
Grid Computing: Techniques and Applications

Barry Wilkinson

CHAPTER 8 USER-FRIENDLY INTERFACES

CONTENTS

- 8.1 Introduction
- 8.2 Grid Computing Workflow Editors
 - Workflows*
 - Workflow Editor Features*
 - GridNexus*
- 8.3 Grid Portals
 - General Features*
 - Available Technologies*
 - Early Grid Portals*
 - Development of Grid Portals with Portlets*
- 8.4 Summary
 - Further Reading
 - Bibliography
 - Self-Assessment Questions
 - Problems

Oct 11, 2008

Do not distribute.

Copyright B. Wilkinson

This material is the property of Professor Barry Wilkinson and is for the sole and exclusive use of the students enrolled in the Fall 2008 Grid computing course broadcast on the North Carolina Research and Education Network (NCREN) to universities across North Carolina.

CHAPTER 8

User-Friendly Interfaces

The focus of this chapter is user-friendly graphical interfaces, which are preferable to command line interfaces for many non-Computer Science users. Such interfaces are important if Grid computing platforms are to become accepted to a broader community of users. The chapter concentrates upon two areas, generating workflows using graphical tools, and providing a Web-based portal to access Grid resources.

8.1 INTRODUCTION

Clearly users would like to see a simple convenient but flexible interface to a Grid platform. We are all familiar with Web interfaces to access the Internet for Internet-based activities. A GUI Grid computing Web-based portal user interface was introduced in Chapter 1 to describe the central activities of a Grid user. Many production Grids use such Grid portals, but also many Grid activities fall back upon the command line interfaces that were described subsequently in the book so far. As Brown et al. (2005) comment:

“Today, many demonstrations of Grids still use command line and batch mode processing. This is style of computation used since the first days of computing. It is difficult for a notice users to discern the difference between a supercomputing environment and a Grid computing environment. Both require the submission of a job, scheduling and execution of that job, and collecting the results at a later time.”

There is actually two issues in the above quote, namely using command line tools to access a remote computer, and running batch jobs but they are related in that the user is distant from the actual computation both in time and space and has limited interaction.

The author concurs with Brown et al. Often one gets the question. “What is new in Grid computing? We have been able to `ssh` into a remote machine to run a job since the early days of computing.” This is true. Issuing a `Globusrun-ws` command on the command line to submit a job and `ssh`-ing into a remote computer and issuing commands to run a job are very similar on the surface. It is actually significantly different underneath as the user does not personally log on a remote machine on the command line to run a Globus job, and we can point to a number of improvements in implementation in the Grid version including single sign-on, significant if one wants to submit jobs to different remote computers, and to do third party transfers, that is, to transfer a file from one remote computer to another from a third. However to do simple job submission and the more powerful things one can do in a Grid platform, including collaborative activities, so far still requires the user to issue commands in console window. Windows users especially are not used to even opening a console window to run application, let alone issuing complex commands. So it is very desirable to offer users a better user interface, the type of interface they have come to expect on personal computers with windows and pull-down menus, drag-and-drop, and double clicking on icons to launch applications. In this chapter, we will describe such graphical user interfaces (GUI’s). We will begin with graphical interfaces for workflows that were introduced in the last chapter and then move onto web based interfaces for general Grid actions such as submitting and monitoring jobs.

8.2 GRID COMPUTING WORKFLOW EDITORS

8.2.1 Workflows

A (computational) workflow describes a sequence of tasks that need to be done to perform an overall computational goal. Each task has inputs and outputs or results. The results of one task are used by other tasks so there are dependencies between the tasks. A simple workflow might be a series of sequential tasks as one might find sequential procedural programming. It is necessary to do each “task” in sequence and allow each task to complete before performing the subsequent task. In Condor (Chapter 7, section 7.2.2), such workflows were created in Condor using a directed acyclic graph, coded in a DAG file. However this a very primitive approach. Dependencies may be very complex. The results of multiple jobs need to be staged as input for multiple subsequent jobs and the inputs and outputs may not be immediately compatible and may require transformations to be acceptable. The tasks themselves may be significant applications. This is especially true for scientific applications. For example, bioinformatic applications might need to use multiple specialized programs to process data. Sometimes they access very large databases. Sometimes, they use visualization tools finally to view the results and assist in its analysis. In a Grid

computing environment, the databases these interlinked applications might be at different sites.

Grid computing has moved to using a Web service approach for its middleware and generally the distributed applications could be exposed as Web services. Then, the workflow becomes a form of Web services workflow. There are languages for constructing Web service workflows. IBM proposed the *XML Web Services Flow Language* (WSFL) targeted towards constructing Web service workflows for business applications in the early 2000's. Microsoft also proposed an XML business Web service orchestration language called XLANG, which is an extension of WSDL. Both have been superseded by the *Web Services Business Process Execution Language* (WS-BPEL or BPEL), an OASIS Standard named in 2004. WS-BPEL is still targeted towards Business applications

8.2.2 Workflow Editor Features

There have been a very large number of projects for creating workflow tools for Grid computing. A number of XML languages have also appeared for Grid computing workflows. Twenty-one Grid workflow projects are listed by von Laszewski (2006). A very detailed taxonomy of workflow editors can be found in (Yu and Buyya 2005). The following are some desirable features for the workflow editor:

- A workflow editor should provide the ability to construct workflow of local and remote components easily.
- Workflows lend themselves to pictorial representation and ideally workflow editors should have a graphical drag-and-drop interfaces.
- The components of the work flow should be able to grouped together into reusable components in other workflows.
- The pictorial representation should be separated from the concrete implementation.
- An XML workflow description is desirable.

Grid workflow editors with graphical interfaces include open source Kepler (Kepler project) and Taverna (Taverna project). Kepler has the broad focus of scientific workflow and not just Grid computing workflows, but can now handle distributed computing. Kepler leverages an open source scientific workflow editor called Ptolemy II developed at the University of California Berkeley, and uses the powerful Ptolemy II graphical interface (called Vergil) directly. Ptolemy is concerned with modeling concurrent processes in real-time embedded systems. Its development can be traced back to the mid 1980's and work on Ptolemy II began in 1996. Ptolemy II version 7.0 was introduced in 2008. In Ptolemy, the individual components of the workflow are called *actors* (from early work elsewhere). Actors communicate between themselves through messages. More details on Ptolemy can be found at (Brooks et al. 2008). Taverna is focused on distributed components (including services) running on local and remote machines. It has its own graphical interface.

In the next section, we shall focus on a Grid computing workflow editor called GridNexus, which has all the features listed above and is very easy to use. We use in our Grid computing classes.

8.2.3 GridNexus

GridNexus was developed at the University of North Carolina Wilmington in early-mid 2000's specially for constructing Grid computing workflows (Brown et al. 2005). As with Kepler, GridNexus is based upon Ptolemy II and uses its graphical interface. GridNexus can be used to construct simple arithmetic workflow including with iteration, but more particularly Web and Grid service workflows. It separates the display of a workflow from its execution by using an XML language called JXPL.¹ The user constructs the workflow by drag-and-drop and drawing actions on a "canvas" on the interface. Executing this workflow will first create a description of the workflow in the JXPL language, The JXPL file so produced is then read and executed by a JXPL processor. When the workflow has Web and Grid services, these components will be accessed from wherever they reside through their URL with a given WSDL interface, as illustrated in Figure 8.1.

The GridNexus graphical interface (derived from Ptolemy II) has a top command ribbon and three frames, two on the left and a main drawing frame on the right. The upper left frame displays a library of modules that can be used in the workflow. The lower left frame displays a thumbnail of the actual workflow being constructed on the right frame (a palette or canvas). Users will select modules and drag then onto the right frame. Once components are placed on this frame, components, outputs can be dragged to inputs. The outputs and inputs will depend upon the modules selected. Many of the graphical user interface features are inherited from Ptolemy II including some terminology, onto which some new features for a Grid

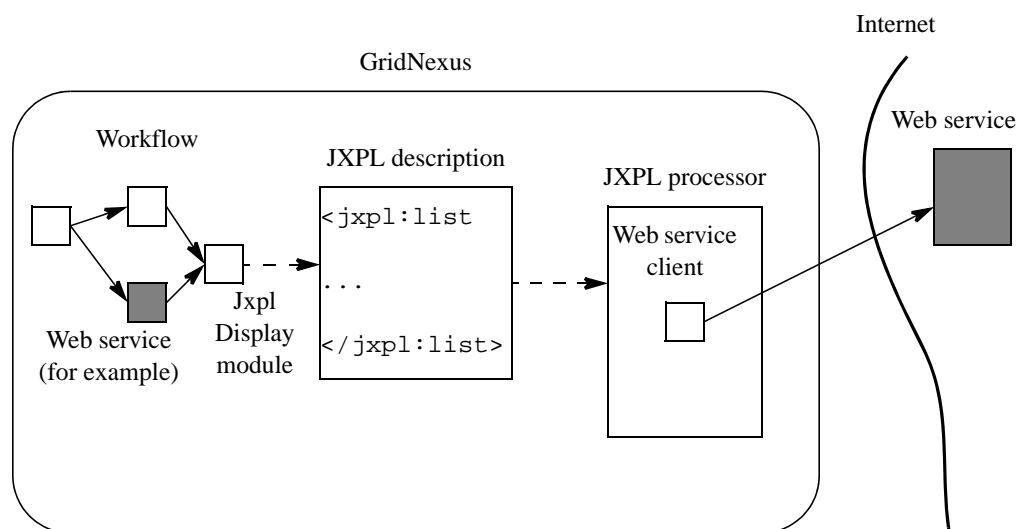


Figure 8.1 Gridnexus accessing remote service. **Check**

¹ JXPL appears now not be an abbreviation for anything although it may have originally have been.

computing environment and in particular the separation of the workflow from its execution with the use of an XML scripting language (JXPL).

Modules in the workflow are categorized as *sources*, *transformers* that convert input value(s) into an output, *sinks*, and various library modules. Many of the basic components are derived directly for Ptolemy II. The constant source module provides a constant value, which can be set by the user (double clicking the constant component of the workflow.) Addition and multiplication together with subtraction, division, modulo, and arithmetic are some transformers.

JxplDisplay is a GridNexus-specific sink module for creating the JXPL script for the workflow and causing it to be executed. A particular feature of Gridnexus is the generation of a JXPL script from the workflow which is decoupled from the actual execution of the workflow. The JXPL script without it being executed can be obtained, perhaps for debugging, by double-clicking the `JxplDisplay` module and deselecting “Evaluate Jxpl”.

A workflow can be created in a so-called *composite* (which comes from Ptolemy). A composite enables a workflow to be considered as a single entity and used in other workflows. A composite will have named inputs and outputs and an internal workflow that uses the inputs and produced the outputs. A user can save composites in library.

Other features include certain appropriate inputs can be *multi-port* inputs that allow more than one source for the input to be merged together. Multi-port inputs would be applicable for such inputs as argument inputs, which could have multiple arguments. Outputs can be drawn to connect to more than one input, broadcasting the output. New functions can be defined. Iteration and recursion can be represented. A LISP-like branching mechanisms is implemented in a so-called *cond* module, which finds the first list element which is true and returns that result. The *prog* module (program module) is used to execute separate workflows without a data dependency between them. It evaluates each input in turn and returns the result of the last one evaluated.

Gridnexus is provided with modules for Grid computing, notably:

- GridExec for submitting GRAM jobs (GRAM client).
- GridFTP and sftp modules for file transfers.
- WS client module
- WSRF (grid service) client module, and

which can be found under `Module library -> Transformer -> Grid`.

The following materials are derived from (Ferner, 2007, 2008).

GRAM Client. GridNexus has a module called `GridExec` provided to submit Globus GRAM jobs. Figure 8.2 shows a simple workflow using `GridExec`. This particular workflow causes the program `/bin/echo` to be executed with the arguments `hello world` from GridNexus. `GridExec` actually constructs an RSL-2/JDD job description file with default settings for output, i.e.

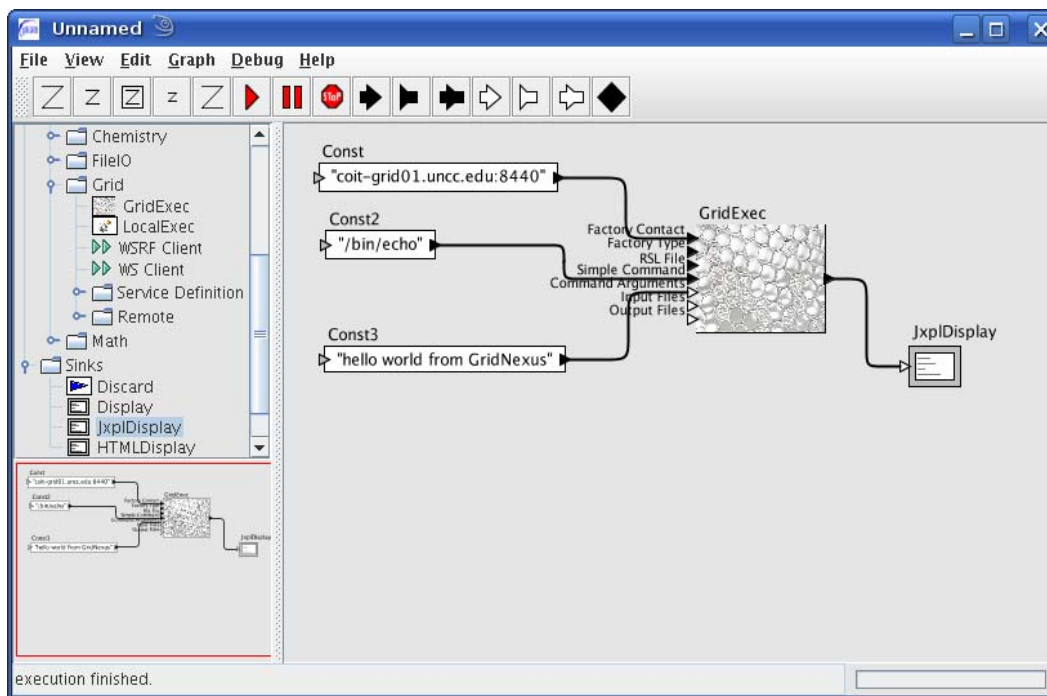


Figure 8.2 Invoking the GridExec module (Ferner 2008). Copied from assign 3 F08 Fig 1

```
<job>
  <executable> //bin/echo</executable>
  <argument>hello</argument>
  <argument>world</argument>
  <argument>from</argument>
  <argument>GridNexus</argument>
  <stdout>${GLOBUS_USER_HOME}/stdout</stdout>
  <stderr>${GLOBUS_USER_HOME}/stderr</stderr>
</job>
```

for the Globus `globusrun-ws` command and essentially causes the command:

```
globusrun-ws -F hostname -submit -f echo.xml
```

to be executed where `echo.xml` is the RSL-2/JDD file for the job. GridExec will also accept the RSL file directly (RSL File input in Figure 8.2). A specific job scheduler can be specified with the `Factory Type` input, for example SGE, Condor, LSF, PBS, etc., specified as a string otherwise the default is the Fork job manager.

The job could be Java program in which case the executable specified is the Java interpreter and the first argument is the Java class file as illustrated in Figure 8.3. In this figure, `/usr/local/java/bin/java` is the Java interpreter and `myIntegral` is the Java program class file. The program arguments are 0 and 5. Notice in this case, the three arguments for the Java interpreter, `myIntegral`, 0 and 5, are input using separate constant boxes that are merged into the `Command argument multi-port` input. Multi-port inputs are identified by white triangles whereas those inputs that do not accept more than one input are identified by black triangles, see Figure

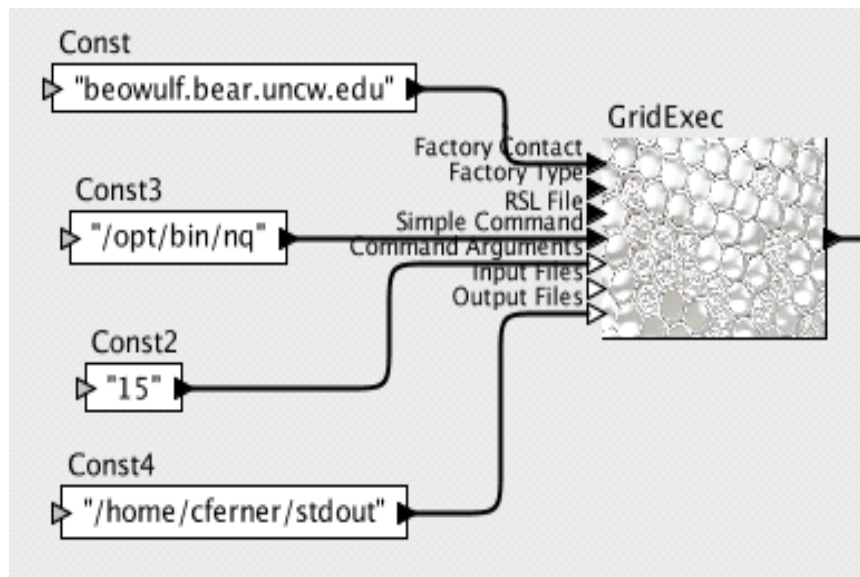


Figure 8.4 Output staging (Ferner 2007). To be changed

8.3. A characteristics of the GridNexus multi-port is that the order that the inputs are drawn on the workflow is important as this determines the order that the inputs enter the multi-port. So for Figure 8.3, the output of Const3 (the Java class file) must be drawn to the Command argument input first, then the other Const boxes which hold the arguments for the Java program.

File Transfers. In the workflows of Figure 8.2 and Figure 8.3, specific input and output files can also be specified using the Input Files and Output Files input. Figure 8.4. shows an example of output staging. In this example, output is streamed to the file `.../stdout` and would correspond to the RSL-2/JDD job description of: (not correct)

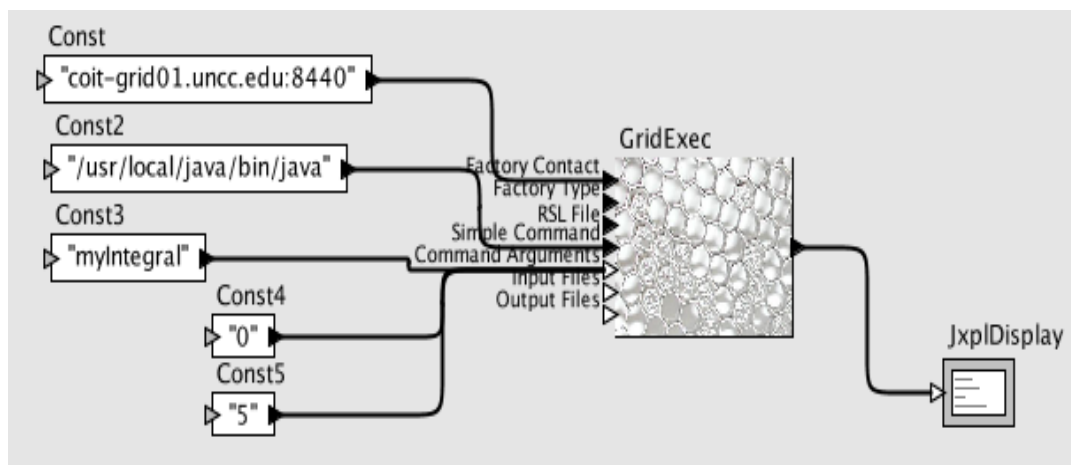


Figure 8.3 Invoking the GridExec module for a Java program. (Ferner 2008) assign 3 S08 Fig 2

```

<job>
  <executable>/bin/echo</executable>
  <argument>15</argument>
  <stderr>${GLOBUS_USER_HOME}/stderr</stderr>
  <fileStageOut>
    <transfer>
      <sourceUrl>file:///home/cferner/stdout</sourceUrl>
      <destinationUrl>gsiftp://gridprof1.bear.uncw.edu/home/
cferner/stdout</destinationUrl>
    </transfer>
  </fileStageOut>
</job>

```

Such file staging requires gridFTP, which is not currently available in Globus 4.0 for a Windows platform. Gridnexus has modules for specific file transfers, both using GridFTP and using SFTP. SFTP is available in Windows. An alternative for file transfers is to use the GridExec module with the `globus-url-copy` command, see (Ferner 2008).

To actually use the GridNexus modules that communicate with Globus installations, the user first has to do some preliminary set-up procedures (assuming a secure container running on the remote machines). The full details can be found at (Ferner 2007, 2008). Briefly, the local `hosts` file needs to contain a mapping hostnames to IP addresses. The certificate authorities need to be recognized. The process for recognizing certificate authorities is similar to as needed for Globus (Chapter 5, Section 5.2.3). The certificate and signing policies files of each certificates authority (`<ca-hashcode>.0` and `<ca-hashcode>.signing_policy`) are downloaded and stored locally in the `.globus/certificates` directory. The user has to obtain a proxy. The process of obtaining a proxy will depend upon where your original certificate is stored. For example a `myProxy` server might be queried for a proxy. The location of the proxy on the local machine needs to given in a file called `cog.properites`². Once these preliminaries are done, one can use GridExec.

WS Client Module. The WS client module acts a client to a Web service with a given WSDL interface document. Once dragged onto the workflow, it is configured by double-clicking it and providing the URL of the Web service WSDL. An example is shown in Figure 8.5. The URL of the Web service WSDL in this case is:

```
http://coit-grid02.uncc.edu:8080/... /axis/MyMath.jws?wsdl
```

In this particular case, the Apache Axis `?WSDL` facility is used that automatically obtains the WSDL file from an already deployed web service. (In fact, using the above URL in a browser should display the WSDL file.) The Web service here is also deployed with the `.jws` instant deployment facility (see Chapter 2, Section).

²CoG Kit (Commodity Grid) is a set of libraries to simplify the development of applications that use Grid computing components. CoG is actually used for the development of some Globus components. See Chapter ?? section ?? for more details on CoG.

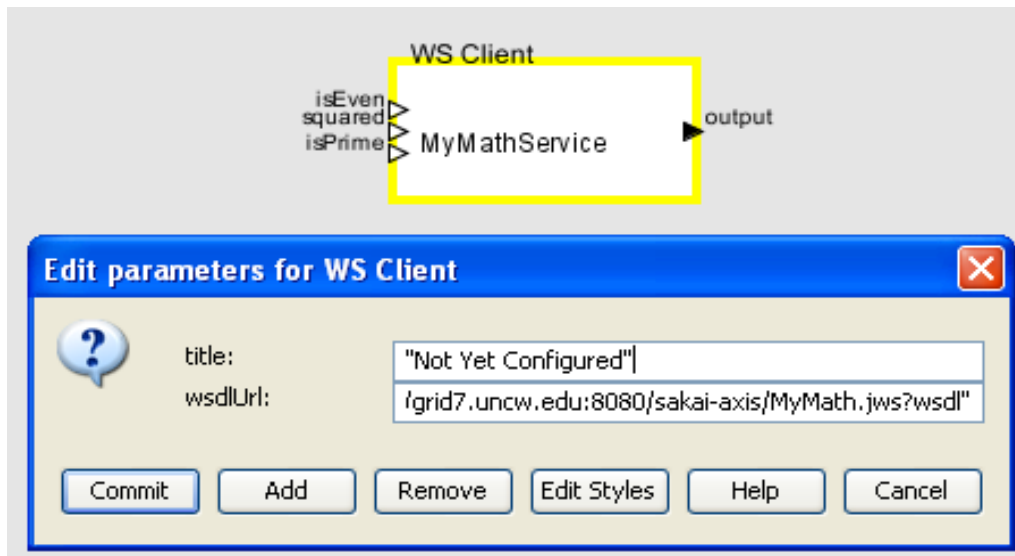


Figure 8.5 Configuring a Web service client module. (Ferner 2007) from assign 5 S07

Once the “Commit” button is clicked, the WSDL document should be read and the service name should be displayed with its named inputs as illustrated in Figure 8.5. Prior to that, no name of inputs would be displayed. This particular service has three inputs (methods), `isEven`, `isPrime` and `squared`. The `isEven` method will return TRUE if the input is even, otherwise FALSE. The `isPrime` method will return TRUE if the input is prime, otherwise FALSE. FALSE will display as an empty list. The square method will return the square of the input. To test this service, one might configure the workflow as shown in Figure 8.6.

WSRF Client. The GridNexus WSRF module acts as a client for a WSRF compliant (Grid) service. This particular module requires two pieces of information to be entered:

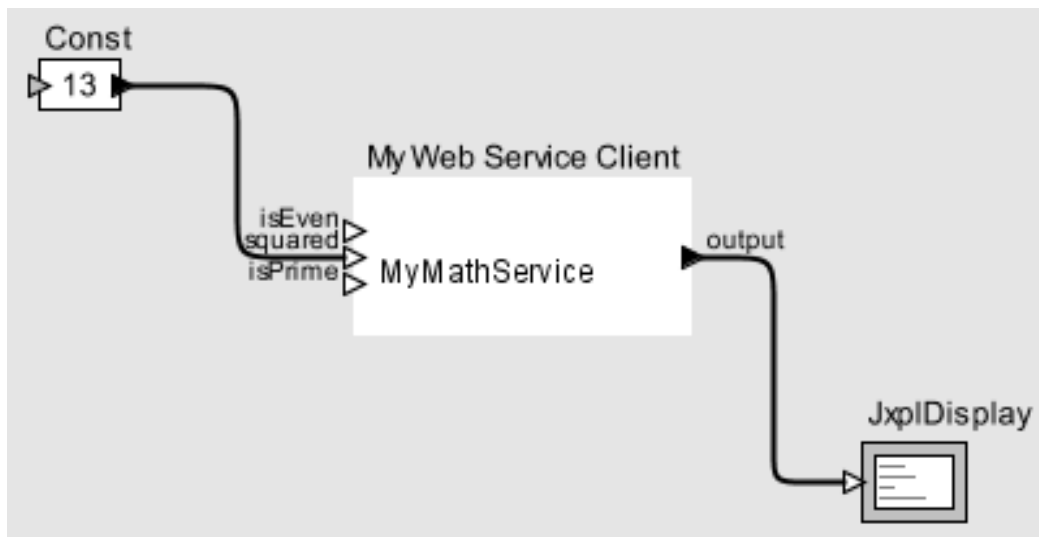


Figure 8.6 Testing a Web service client module. (Ferner 2007) from assign 5 S07

- The URL of the service, or factory URL
- The location of the `AddressingLocator` class of the service.

to be able to configure itself to work with the service (Ferner, 2007).³ The name the `AddressingLocator` class is found from the stub jar package of the service. With these two items entered, the operation names should be displayed as shown in Figure 8.7. Then the service can be used in a workflow such as shown in Figure 8.8.

The `const` module is used to activate the `create` method. The `create` method has no arguments. In GridNexus, as in the underlying Ptolemy, a module (actor) method must be passed something to invoke it. It was decided to use “`false`” in GridNexus. This approach has a problem if a method had an argument that is false. The `SetQ` module assigns a value to a symbol and is used here to set a value to a symbol “`MathEpr`”, which then will be displayed as such from the `JxplDisplay` module. The EPR symbol can then be used subsequently to destroy service instance. (resource associated with ??)

Figure 8.9 shows a workflow consisting of two WSRF Web services. This workflow uses a `Prog` module to execute both Grid service clients.

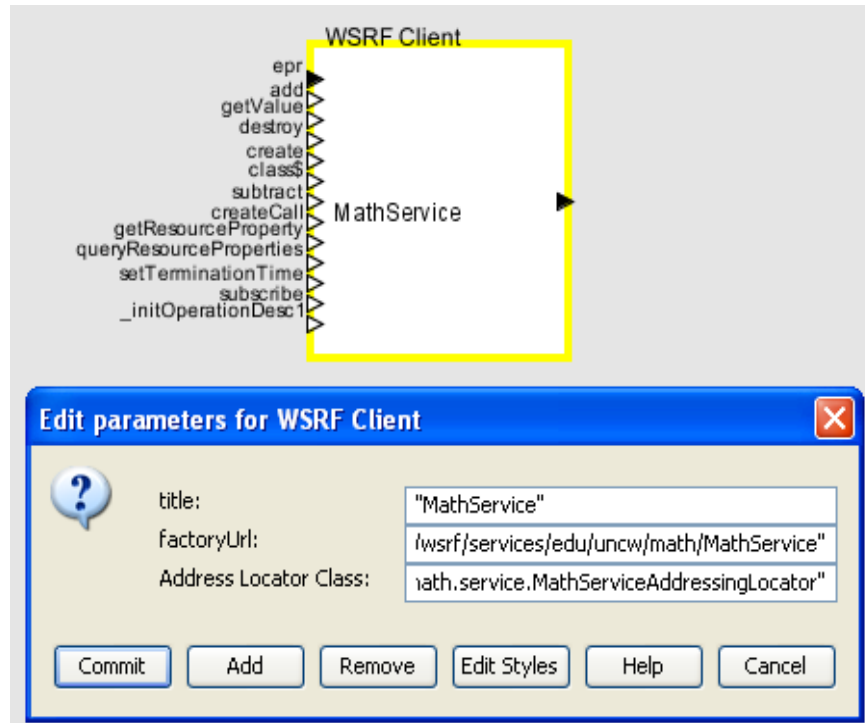


Figure 8.7 Gridnexus WSRF client set-up. (Ferner 2007) from assign 5 S07 - change

³With WSRF Grid services, it is not possible to obtain all the necessary interface information from the WSDL.

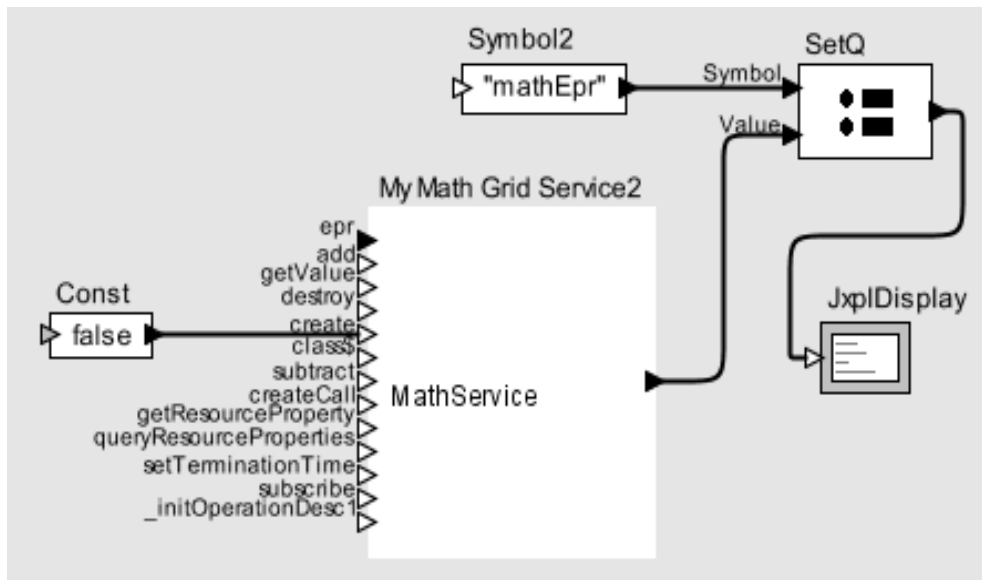


Figure 8.8 Invoking a Grid service client. (Ferner 2007) from assign 5 S07 - change

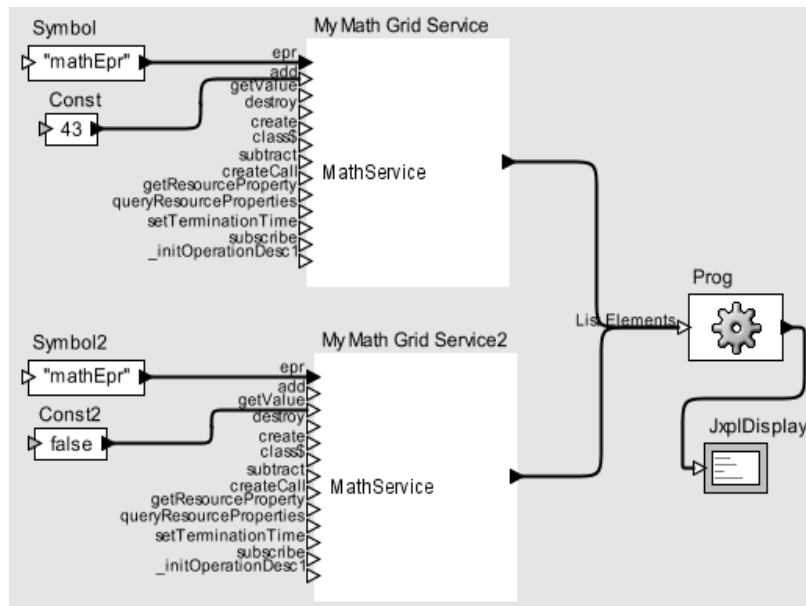


Figure 8.9 GridNexus workflow with two WSRF services. (Ferner 2007) assign 5 S07 - change

8.3 GRID PORTALS

8.3.1 General Features

A Grid portal is a Web page designed to provide a user-friendly interface to a Grid computing environment rather than a command line interface. By its very nature, the portal must support dynamic content, that is, Web pages that can be altered to display

different information during viewing of page or when the page is refreshed. The portal will be hosted as a dynamic Web page on a server and will be accessed by the user through a Web browser from anywhere. The portal should hide the details of the Grid middleware and provides single sign-on and access to Grid computing services, distributed resources and Grid information.

Access to Grid computing services includes the familiar services:

- Security Services
 - Management of certificates
- Remote File Management
 - Access to files and directories
 - Moving files
- Remote job management
 - Job submission
 - Workflow management
- Grid information services
 - Static information (machine type, etc.)
 - Dynamic information (machine load, etc.)

A common generic layout of a portal is shown in Figure 8.10. Users can select from tabs and menus items in much the same way as a one would navigate a commercial Web site. Typically, an active window in the center is used to display information and for the user to create actions such as to run a job. Some early portals would display floating windows after users made their selections. Some portals would have the facilities for graphical output. Feeds from Internet sources are also a possibility. Mostly, the user will fill in HTML text fields and click on buttons. To run a job, for

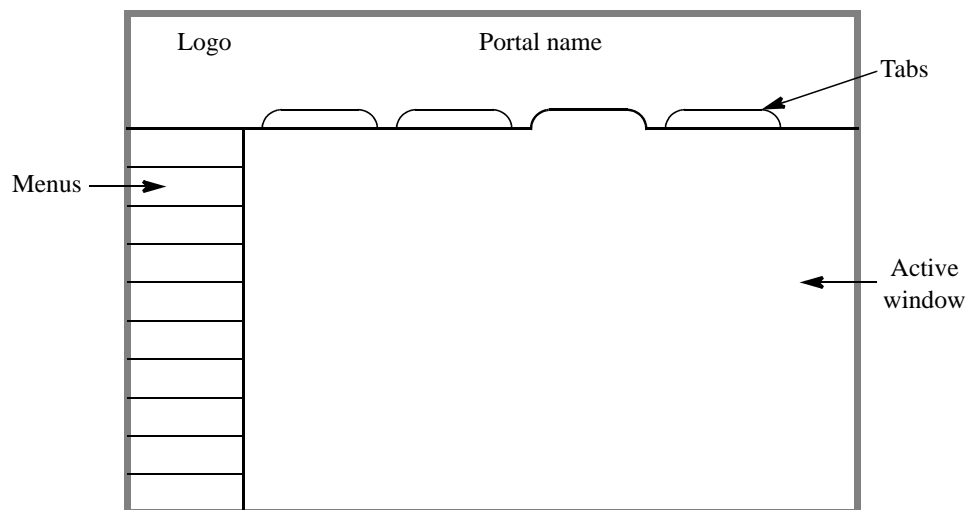


Figure 8.10 Generic layout of a Grid portal.

example, various information has to be entered such as the name of the executable, the arguments, names of input and output files, etc., as was introduced in Chapter 1.

The portal usually incorporates user registration and proxy management components. The most notable proxy management component is the myProxy server. Portals also provide access to information – anything related to tasks at hand, including communication with people in the virtual organization. In fact, some portals started simply as informational portals in the same vein as Web portals such as Yahoo.

A Grid portal comes in one of two forms:

- A general-purpose portal as a front-end to access a Grid computing platform in non-specific application domains, for example SURAGrid portal.
- A portal that is tailored to a particular application domain, for example a bio-informatics portal.

Sometimes, the term *science portal* is used to emphasize its use in science domain. The term *gateway* is also used to describe a portal, for example a *science gateway* for scientists. Scientists in this context could mean many scientific disciplines such as physics, chemistry, biology, etc. The design of a portal that is application non-specific may be different to a portal that is specific to a particular application domain, as the users requirements are different. An application-specific portal should provide access to specific tools of the application domain. A simple non-specific portal is our Grid computing course portal illustrated in Chapter 1. Any self-respecting Grid computing project has a portal. Table 8.1 lists a few of the many scientific portals. There are in fact literally hundreds of portals around the world. Teragrid lists 32 separate portals just associated with TeraGrid (TeraGrid Programs: Science Gateway).

Most Grid portals are created with a *portal toolkit*. Portal toolkits provide a software framework and components to put together a portal easily. The software components can be reused, and potentially components developed by others can be incorporated. Ideally, standard interfaces should exist. Ideally, the *presentation layer* that the user sees should be separated in the software construction from the back-end. Before looking at specific portal toolkits, let us first look at some available technologies for putting together portal toolkits.

8.3.2 Available Technologies

Dynamic Content. *Dynamic content* refers to altering the display of a Web page usually due to user requests, as opposed to *static content* in which the Web page simply displays what is specified in the Web page without the possibility of it changing under varying circumstances. Dynamic content can be done at the client side, that is, after the page is downloaded. The downloaded HTML page already has code embedding that is then executed by the browser and processes user input and alters the web page accordingly. The JavaScript language was designed specifically for such embedded code and all browsers support JavaScript. An excellent introduction to JavaScript is (Knuckles 2001). An alternate way of providing dynamic content is at the server side, that is, when a Web client makes request within a Web page, the server receives this request and simply sends the appropriately altered HTML page,

TABLE 8.1 EXTREMELY SMALL SAMPLE OF SCIENCE PORTALS IN PERIOD 2000 - 2008

Name	URL
Biomedical Information Research Network (BIRN)	https://portal.nbirn.net/gridsphere/gridsphere
Chronos system (Geoscience)	http://portal.chronos.org/gridsphere/gridsphere
GEONgrid (Geoscience)	https://portal.geongrid.org/gridsphere/gridsphere
CIMA X-Ray Crystallography Common Instrument Middleware Architect for Grid enabling" instruments	http://cimaportal.indiana.edu:8080/gridsphere/gridsphere
LEAD (Linked Environments for Atmospheric Discovery)	https://portal.leadproject.org/gridsphere/gridsphere
NEESGrid (Network for Earthquake Engineering Simulation)	https://central.nees.org/login.php
RENCI science portal (bioinformatics)	https://portal.renci.org/portal/
TACC User portal (Texas Advanced Computing Center)	https://portal.tacc.utexas.edu/gridsphere/gridsphere

which is then displayed. Such server-side dynamic content is used for example when the request requires access to a remote database. Both client-side and server-side methods can be combined.

Grid portals can use client-side dynamic content but generally require server-side dynamic content. Server-side dynamic content can use regular programming languages such as C and Java to create the HTML pages that are sent to the client, but also there are technologies specifically for generating server-side dynamic content including CGI (Common Gateway Interface), PHP (originally Personal Home Page, now Hypertext Preprocessor), ASP.NET (Active Server Pages .NET framework), Java Servlets, and JSP (Java Server Pages). CGI is the oldest, which provides a standard protocol between Web servers and client applications. PHP and ASP.NET are non-Java technologies. PHP is a scripting language specifically designed for providing server-side dynamic content. ASP.NET is a Web application framework developed by Microsoft that can provide dynamic content and a successor to ASP (Active Server Pages). However, many Grid portals focus on Java implementations using Java Servlets and JSP. GridSphere, which we mentioned in Chapter 1 as our course portal uses Java servlets and JSP. Hence, let us look at this technology in more detail. Being Java based, it is also platform independent and not tied to one operating system or manufacturer.

Java Servlets. Java servlets are small Java programs (objects) that receive requests from Web clients and generate responses in the environment of a Java Web container, usually handling HTTP requests/responses. The Servlet interface in `javax.servlet` package defines required methods that must be implemented for client-servlet interaction. A servlet container will map URLs to specific servlets. The *servlet container* is a form of Web server that provides an environment for servlets. Another term used is *servlet engine*, which supports servlets. Apache Tomcat is a servlet engine. Servlets can maintain state across server transactions by various means. More information about servlets can be found at (Wikipedia entry: Java Servlets) and its contained links.

Java Server Pages (JSP). Using Java servlets alone would typically require the invoked Java programs to create the HTML using `println` statements. Java Server Pages (JSP) technology is a complementary SUN technology to Java servlets that is used to create Java servlet code from static content. A JSP file is an HTML page with embedded JSP tags. The JSP tags provide for creating the servlet Java code. This Java code is created automatically from the JSP file by a JSP compiler. The final servlet code itself might be fully compiled machine-executable code or Java byte code executed by a Java virtual machine (JVM).

Figure 8.11 shows the interaction of a client with a servlet engine, JSP files, and servlets. The client makes a request for a Web page. Pages with the extension `.jsp` indicate a JSP page. Using a just-in-time approach, the first time the `.jsp` page is accessed, the servlet engine will take the `.jsp` page and create the servlet code. The servlet, through its methods, will be called and create the HTML page which is directed to the Web client for display. Should the page be requested again, the servlet is already created and will respond.

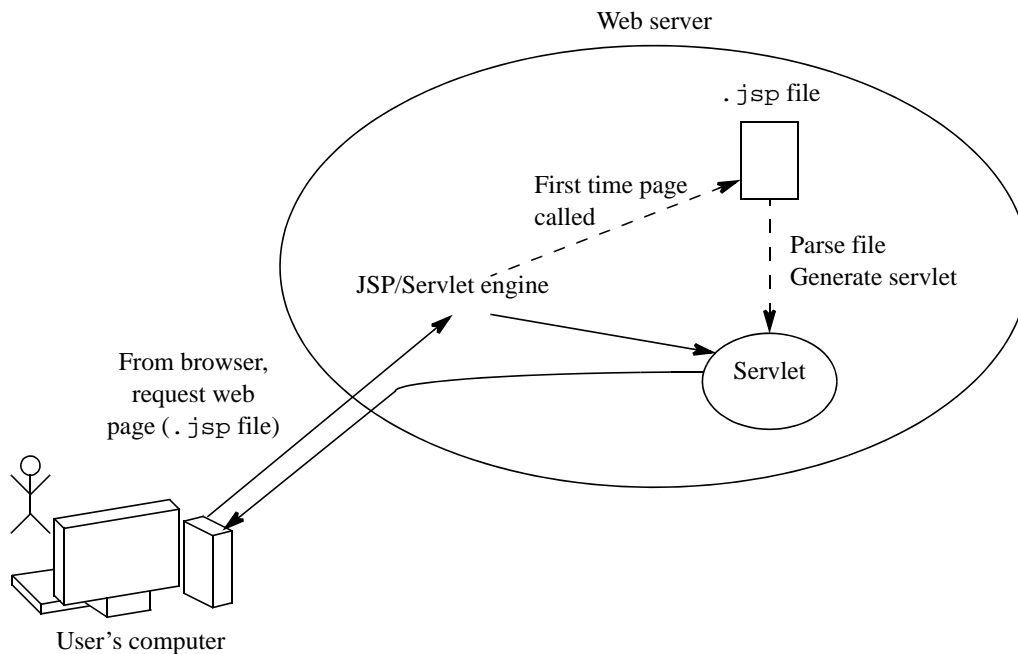


Figure 8.11 JSP/Java servlet environment.

JSP additions to the HTML page are in the form of JSP tags provided for inserting code and related actions. Three tags are available for inserting code. The *declaration* tag (`<%! ... %>`) is used to declare variables and methods, i.e.:

```
<%!  
    Java variable declarations or/and  
    Java methods  
%>
```

The *scriptlet* tag (`<% ... %>`) can be is to include Java code including variable declarations and methods, but also is broader to include any Java code fragment, i.e.:

```
<%!  
    Any Java code fragment  
%>
```

The *expression* tag (`<%= ... %>`) will compute a Java expression and convert the result into a string that is inserted into the HTML code in-line into the HTML code, i.e.

```
HTML code <%= Java expression %> HTML code
```

There are two other tags in JSP. The *directive* tag (`<%@ directive ... %>`) provides information about the JSP page. This directive allows, among other things, a tag library that can extend the functionality of tags to be included and other files. These files could be HTML files or other JSP files and enables reuse and separation of code functions. An example of including code is:

```
<%@ include file = "shared/template.html" %>
```

The *action* tag `<jsp: ... >` is used to invoke server-side JavaBeans⁴, transfer control to another page, and support for applets. The full details of JSP is outside the scope of this textbook and the reader is referred to the SUN home page for JSP (<http://java.sun.com/products/jsp/docs.html>) for more information.

Commodity Grid (CoG) Kits. Commodity Grid (CoG) Kits were conceived during the early development of Globus from 1996 onwards (von Laszewski et al. 2001). It had the objective of combining commodity technologies with Grid components (hence the name “Commodity Grid Kit”) and providing a higher-level interface to Grid components. The so-called commodity technologies referred to here are accepted software components and standard protocols and should not be confused with hardware commodity components. Software components includes common libraries, programming languages such as Java, C, and Python, and standard distributed computing frameworks and Internet protocols. The first and most prevalent

⁴ Java classes that encapsulates a group of objects into a single object (a bean) which can then be treated as such.

Commodity Grid Kit is the Java CoG kit, which provides Java APIs that enable, for example, one to submit and monitor Globus jobs and transfer files by CoG calls within a Java application program, in much the same way as we saw in Chapter 7. This avoids the use of lower level Globus APIs, which change from one version of Globus to another. CoG kit development continued during the same time period as Globus. In fact, Java CoG is used in GT 3.2 and GT 4 for the Java-based GSI, gridFTP, myProxy, and GRAM. A part of the CoG kit, known as *JGlobus* is now included in the Globus 4 distribution. CoG kits encourage the reuse of code avoiding duplication of effort, and hopefully isolates the application programming from changes in the underlying Grid middleware, which changed rapidly during the early 2000's and continues to date.

Figure 8.12 shows a sample CoG kit program to transfer files. Java CoG also provides facilities for graphical interfaces targeted at portal design, and became a

```
filetransfer.java
import org.globus.cog.abstraction.impl.common.AbstractionFactory;
import org.globus.cog.abstraction.impl.common.task.ServiceContactImpl;
import org.globus.cog.abstraction.interfaces.FileResource;
import org.globus.cog.abstraction.interfaces.SecurityContext;
import org.globus.cog.abstraction.interfaces.ServiceContact;
import org.ietf.jgss.GSSCredential;

public class filetransfer {
    public static void main( String args[]){
        filetransfer trans = new filetransfer();
        try{
            trans.transferFile();
        }catch( Exception e){

        }
    }
    public void transferFile() throws Exception{
        FileResource client = null;
        client = AbstractionFactory.newFileResource("gridftp");
        SecurityContext securityContext=
        AbstractionFactory.newSecurityContext("gridftp");
        securityContext.setCredentials(null);
        client.setSecurityContext(securityContext);

        ServiceContact serviceContact=
        new ServiceContactImpl("coit-grid03.uncc.edu:2811");

        client.setServiceContact(serviceContact);

        //Start the client
        client.start();
        client.getFile
        ("/home/username/stdout.txt", "home/username/stdout.local.txt");
        client.stop();
    }
}
```

Figure 8.12 CoG kit program to transfer files (Villalobos 2007). [Check source](#)

central component in early portals and OGCE portals (Alameda et al. 2007) see later. It also developed workflow facilities (von Laszewski and Hategan 2005).

8.3.3 Early Grid Portals

Several portal toolkits have been developed since the mid 1990's. The early ones began before standards were developed for portal design, which we will describe later, and used existing technologies such as Java servlets, JSP, and CoG Kit. These portals are called first generation Grid portals by Li and Baker (2005), who describe first generation portals as mainly using a three-tier architecture. The client browser is in tier one, which communicates with a Web server and proxy credential server in tier two, which communicates with an application manager in tier three, which manages grid resources also in tier three. Usually, there needs to be a persistent storage or a database to hold information, both static information about the Grid environment and dynamic information. In Li and Baker's description, the application manager may use an events archive to store the state of the application, which can be monitored by the Web browser. Mostly these portals had very little ability, if any, to be customized. Some examples of general-purpose portal toolkits in this period include:

- Grid Portal Development Kit (GSDK) – used JSP for the presentation layer and JavaBeans and Java CoG back-end. GSDK is not now supported.
- NPACI Grid Portal Toolkit (Gridport) (National Partnership for Advanced Computational Infrastructure) – used HTML for the presentation layer and Perl/CGI
- Ninf/Gridspeed portals – used JSP/Java Servlet for the presentation layer and Java CoG back-end

All of the above interfaced to a MyProxy server and the Globus toolkit. Such general purpose portal toolkits can be used to create a portal, for either a broad non-specific application Grid computing platform or for narrow application domains such as for high energy physics. Examples using early grid portal toolkits include the National Partnership for Advanced Computational Infrastructure (NPACI) Hotpage Grid portal based upon GridPort. (Thomas and Boisseau 2003) The NPACI Hotpage Grid portal provided access to NPACI resources.

8.3.4 Development of Grid Portals with Portlets

Early Grid portals “tools” were not very flexible. They were often tied to specific programming tools and Grid software, such as Globus 2.4. The specific programming structure was not suitable for users to develop portals themselves. The API's were not standardized. In contrast, the portal implementation should be flexible, meet Grid industry standards, and be able to be extended using parts developed by others. Now that Grid computing has embraced Web services as a core technology, the portal framework should interact with these services. A software engineering approach called the Model-View-Controller (MVC) originally developed in the late 1970's is attractive, in which the architecture is divided into a model component that is responsible for manipulating the data and domain-specific information, the view component

that is responsible for the user's visual interface and a controller component that is responsible for handling the input/output and responding to requests and events.

Portlets. A general approach for portal design developed in the early-mid 2000's is to use "software components" called *portlets*. A presentational layer of the portal is constructed with portlets. Each portlet provides a specific functionality and a window within the portal as illustrated in Figure 8.13. Each portlet might be associated with a particular Grid service depending upon the functionality. The portal can use as many portlets as is needed to create the required functionality. Portlets provide for all the functionality that would be expected to access Grid resources, including:

- Proxy certificate management
- Job submission and run-time management
- Remote file transfers
- Access to information services (resource status, etc.)
- Collaborative tools (email, chat, discussion boards, etc.)

and also depending upon the application, specialized portlets for interfacing to domain specific applications.

The portlets themselves can be compared to servlets and requires a similar environment called a *portlet container* managed by a portlet server. In general, portlets do not communicate with each other, only with the services they front-end, and only provide for the presentation-level. With the portlet approach, it should be easy to reconfigure user's view. Different portlets from different sources should be able to be plugged into portal. Central to its broad adoption and to enable third party plug-in's is the development of a portlet standard. Several vendors developed portlet

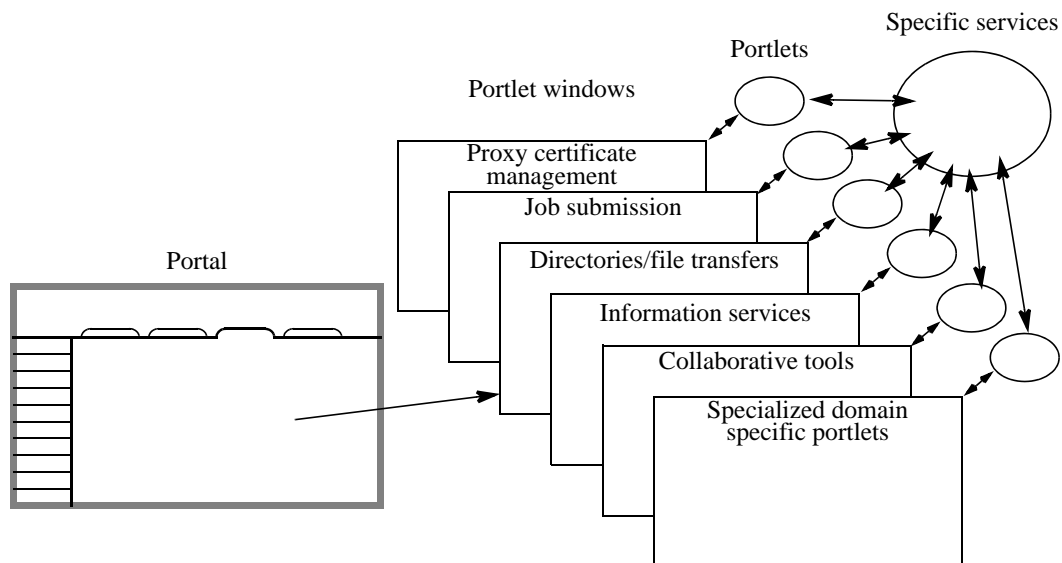


Figure 8.13 Portlets.

API's including IBM's Websphere portlet API's. The open-source Apache Jetspeed project embodied portlets.

JSR 168. After the early experiences of portal designs in the mid-late 1990's, an effort was made to develop a Java portlet specification in the 2000-2002 period leading to the Java Specification Request JSR 168 Portlet Specification released in Oct. 2003. JSR 168 is also called Java Portlet Specification version 1.0 and is based upon the Apache Jetspeed portlets.

Portlet code generally has the following structure:

1. Initialize
2. Render portlet
3. For a request received:
 - Accept request and perform required back-end actions
 - Render display according to result
4. Finalize and destroy portlet

The JSP 168 method for Step 1 is `init()`. If the portlet is to be rendered, the appropriate JSR 168 method is `doView()`. Handling a request is illustrated in Figure 8.14. The two JSR 168 methods for handling a request are `processAction()` and `render()`. `processAction()` takes two objects `ActionRequest` and `ActionResponse` for input and output respectively. Apart from the portlet code, XML deployment descriptors are needed. We shall describe a specific JSR 168 portlet later. More information about the JSR 168 API's can be found at (Abdelnur and Hepper 2003) and (Sun 2003).

Other work was obviously going on during this period on Grid tools. The National Science Foundation started the Middleware Initiative (NMI) in 2001 initially over 3 years to create and deploy network services focusing on Grid and related software. It provided a centralized location for important Grid software including Globus, Condor, MPI-G2, and subsequently a new collaborative project

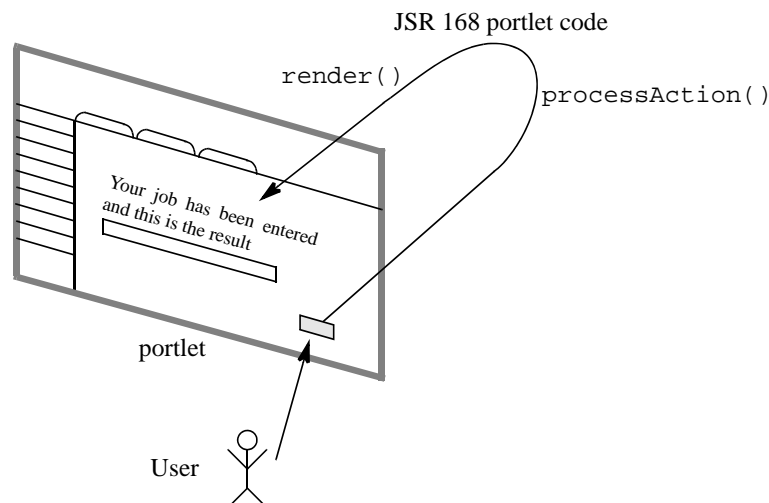


Figure 8.14 JSR 168 Portlet activities.

called the Open Grid Computing Environment Consortium (OGCE), which was established in 2003 to create collaborations and shareable components for portals. The OGCE portal release 2 consisted of a core set of JSR 168 compatible Grid portlets. The portal was independent of the specific container. Two portal containers were supported, uPortal and GridSphere. Originally GridSphere and OGCE2 together would be described as a OGCE2/GridSphere portal, but subsequently it was simply referred to as GridSphere.

GridSphere. GridSphere uses JSR 168 portlets and became extremely well adopted. Many subsequent production Grid portals are based upon GridSphere as evidenced from the URL's of the portals listed Table 8.1. Our Grid course portal is based upon GridSphere. The GridSphere portal can be identified for sure by the portal URL when the default directory is used (. . . /gridsphere/gridsphere). The portal may use the default HTTP port 80 or an alternative non-privileged HTTP port 8080. Production Grid portal also often use the secure HTTPS, which would use the default port 443, or an alternative specified non-privileged port such as 8440. The port chosen has not to conflict with the Globus container if running on the same server, which typically runs on port 8443 by default. It is necessary to ensure that user certificates are signed by a certificate authority that is recognizable to the portal. Typically, portals are not fully accessible except by registered users with certificates signed appropriately.

The GridSphere core portlets include:

- Login
- Locale, profile and layout personalization
- Administration portlets for creation of users, groups, portlet management
- Localization support French, English, Spanish, German, Dutch, Czech, Polish, Hungarian, Italian, Arabic, Japanese, and Chinese.

onto which many other portlets can be installed from various sources, for example the myProxy server portlet, Globus job submission and control portlets, information services portlets, collaborative tools such as Sakai, etc.

GridSphere can be installed easily on a personal computer (Windows, mac, or or Linux) and it is convenient to do so to develop one's own portlets, see for example, an assignment in our course (Villalobos, Land, and Wilkinson 2007). Portlets in GridSphere are deployed into a servlet engine, for example Tomcat as shown in Figure 8.15. First one would install the Tomcat servlet engine. Then one would install GridSphere. The default location for the installed GridSphere is `http://localhost:8080/gridsphere/gridsphere` or simply `http://localhost:8080/gridsphere`. At this location is a set-up screen to create the portal administrator account, Once configured, one gets the login screen as shown in Figure 8.16. Many production portals use this login page. Note the neat language selection at the top right side of the login page, which will change the language used in the text and alter the layout to match language customs if necessary. Once logged in, a row of tabs will display, one for each portlet installed, in the manner of Figure 8.10.

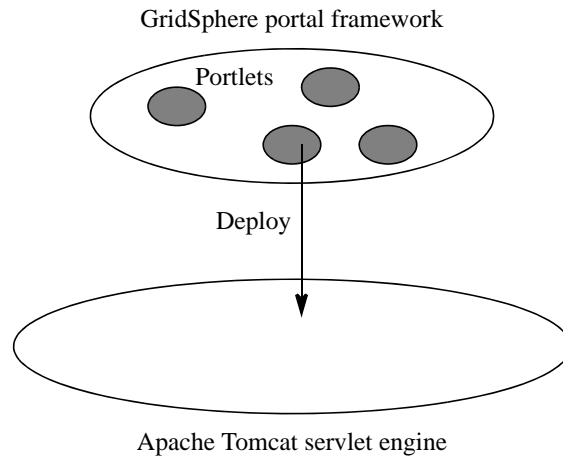


Figure 8.15 GridSphere portlets deployed into a servlet engine (Tomcat).

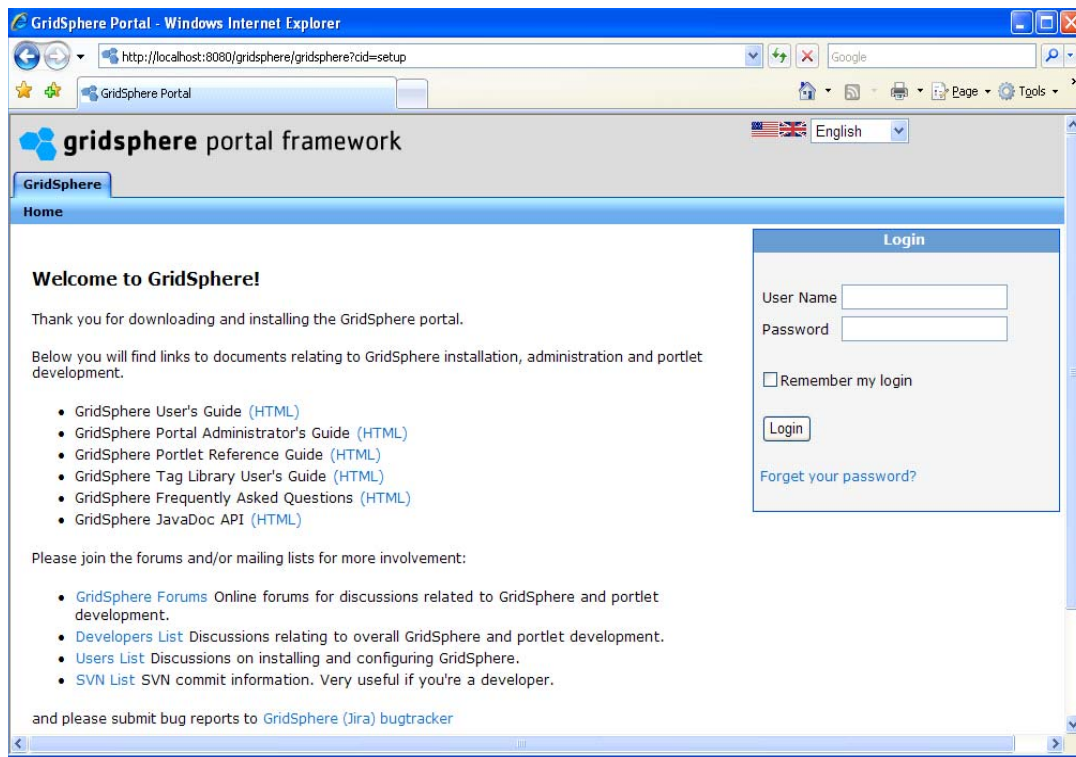


Figure 8.16 Default GridSphere login page.

Creating and Deploying a JSR 168 portlet in GridSphere. It is extremely easy to create one’s own portlet using sample code, as described in the assignment described in (Villalobos, Land, and Wilkinson 2007). The following is directly from this assignment. Here, we will briefly go over the steps. Suppose one wanted to create a simple portlet as a front-end to a Java program. To make it concrete, suppose the Java program would just tell whether a number was even or odd. Although trivial, this will illustrate the steps required, which would be applicable to larger realistic Java programs. Suppose the portlet should look as shown in Figure 8.17. This layout

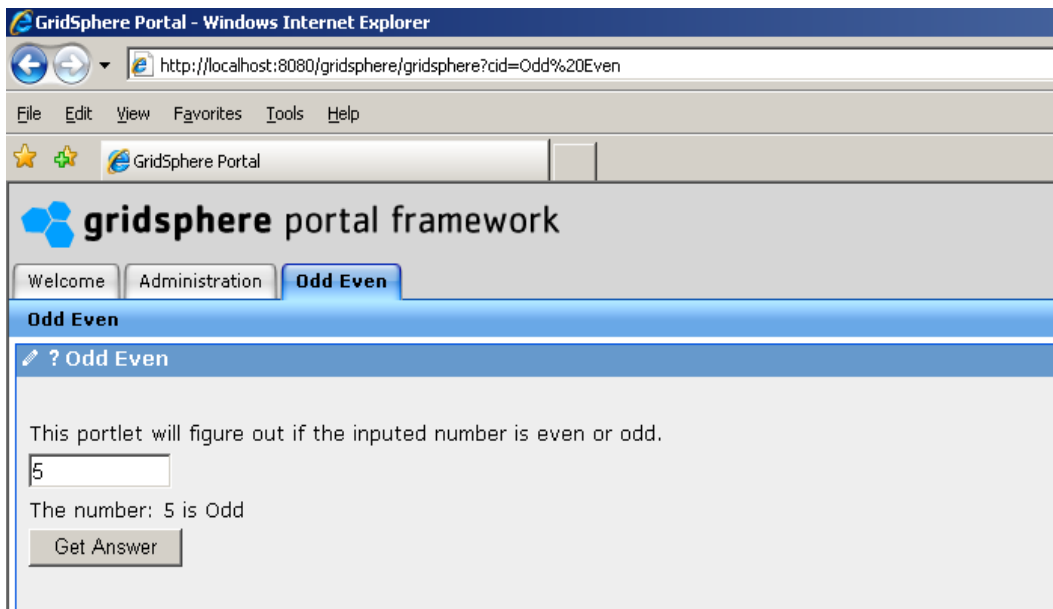


Figure 8.17 Final portlet displayed in GridSphere.

can be described by an HTML table consisting of a single column and four rows as shown in Figure 8.18, The first column holds text (which may wrap around depending upon the desired width of the table). The second row holds where a user will enter a number. The bottom row holds an HTML button where the user indicates input is read to be read, and the third row holds the result text output indicating whether the number is even or odd.

GridSphere provides an ant⁵ script to create the directory structure and sample files needed for a portlet. The portlet designer then simply needs to provide the Java source file that does the required evaluation and modify the supplied portlet deployment descriptor files accordingly. Suppose the project directory is called `oddeven-portlet`. All files relating to the portlet are placed in folders within this directory.

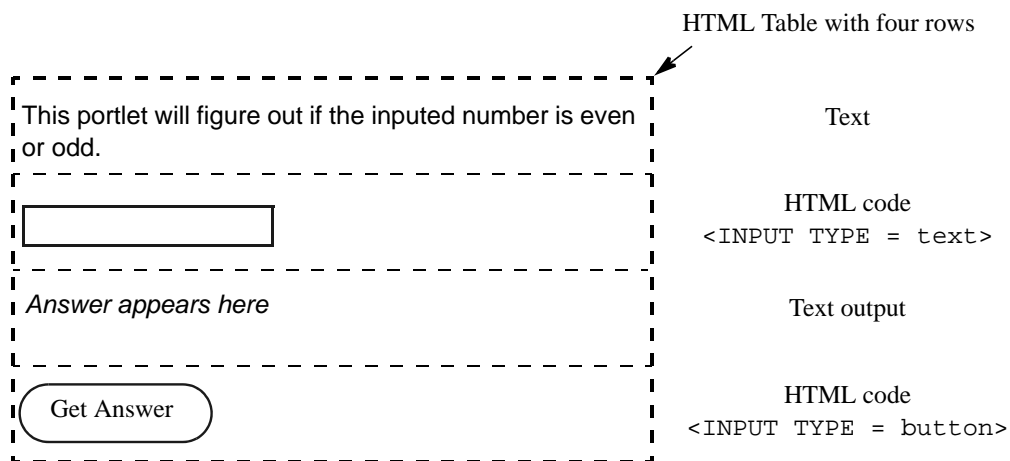


Figure 8.18 Portlet layout.

⁵ A java-based build tool from the Apache project similar to make but using XML scripts.

The files needed are:

- Java source file for portlet
- HTML/JSP file describing the layout
- Three portlet deployment descriptor files

HTML/JSP File. The layout is defined by a JSP file called `MainPage.jsp` as shown in Figure 8.19. This file is mostly simple HTML code modified with JSP tags added at the beginning. `<portlet:defineObjects/>` creates `renderRequest`, `renderResponse` and `portletConfig` objects. `<%@ taglib uri="LibraryURI" prefix="tagPrefix" %>` declares that custom tags are used defined in a tag library given by the URI, and provides prefixes names. The table width is defined in the JSP code (Figure 8.19) has 500 pixels and hence the first line of text wraps around in the final display (Figure 8.17).

Source File. The Java source file causes the actions required when the user interacts with the portlet. The Java source code is shown in Figure 8.20. This program calls the method `init()` to initialize the JSR 168 portlet, and provides two methods for handling user requests, `action()` and `prepare()`. `action()` perform the required computational actions upon a request and creates data for the HTML text box. `prepare()` renders the portal. `isEven()` is a helper method for `action()`.

```

<%@ taglib uri="/portletUI" prefix="ui" %>
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet" %>
<portlet:defineObjects/>
<ui:form>
  <ui:table width="500">
    <ui:tablecell>
      This portlet will figure out if the inputed number is even or
      odd.
    </ui:tablecell>
    <ui:tablecell>
      <ui:textfield size="10" beanId="valueTF1"/>
    </ui:tablecell>
    <ui:tablecell>
      <ui:text beanId="answer"/>
    </ui:tablecell>
    <ui:tablecell>
      <ui:actions submit action="action" value="Get Answer"/>
    </ui:tablecell>
  </ui:table>
</ui:form>

```

Figure 8.19 JSP code Java source code, `MainPage.jsp`. [Check source](#)

```

package edu.uncc.abw.portlets;
import org.gridlab.gridisphere.provider.*;
import javax.servlet.UnavailableException;
import javax.portlet.*;
import java.io.IOException;
import java.io.*;

public class OddEven extends ActionPortlet {
    private static final String DISPLAY_PAGE = "MainPage.jsp";
    public void init(PortletConfig config) throws PortletException {
        super.init(config);
        DEFAULT_VIEW_PAGE = "prepare";
    }

    public void action(ActionFormEvent event) throws PortletException {
        TextFieldBean value1 = event.getTextFieldBean("valueTF1");
        TextBean answer = event.getTextBean("answer");
        int val = Integer.parseInt( value1.getValue() );

        if(value1.getValue() == null ){
            answer.setValue("");
        }
        else {
            if( isEven(val) ){
                answer.setValue("The number: "+value1.getValue()+" is Even");
            }else{
                answer.setValue("The number: "+value1.getValue()+" is Odd");
            }
        }
        setNextState(event.getActionRequest(), DISPLAY_PAGE);
    }

    public void prepare(RenderFormEvent event) throws PortletException {
        setNextState(event.getRenderRequest(), DISPLAY_PAGE);
    }
    public boolean isEven(int val){
        return val % 2 == 0;
    }
}

```

Figure 8.20 Java source code for portlet, OddEven. java. [Check source](#)

The HTML/JSP file is identified with DISPLAY_PAGE and used in rendering routine.

Deployment Descriptor Files. There are three deployment descriptor files:

- portlet.xml – JSR 168 standard, describing the portlet
- layout.xml – GridSphere file describing layout of portlet within page
- group.xml – GridSphere file describing a collection of portlets

These files are very simple XML files that do not require namespaces and can be easily deciphered, see next. There are other deployment files that are generated automatically during deployment but the above are the only ones that the user would alter.

The `portlet.xml` file is shown in Figure 8.21 and identifies the Java class file for the Java portlet program, `edu.uncc.<username>.portlet.OddEven.java.class`. (This class file will be found in the build directory once the Java source program is compiled and the portlet deployed.)

The layout file, `layout.xml` is shown in Figure 8.22 and specifies the portlet “tab” layout and also specifies the path to the class file. The layout is described in terms of a table with a specified number of columns and rows in the same manner as HTML pages. This particular portlet has one row and one column, but multiple rows and columns could be specified. Note that the HTML layout required for the Java program is defined elsewhere in the JSP page associated with the Java program

The group file, `group.xml`, is shown in Figure 8.23 and specifies the group membership name and role for the portlet. Portlets are assigned a particular group for managing portlets. After deploying the portlets in the portal, the group will be used to select those portlets required. (check)

Building and Configuring. Once all files are in place, the portlet is compiled and deployed with a GridSphere provided Ant script. Then Tomcat must be restarted and the portlet group membership configured using the GridSphere Portlet group manager portlet.

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-
app_1_0.xsd"
  version="1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-
app_1_0.xsd">
  <portlet>
    <description xml:lang="en">Odd Even Portlet</description>
    <portlet-name>OddEven</portlet-name>
    <display-name xml:lang="en">Odd Even Portlet</display-name>
    <portlet-class>edu.uncc.abw.portlets.OddEven</portlet-class>
    <expiration-cache>60</expiration-cache>
    <supports>
      <mime-type>text/html</mime-type>
      <portlet-mode>edit</portlet-mode>
      <portlet-mode>help</portlet-mode>
    </supports>
    <supported-locale>en</supported-locale>
    <portlet-info>
      <title>Odd Even</title>
      <short-title>Odd Even</short-title>
      <keywords>odd even</keywords>
    </portlet-info>
  </portlet>
</portlet-app>
```



Figure 8.21 portlet.xml. [Check source](#)

```

<portlet-tabbed-pane>
  <portlet-tab label="Odd Even">
    <title lang="en">Odd Even</title>
    <portlet-tabbed-pane style="sub-menu">
      <portlet-tab label="oddeventab">
        <title lang="en">Odd Even</title>
        <table-layout>
          <row-layout>
            <column-layout>
              <portlet-frame label="Odd Even">
                <portlet-class>
                  edu.uncc.abw.portlets.OddEven
                </portlet-class>
              </portlet-frame>
            </column-layout>
          </row-layout>
        </table-layout>
      </portlet-tab>
    </portlet-tabbed-pane>
  </portlet-tab>
</portlet-tabbed-pane>

```

Specified columns and rows of a table in portlet. Components are in cells of table.

Path to portlet class file Path to portlet code (There is an alternative format using # symbol)

Figure 8.22 layout.xml. [Check source](#)

```

<?xml version="1.0" encoding="UTF-8"?>
<portlet-group>
  <group-name>userPortlets</group-name>
  <group-description>The demo group</group-description>
  <group-visibility>PUBLIC</group-visibility>
  <portlet-role-info>
    <portlet-class>edu.uncc.abw.portlets.OddEven</portlet-class>
    <required-role>USER</required-role>
  </portlet-role-info>
</portlet-group>

```

Group name

Path to portlet class file

Figure 8.23 group.xml. [Check source](#)

JSR 286 and WSRP. JSR 286 Portlets Specification v2.0 is an updated version of JSR 168 released in June 2008 after about five years of review and development. JSR 286 is (thankfully) backward compatible with JSR 168 portlets (SUN 2008). JSR 168 portlets can be deployed in JSR 286 portlet containers. JSR 286 incorporates inter-portlet communication which was absent in JSR 168. OASIS introduced a standard for defining a Web service interface for interacting with “presentation-oriented Web services” called Web Services for Remote Portlets (WSRP) version 1 in 2003. It uses WSDL for its interface description. WSRP Version 2 was introduced in 2008 (Web Services for Remote Portlets Specification v2.0 2008). JSR 286 includes an alignment with WSRP version 2.

8.4 SUMMARY

Within the focus of providing user-friendly interfaces, this chapter has two major topics: workflow editors with a graphical user interface, and Web-based Grid portals. In workflow editors, it introduced the GridNexus workflow editor in some detail. With the Grid portal section, several relevant technologies were introduced for creating server-side dynamic content in Web pages:

- Java servlets
- Java server pages (JSP)
- Commodity Grid Kits
- JSR 168 Portlets

and then went on to describe the GridSphere portlet toolkit in some detail including sample files that need to be created for a portlet.

FURTHER READING

Much of the further reading material for this chapter can be found on-line through the homes pages of the various technologies that have cited as references. A early description of CoG kit can be found in (von Laszewski, et al. 2001). The CoG kit is also described a chapter of the book *Making the Global Infrastructure a Reality* (von Laszewski 2003). Historical information on first generation portal toolkits can be found in the same book (Novotny 2003), (Thomas and Boisseau 2003).

BIBLIOGRAPHY

- Abdelnur, A, and S. Hepper. 2003 Java™ Portlet Specification Version 1.0, Sun Microsystems, Inc. Oct. 7.
- Alameda, J., M. Christie, G. Fox, J. Futrelle, D. Gannon, M. Hategan, G. von Laszewski, M. A. Nacar, M. Pierce, E. Roberts, C. Severance, and M. Thomas. 2007. The Open Grid Computing Environments Collaboration: Portlets and Services for Science Gateways *Concurrency and Computation Practice and Experience* 19 no.6:921–942.
- Brooks, C., E. A. Lee, X. Liu, S. Neuendorffer, Y. Zhao, and H. Zheng. 2008. Heterogeneous Concurrent Modeling and Design in Java (Volume 1: Introduction to Ptolemy II). EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2008-28, April 1. <http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-28.pdf>
- Brown, J. L., C. S. Ferner, T. C. Hudson, A. E. Stapleton, R. J. Vetter., T. Carland, A. Martin, J. Martin, A. Rawls, W. J. Shipman, and M. Wood. 2005. GridNexus: A Grid Services Scientific Workflow System. *Int. Journal of Computer Information Science (IJCIS)* 6, no 2 (June 20): 72–82.
- Ferner, C. 2007. Assignment 5 Using GridNexus to Create Workflows that use Web and Grid Services, ITCS 4146/5146: Grid Computing (Spring 2007). <http://www.cs.un->

- cc.edu/~abw/ITCS4146S07/assign5S07.pdf.
- Ferner, C. 2008. Assignment 3 GridNexus Job Submission, ITC 4146/5146: Grid Computing (Fall 2008). <http://www.cs.uncc.edu/~abw/ITCS4146F08/assign3F08.pdf>.
- Gridsphere portal framework. <http://www.gridsphere.org/gridsphere/gridsphere>
- GridNexus UNC-Wilmington Grid Computing Project. The University of North Carolina Wilmington. Content last updated on July 24, 2007. <http://www.gridnexus.org/>
- Grid Workflow Forum. <http://www.gridworkflow.org/snips/gridworkflow/space/start>
- Kepler Project. <http://kepler-project.org/>
- Knuckles, C. D. 2001. *Introduction to Interactive Programming on the Internet using HTML and JavaScript*. New York: John Wiley & Sons.
- Li, M. and M. Baker 2005. *The Grid Core Technologies*. Chichester England: Wiley.
- Novotny, J. 2003. The Grid Protal Development Kit. in Grid Computing. In *Making the Global Infrastructure a Reality*, ed. F. Berman, G. C. Fox and A. J. G. Hey, 657–673. Chichester England: Wiley.
- Open Grid Computing Environments Portal and Gateway Toolkit. http://www.collab-ogce.org/ogce/index.php/Main_Page
- Sun. 2003. Introduction to JSR 168 — The Java Portlet Specification. <http://developer.sun.com>
- Sun. 2008. Introducing Java Portlet Specifications: JSR 168 and JSR 286. Sun Developer Network. <http://developers.sun.com/portalserver/reference/techart/jsr168/>
- Taverna project website <http://taverna.sourceforge.net>
- TeraGrid Programs: Science Gateways. http://www.teragrid.org/programs/sci_gateways/#communities
- Thomas, M. P., and J. R. Boisseau 2003. Building Grid Computing Portals: The NPACI Grid Portal Toolkit. In *Making the Global Infrastructure a Reality*, ed. F. Berman, G. C. Fox and A. J. G. Hey, 657–673. Chichester England: Wiley.
- uPortal Evolving Portal Implementation from Participating Universities & Partners. <http://uportal.org/>
- Villalobos, J. 2007. Creating Java Cog Programs that Connect Grid services. <http://www.cs.uncc.edu/~abw/ITCS4146S07/???.doc>.
- Villalobos, J., Land, J., and B. Wilkinson. 2007. Portlet Development: Creating a Simple Portlet. ITCS 4146 Grid Computing (Spring 2007) Portlet design assignment. <http://www.cs.uncc.edu/~abw/ITCS4146S07/assign6aS07.doc>.
- von Laszewski, G., I. Foster, J. Gawor, and P. Lane. 2001. A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience* 13 no 8–9:643–662. <http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--cog-cpe-final.pdf>.
- von Laszewski, G., and M. Hategan 2005. Java CoG Kit Workflow Concepts. *Journal of Grid Computing* 3, nos 3–4:239–258. <http://www.mcs.anl.gov/~gregor/papers/vonLaszewski-workflow-jgc.pdf>.
- von Laszewski, G., J. Gawor, S. Krishnan, and K. Jackson 2003. Commodity Grid Kits Middleware for Building Grid Computing Environments in Grid Computing. In *Making the Global Infrastructure a Reality*, ed. F. Berman, G. C. Fox and A. J. G. Hey, 639–656. Chichester England: Wiley. <http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--grid2002book.pdf>
- Web Services for Remote Portlets Specification v2.0. 2008. OASIS Standard. <http://docs.oasis-open.org/wsrp/v2/wsrp-2.0-spec.html>.
- Wikipedia entry: ASP.NET. <http://en.wikipedia.org/wiki/ASP.NET>.

Wikipedia entry: Common Gateway Interface. http://en.wikipedia.org/wiki/Common_Gateway_Interface.

Wikipedia entry: Java Servlets. http://en.wikipedia.org/wiki/Java_Servlet.

Wikipedia entry: PHP. <http://en.wikipedia.org/wiki/PHP>.

Yu, J., and R. Buyya. 2005. A Taxonomy of Workflow Management Systems for Grid Computing, *Journal of Grid Computing* 3, nos 3–4:171–200.

SELF-ASSESSMENT QUESTIONS

The following questions are multiple choice, Unless otherwise noted, there is only one correct question for each question.

1. What is GridNexus?
 - a) A software package to run Grid jobs
 - b) A Grid portal
 - c) A software package to create workflows
 - d) A term used to describe an amorphous Grid

2. What is JXPL?
 - a) A job description language
 - b) An XML language for JSR 168 portlets
 - c) The XML language used in GridNexus to describe workflow
 - d) A Java XML programming language

3. What is the purpose of the GridNexus *JxplDisplay* module?
 - a) To display the workflow
 - b) To compute the JXPL document for the workflow and optionally evaluate it
 - c) To display JXPL a man page
 - d) To display all the input for the workflow

4. What is the purpose of the GridNexus *WS Client* module?
 - a) Provides a module for a local or remote Web Service
 - b) Provides a module to access a local or remote Web Service
 - c) Provides a module to access a Weather Service
 - d) Provides a module to access a client for a Weather Service

5. How is the GridNexus *WS Client* configured?
 - a) It self configures
 - b) Provide a title or name
 - c) Provide the URL of the Web service WSDL document
 - d) Provide the URL of the Web service server
 - e)

6. How is the GridNexus *WS Client* configured?
 - a) It self configures

- b) Provide a title or name
 - c) Provide the URL of the Web service WSDL document
 - d) Provide the URL of the Web service server
7. What is the difference between the GridNexus *WS client* and the GridNexus *WSRF client*?
- a) Nothing significant
 - b) WS client interfaces to a Web Service and WSRF client interfaces to a Grid service
 - c) WSRF client is a backward-compatible improvement to the WS client
 - d) WS client is a backward-compatible improvement to the WSRF client
8. What is the purpose of the GridNexus *GridExec* module?
- a) Execute a Grid service
 - b) Submit a job to GRAM
 - c) Create a Grid execution service
 - d) Performs an ssh connection to a remote computer
9. What is the GridNexus *composite* module?
- a) A workflow with every module connecting to every other module
 - b) A module composed of every type of module that exist in GridNexus
 - c) A workflow considered as one unit with a set of inputs and a set of outputs
 - d) A module that two modules with the output of the first module connecting to the input of the second module.
10. Which module in GridNexus was inherited from the underlying Ptolemy software?
- a) cond
 - b) composite
 - c) GridExec
 - d) GridFTP
11. What is a Grid portal?
- a) Software for moving information between Grids
 - b) A database of Grid resources
 - c) The interface between Grid services
 - d) A Web-based application for a user to access Grid services and resources
12. What is meant by *dynamic content* in the context of web pages?
- a) A Web page that keeps changing a regular intervals
 - b) A Web page that changes in response to changing information and client requests
 - c) A Web page that is constructed using a just-in-time approach, when requested
 - d) A Web page that has to be refreshed at regular intervals
13. What is a Java servlet?
- a) Java program that handle request from Web clients
 - b) A Web service written in Java
 - c) A small Java-based server

- d) A Java program that creates small objects
- 14.** What is a Java Server Page (JSP)?
- a) Java code that implements a Web service
 - b) A SUN technology used in conjunction with servlets to create web pages dynamic content.
 - c) Java code that can implement a server
 - d) An XML language used to describe Java server programs
- 15.** What is Commodity Grid (COG) kit?
- a) A toolkit to create a Grid platform for Grids on commodity cluster hardware
 - b) A toolkit providing higher level interfaces to Grid components
 - c) A toolkit specifically to trading commodities on the Grid
 - d) Software beneath Globus to access the Grid resources
- 16.** What is a portlet?
- a) Software component used for a window/area of a portal
 - b) A pull-down menu in a Grid portal
 - c) A group of portals
 - d) A small Grid portal
- 17.** What is one difference between JSR 168 and JSR 286
- a) Nothing
 - b) 120,000 new APIs
 - c) JSR 286 includes facilities for inter-portlet communication
 - d) They are completely different standards with nothing in common

PROGRAMMING ASSIGNMENTS

Assignment 8-1 require a software environment for deploying portals and portlets. It can be done on a personal computer (Windows, mac, or Linux). Instructions on installing the software is provided in the referenced links. Assignment 8-2 requires an account on servers with Globus installed. (Could provide guest accounts??)

8-1 Follow the instructions given for the assignment at <http://www.cs.uncc.edu/~abw/ITCS4146S07/assign6aS07.doc> to install GridSphere and deploy the simple oddeven portlet. Develop and deploy another portlet that allows the user to perform add, subtract, multiply and divide operations on two numbers entered by the user.

8-2 Follow the instructions given for the assignment at <http://www.cs.uncc.edu/~abw/ITCS4146S07/assign???.doc> to install the CoG kit libraries, test the installation and compile and run the sample code to submit jobs and transfer files.