl:apply-templates />
    </xsl:template>
  - <xsl:template match="pbsParams:mailListData">
    - <xsl:if test="not(.='')">
        #PBS -M
        <xsl:value-of select="." />
        <xsl:text />
      </xsl:if>
    </xsl:template>
</xsl:stylesheet>
```

*Figure C.6 Batch Stylesheet.*

*Appendix D*

# PROJECT HISTORY

This project when it began in January 2002 had several goals, which are reflected in the project motivations found at the end of Chapter 1. Each feature that we desired was studied and appropriate technology was sought to supply a fitting solution for each problem. The first goal was simply to get a portal up and running and to understand how to learn about and use the new realm of software that the large collection of open source projects represented. Once I arrived at Jefferson Lab to spend the summer of 2002, I worked on just running the portal on my laptop and trying to get the basic features described in its documentation to work, and this was when I developed the job monitoring tools. Much of this was based on the experience I had working with XML, DTDs, and Schema at Jefferson Lab in the summer of 2001 under Mark Ito. I spent a lot of time asking Jetspeed developers and other experts like Chip Watson how to understand user interaction with the portal, how to use the portal environment to activate the portal's extra features, and what features would be most useful in a physics community. I also had to figure out how to create a link between Apache and the Tomcat engine in which Jetspeed resided. That June, I attended a conference of the Grid Computing Environment working group of the Global Grid Forum at Indiana University and presented talks on current FSU portal projects and what I had learned about using Jetspeed.

At the end of that summer and in the first part of the fall semester I wrote most of the job submission code and features. It was during this time in November when the first version of Axis was released, so with some help from Greg Riccardi I found out about its capabilities, downloaded and installed it, and wrote a first edition web service which would create scripts on my desktop. In

January I installed and populated the portal and web service on the HNP cluster data grid node at FSU. Later in the spring semester this was polished and it gained its final form, and subsequently the first job submission was made in April. The first useful job submission of a simulations job was in May, and a little after that the SRM data grid was integrated. June saw the completion of this thesis as well as further refinement of the simulations service to contain conditional steps such as the post-processing and the analyzer.

# BIOGRAPHICAL SKETCH

I was born in Norman, OK, USA, in 1980, and moved to Tyler, Texas in 1986, where I attended school from first grade through high school graduation at Robert E. Lee High School. I was accepted to Florida State University in 1998, attended on a National Merit Scholarship, and received a B.S. from the College of Arts and Sciences in Fall 2001, majoring in Computational Physics. Computational Physics is a unique program at FSU in which a student takes all required classes for a physics major and heavily minors in Computer Science and Math. Upon graduation, the student may pursue a Master of Science in Physics specializing in a Computational Physics project, and that was the basis for this project.