IST-2001-32133
GridLab - A Grid Application Toolkit and Testbed

GridLab Portal Design

Author(s):	Jason Novotny, Michael Russell, Oliver Wehrens
Document Filename:	GridLab-4-1.1-0006-GridLabDesign
Work package:	Work Package 4: Portals
Partner(s):	MPG
Lead Partner:	MPG
Config ID:	GridLab-4-1.1-0006-1.0
Document classification:	PUBLIC

Abstract: This document specifies the architecture and implementation design of the GridLab portal. This document is intended to be used in the development of the GridLab portal and adheres to the requirements specified in the WP4 requirements analysis document.

Contents

1 Introduction

Grid enabled portals have become increasingly popular as a platform to provide application scientists access to Grid services and resources. The goal of the GridLab project is to enable Grid application development by supporting higher level libraries and services. The GridLab Portal will provide scientific researchers with a single point of entry from which they can gain easy access to the broad array of Grid services developed by the other work packages in the GridLab project. Users will be provided with a friendly and easy to use interface by allowing them to interact with these services through standard means such as a web-browser or PDA. The hourglass model depicting the role of the GridLab portal as a gateway between end-users and GridLab service providers is shown below:

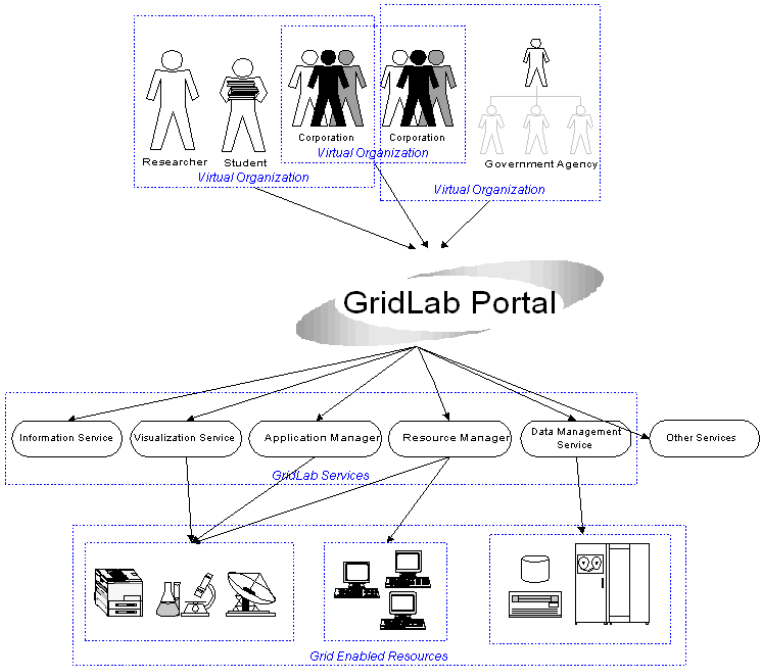

Figure 1: Shows how the portal acts as an intermediary between users of Grid Resources and the resources themselves.

The design of the GridLab portal will allow for maximum modularity and flexibility to support the many different needs and requirements of both the GridLab portal users, typically computational scientists and the GridLab service providers. The scientists require an easy to use, efficient and value-laden set of interfaces for interacting with the Grid to make their work more productive. The service providers need to provide a client API or protocol specification as well as information on how the service is to be used in practice. This includes hints on developing a useful visual portal interface to the service or information on how the service is intended to be used in conjunction with additional services made accessible from the portal.
The GridLab portal will be implemented using the GridSphere portal framework. The GridSphere framework provides a general portal architecture that supports virtual organizations comprised of scientists and project developers as well as an API for the development of reusable, modular components that serve to access the services being developed within the GridLab project. These components, called Portlets, and the overall portal architecture are described in more detail in the following sections.

2 GridLab Portal Consumers

The design and delivered functionality of GridSphere will be based on the requirements of the portal users. The following four group categorizations serve to highlight the range of functionality and differing requirements of the supported portal community.

2.1 End Users

End users are generally application scientists wishing to use the portal as a useful gateway for performing some set of tasks that require access to Grid services and resources. In essence, the ultimate usefulness of the portal will depend on its ability to simplify to some extent the work done by computational scientists by providing an easy to use, feature-laden work environment. Supporting end users amounts to capturing the application requirements of specific research groups and providing the same capabilities from the portal. In practice, the use of a web browser or PDA is much more limiting than the workstation running specialized applications that a scientist is used to. This trade-off is acceptable as the value of the portal is realized more in integrating the various applications and services required by the scientists accessible via a common platform as opposed to providing an equivalent level of functionality as the set of tools on a scientists workstation.

2.2 Portal Administrators

Portal administrators are responsible for the deployment and management of the portal and are exposed to the technical issues of application server configuration and software installation. A requirement of portal administrators is that the portal be easy to install and deploy with a minimum of site specific configuration changes. Ideally, the portal will expose much of its administration tasks via the portal to allow administrators the ability to reconfigure the portal from a web browser or PDA. Portal administrators are also responsible for providing roles and permissions to users and general user management.

2.3 Portal Developers

Portal developers are responsible for the design and implementation of GridSphere as outlined in this document and satisfying requirements of the overall portal community. Portal developers also include providers of new portal functionality or portal enhancement. A key design consideration in the GridSphere architecture is to provide a framework for the easy development of new portal components, while requiring a minimum of knowledge about the underlying portal implementation.

2.4 Service Providers

GridLab service providers are represented by other GridLab work packages responsible for the development of additional Grid services such as the resource broker (WP 9), information services (WP 10), data management and visualization (WP 8), and monitoring (WP 11) . For any service a provider wishes to expose to the portal, a protocol specification or client API must be provided. The portal will provide generic support for OGSA[27] based Grid services making it easy for a portal developer or administrator to readily make the OGSA service portal accessible. Service providers that do not plan on interacting with end-users of the GridLab project can still benefit from providing portal access. The portal could be used for remote service administration and management by a service provider if so desired.

3 Portal Architecture Overview

The GridLab portal architecture is based on the standard 3-tier architecture adopted by most web applications as shown in Fig. (§2). Tiers represent physical and administrative boundaries between the end user and the resources accessed via the application server.

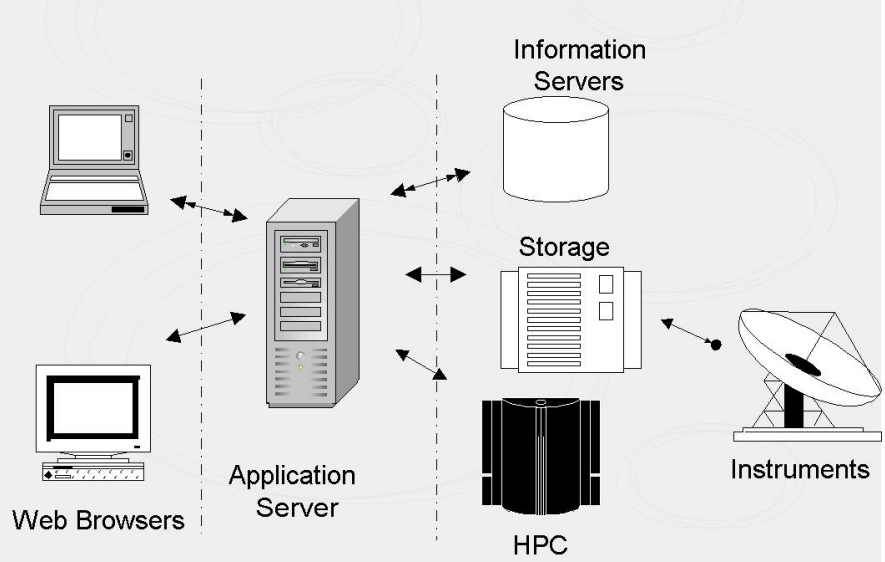

Figure 2: Standard 3-tier web architecture

Tier 1 represents the client and may include any device a user wishes to use to access the portal. Typically, tier 1 is represented as a web browser platform, but may also include mobile devices including mobile phones and PDA's. In fact, the Mobile Access work package (WP 12) of the GridLab project is concerned with providing mobile support to end users. We plan on collaborating closely with WP 12 to ensure the GridLab portal is also accessible to mobile users. In the case of web browsers, the Portals WP requirements document specifies the use of a secure (SSL-capable) web browser that can support DHTML/CSS/Javascript for improved interactivity, and cookies to allow session data to be transferred between the browser and the application server. The second tier is the application server and is responsible for handling HTTP/S requests generated from a client and is necessarily multi-threaded to allow multiple concurrent client connections. The GridLab portal augments the application server with Grid enabling software and support for multi user access to Grid services. All other resources accessed by GridSphere including databases, credential repositories and high-performance computers fall under the third tier, known as the back-end. Back-end resources are generally under separate administrative control from the web application server and subject to different policies and use conditions. GridSphere is specially tailored to provide access to Grid resources as the back-end resources. It is generally assumed that Grid resources understand a subset of defined Grid and Internet protocols and/or provide Grid services[27].

3.1 GridLab Portal Architecture

4 GridSphere Framework Overview

As learned from the previous portal development projects, the time and effort involved in building a robust portal environment from scratch is prohibitively high. Sadly enough, many research groups around the world have been building ”stove-pipe” portal solutions from the ground up with very little emphasis on reusability and extensibility. Application frameworks are desiged to address this limitation by providing a ”semi-complete” domain-specific application that can be specialized to produce custom applications[**?**]. Frameworks, however, are not a silver bullet; they must be well designed, tested and provide useful, comprehensive documentation in order to gain widespread success and adoption. Design patterns[1] are also an important aspect of component based frameworks and offer solutions to commonly recurring software development problems. Patterns and frameworks integrated together both facilitate reuse by capturing successful software development strategies. The GridSphere Portal framework makes use of many commonly understood design patterns and can therefore be understood and improved upon by other portal framework developers. The goal of the GridSphere framework is to make the development of new portal interfaces and offer new functionality as easy as possible. It is envisioned that users of GridSphere can simply download and install the code, and easily develop new functionality after reading the User's guide which offers a recipe for developers to follow. The GridSphere framework is an example of a *white-box* framework. Whitebox frameworks rely heavily on OO language features like inheritance and dynamic binding to achieve extensibilty. Existing functionality is reused and extended by (1) inheriting from framework base classes and (2) overriding pre-defined hook methods.[**?**]. The difficulty with white-box frameworks is that developers are required to have some knowledge of base framework classes. In the case of GridSphere, however, the core framework classes that framework users must have knowledge of are not internal GridSphere specific components, but a community supported API, called the Portlet API.

4.1 GridSphere Portlet Framework

Portlets [22] have become an increasingly popular concept used to describe visual user interfaces to a content or service provider. From a technical perspective, portlets represent modular, reusable software components that may be developed independently of the general portal architecture and offers a specific set of operations. For instance, portlets may provide users with an updated list of stock quotes or content from a news feed. Within the GridLab portal framework, portlets offer atomic functionality such as a job submission component or a remote file browser interface. In a web browser or PDA, portlets can be aggregated together supplying the user with easy and efficient access to multiple sources of content, services and applications. A portal user may be able to add (subscribe) or remove portlets from their personalized portal page depending on their needs. The Portlet API is in the process of being reviewed within the Java Community Process (JCP) and a Java Specification Request (JSR)[23] is currently in the process of being standardized.
Currently, the Jakarta Jetspeed project[11] provides an implementation of the evolving portlet API and a complete portal framework based upon another Jakarta project, Turbine[12]. Believing that the Jetspeed/Turbine framework would suit most of the GridLab portal requirements, an evaluation[26] was made. Unfortunately, we determined that Jetspeed did not provide a suitable development platform in its current state and the dependencies upon existing large over-extended software was too high. In the evaluation, a comparison was made to a commercial portlet implementation offered by IBM's WebSphere Portal Server. While we lacked the time

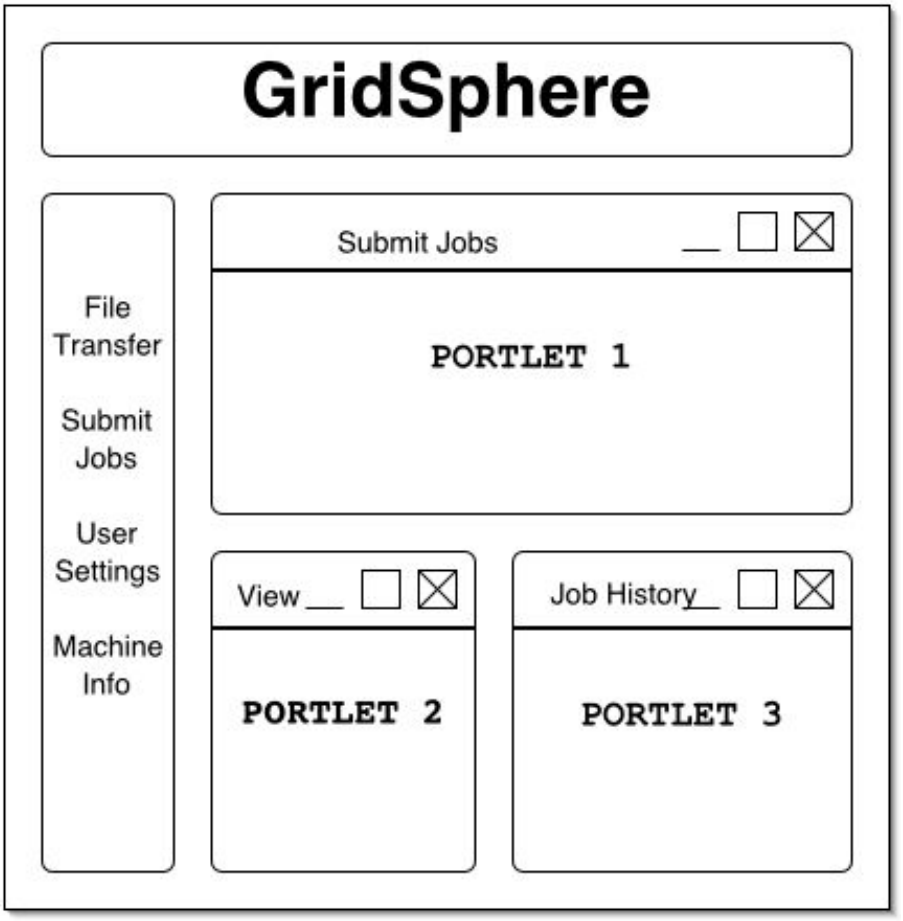

Figure 3: Portal Page Containing Portlets

and money to evaluate WebSphere software, the WebSphere Portlet Development Guide[24] offers a very solid description of the portlet API that comes bundled with WebSphere and appears far more functional than the constanty evolving portlet API specified in Jetspeed. In fact, the WebSphere Portlet API was actually based upon Jetspeed in version 2.1 and inherited from many of the core classes. WebSphere Portal version 4+ appears to be a refactoring of the portlet API to reduce the dependance on Jetspeed and offer a more coherent model for describing portlets. Because the Portlet API implementation offered in WebSphere is not open-source, it would be impossible to modify the internal libraries to suit the needs of the GridLab project. In addition, the software is prohibitively expensive thereby denying other academic research groups the ability to use our software components.
Ultimately, while we wish to use the final standardized portlet API, however it will not appear for at least another 6 to 12 months. Therefore, the approach that will be taken is to design a customized implementation of the portlet API that uses the IBM WebSphere Portlet API where useful as this is the most likely path for the final standardized portlet API in the JCP. The GridSphere Portlet API is intended to be lightweight and offer only the base functionality required to implement the portlet concept in practice. Hereafter, Portlet API will be taken to refer implicitly to the GridSphere Portlet API as the actual JSR is still waiting to be finalized and approved. The following sections detail the design of the GridSphere Portlet API and its relation to the existing Servlet model. Diagrams, in the Unified Modeling Language (UML), are

provided where useful and it is expected that as the design is further refined a more thorough object class specification will be provided in UML. Currently, the primary goal of the design is to provide a conceptual overview of the necessary software components without going into too much implementation details.

4.2 Portlet Design

The implementation of the Portlet class is based on the IBM WebSphere Portlet API version 4.1. The WebSphere Portlet API provides the following portlet class hierarchy shown in (§4): The IBM WebSphere Portlet is used by the portlet container to invoke the portlet. As part of running within the portlet container each portlet has a life-cycle similar to a servlet lifecycle. The corresponding methods are called in the following sequence:

- The portlet is constructed, then initialized with the init() method.
- Any calls from the portlet container to the service() method are handled.
- The portlet is taken out of service, then destroyed with the destroy() method, then garbage collected and finalized.

An *application portlet* is a newly deployed portlet that maintains configuration information for all portlet instances in a PortletConfig object that is defined initially by the portlet deployment descriptor.
A *concrete portlet* instance is a deployed application portlet that is parameterized by a PortletSettings object that is defined in the portlet deployment descriptor. There can be multiple concrete portlet instances per application portlet. As an example, multiple concrete monitoring portlets may specify different monitoring services to contact for information.
A *user portlet* instance is defined as an instantiated concrete portlet instance containing persistent user data plus the user's session. The virtual instance is created and destroyed with the login() and logout() methods, respectively. If a portlet provides personalized views these methods should be implemented.
The Base Portlet class directly extends from HttpServlet implying that like servlets, portlets must also be registered in the standard web.xml deployment descriptor of a servlet container. This will described in more detail later. The Portlet class provides a definition of the Portlet.Mode class and abstract methods for the portlet lifecycle methods and the abstract login and logout methods implemented from the PortletSessionListener interface.
The PortletAdapter extends from the Portlet class and provides base implementations for the portlet lifecycle methods as well as skeleton implementations of the login and logout methods provided by the PortletSessionListener. The login method allows portlet developers to instantiate resources on a users behalf when a user is logged into the portal. Additionally, the PortletAdapter provides methods to support the Portlet *modes*, a description of the Portlets presentation. The supported Portlet modes described:

- View - this is the normal mode of operation for a Portlet
- Edit - allows the Portlet to be customized by the user
- Configure - allows the Portlet to be configured by a portal administrator
- Help - displays help information to provide additional information on a Portlet

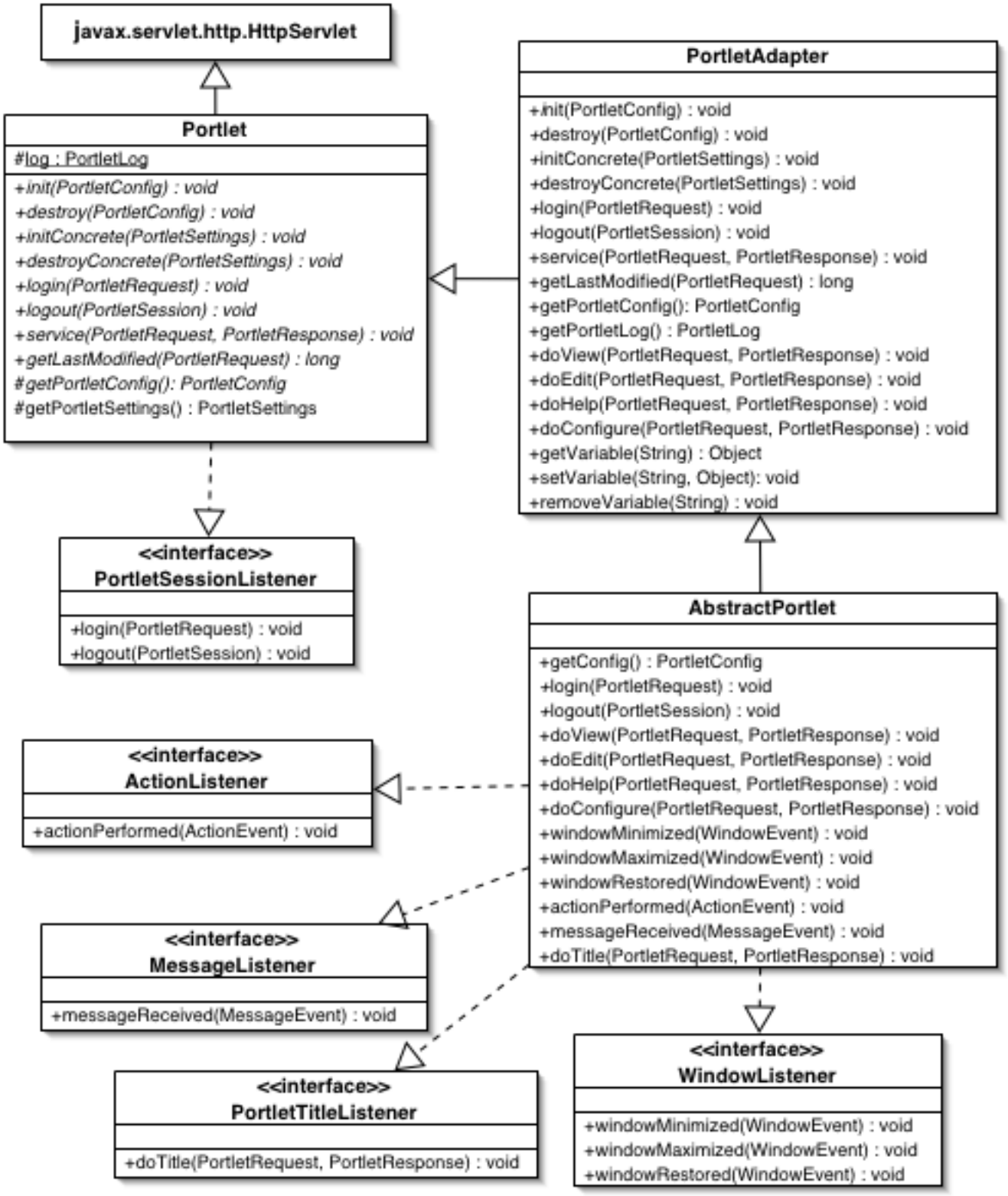

Figure 4: Base Portlet API

The service method of the PortletAdapter is also responsible for invoking the proper doView, doConfigure, or doEdit method based on the portlet mode dictated by the set of icons on the title bar.
The AbstractPortlet provides an abstract base implementation of the PortletAdapter class and also implements the four types of action listeners described in the Portlet events section. The AbstractPortlet is responsible for making sure the appropriate action is invoked when a ActionEvent, WindowEvent or MessageEvent occurs.

4.3 Portlet Events

Portlet events come in the following four flavors:

- Actions events - General portlet events such as form submission, etc.
- Window events - Triggered when the window state of the portlet changes
- Message events - Triggered when a portlet sends a message to another portlet
- Title events - Called by the portlet container to render the portlet title.

Events are handled by the AbstractPortlet class which implements the ActionListener, WindowListener, MessageListener and PortletTitleListener interfaces. Portlets that wish to handle events must provide implementations of the Listener methods. An ActionEvent object containing the PortletRequest and PortletResponse objects gets passed into the actionPerformed method of the portlet. This allows portlet developers to perform all back-end logic operations in the actionPerformed method. Similary, WindowEvents are passed into windowMinimized, Maximized, Resized methods in portlets that choose to override the default WindowListener methods.

4.4 Portlet Deployment Descriptor

The portlet deployment descriptor defines a portlets capabilities and initialization parameters in XML format. A portlet application definition is represented by the application portlet description and the one or more concrete portlet descriptions. Multiple application definitions comprise a portlet collection.
The application portlet description contains the following configuration attributes:

- Portlet name - the name of the portlet
- Servlet name - the name of the corresponding servlet defined in web.xml
- Portlet config - specifies portlet initialization parameters see PortletConfig
- Supported modes - specifies which modes are supported by this portlet
- Allowed window states - specifies which window states are supported by this portlet

The concrete portlet description contains the following attributes:

- Context params - specifies concrete portlet application settings parameters in the PortletApplicationSettings object
- Default locale - the default locale to use when displaying this portlet
- Language - provides title, description and keywords in the chosen language

- Access information - specifies which roles within specified groups may access the portal
- Config params - specifies concrete portlet setting parameters in the PortletSettings object

The following descriptor describes an example Hello World portlet:

```
<portlet-app-def>

<portlet-app id="org.gridlab.gridsphere.portlets.core.HelloWorld.666">
    <portlet-name>Hello World Portlet Application</portlet-name>
    <servlet-name>HelloWorld</servlet-name>
    <portlet-config>
        <param-name>Portlet Master</param-name>
        <param-value>yourid@yourdomain.com</param-value>
    </portlet-config>
    <cache>
        <expires>120</expires>
        <shared>true</shared>
    </cache>
    <allows>
        <maximized/>
        <minimized/>
        <resizing/>
    </allows>
    <supports>
        <markup name="html">
            <view/>
            <edit/>
            <help/>
            <configure/>
        </markup>
        <markup name="wml">
            <view/>
            <edit/>
            <help/>
            <configure/>
        </markup>
    </supports>
</portlet-app>

<concrete-portlet-app id="org.gridlab.gridsphere.portlets.core.HelloWorld.666.2">
    <context-param>
        <param-name>foobar</param-name>
        <param-value>a value</param-value>
    </context-param>
    <concrete-portlet>
        <portlet-name>Hello World</portlet-name>
        <default-locale>en</default-locale>
        <language locale="en\_US">
            <title>Hello World - Sample Portlet \#1</title>
```

```
            <title-short>Hello World</title-short>
            <description>Hello World - Sample Portlet \#1</description>
            <keywords>portlet hello world</keywords>
        </language>
        <config-param>
            <param-name>Portlet Master</param-name>
            <param-value>yourid@yourdomain.com</param-value>
        </config-param>
        <allowed-access visibility="public">
            <role>ADMIN</role>
        </allowed-access>
    </concrete-portlet>
</concrete-portlet-app>

</portlet-app-def>
```

4.5 Core Portlet Interfaces

Just as portlets themselves inherit functionality from the Servlet API, the Portlet API also provides wrappers around standard Servlet interfaces to provide a more portlet-centric view of the portal. For instance, PortletRequest inherits from HttpServletRequest and provides additional functionality and an API for dealing with portlet sessions, modes, users and clients (a client defines the client platform used for accessing the portal). Similarly, PortletResponse and PortletSession subclass HttpServletResponse and HttpSession to provide portlet specific functionality for generating responses and handling sessions. PortletConfig and PortletContext (§6) inherit from ServletConfig and ServletContext respectively and provide methods for portlet initialization, obtaining portlet container information and obtaining localized text and logging. The portlet container is responsible for the generation of these objects and passing them on to the Portlet classes via login, logout, service, etc.
The PortletSettings object provides dynamic portlet customization information including the PortletApplicationSettings. Upon startup, the PortletSettings contains the concrete portlet information from the portlet deployment descriptor and can be modified by authorized users, thus dynamically modifying a portlets configuration information.

4.6 Portlet Container

The portlet container is responsible for the execution and lifecycle management of portlets. The controller servlet, GridSphereServlet, uses the PortletManagerRegistry to handle the administration of portlets including installation, removal, initialization and shutdown. The relevant container classes are shown in Fig. (§7).
When the PortletManagerService is instantiated in the GridSphere servlets init() method, it loads in the portlets specified in the core portlets and manager portlet web applications which form the foundation portlets provided by the GridSphere framework. Because portlets are packaged as separate web applications they are organized as data structures of Portlet web applications containing multiple application portlets, each one of which may contain multiple concrete portlets. These data structures Fig. (§8) are maintained by the Portlet Registry, a singleton that simply maintains a hash of application portlets.
The portlet container dispatches events to portlets by combination of the PortletInvoker and the PortletDispatcher. The PortletDispatcher is created initially when the ApplicationPortlet

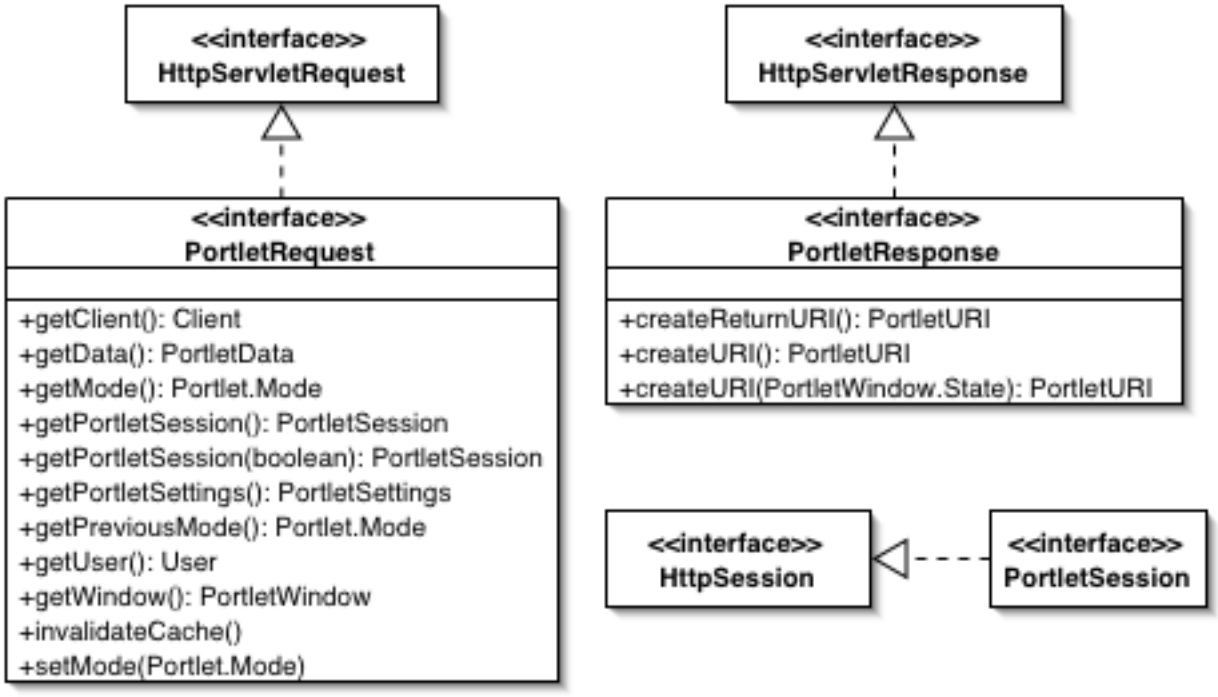

Figure 5: Portlet Extensions to Servlet API

is constructed and contains the required servlet dispatcher to invoke the desired portlet. The PortletInvoker is simply a wrapper around the PortletDispatcher that allows clients to perform lifecycle methods on a particular portlet by passing in the concrete portlet ID. The methods of the PortletInvoker and PortletDispatcher are very similar to the Portlet classes, with the exception that a request and response must be passed into every method since it requires a servlet request dispatcher to invoke the actual portlet (servlet).

4.7 Portlet Packaging and Deployment

One of the primary goals of the GridSphere framework is to provide support for third-party portlet development and make it as easy as possible to develop new portlets without requiring access or knowledge of the portlet container. To support this, portlets are deployed using the same mechanisms as Java Servlets. A portlet is packaged into a Web Application Repository (WAR) file containing portlets, JSP pages, utility classes, static documents, client side Java classes (applets) and descriptive meta information that ties the preceding elements together. The hierarchical structure of a WAR is described in the Servlet 2.3 specification[9] and listed below:

- /jsp
- /WEB-INF/portlet.xml
- /WEB-INF/web.xml
- /WEB-INF/layout.xml
- /WEB-INF/classes/java files e.g. Portlets, beans, etc.

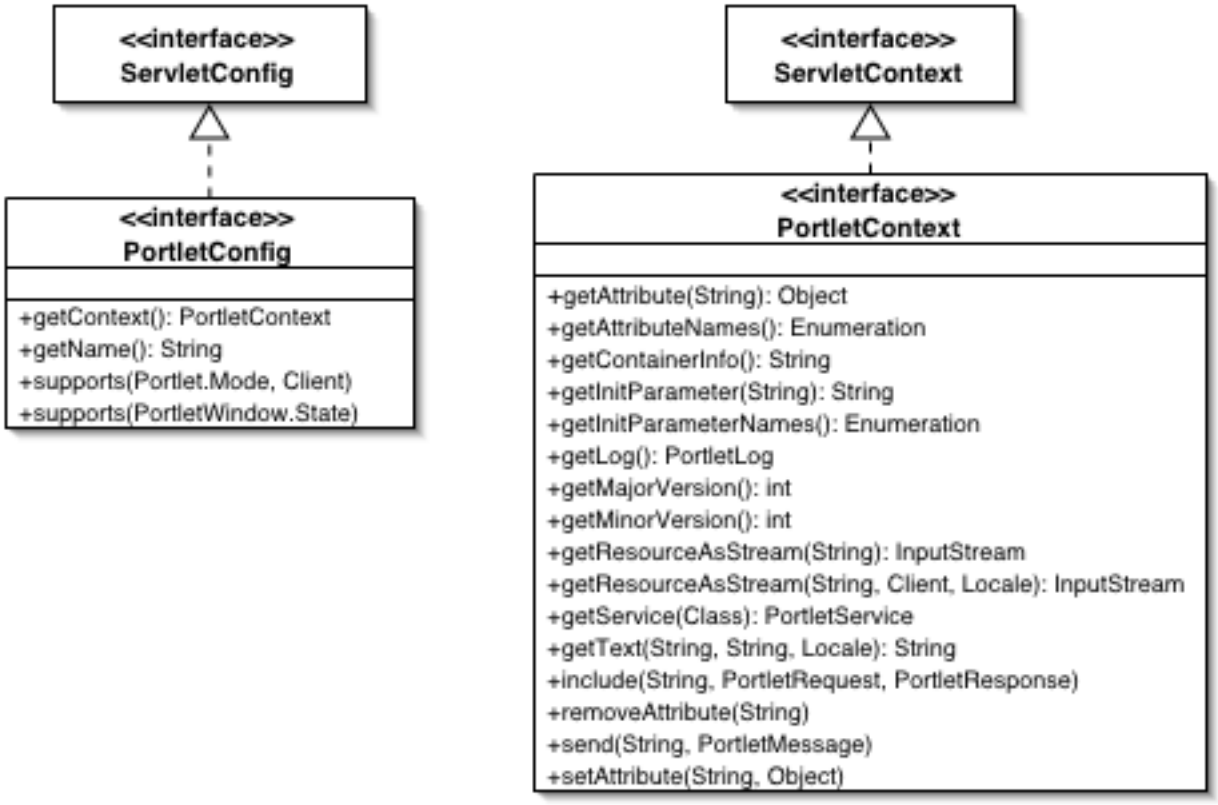

Figure 6: Portlet Extensions to Servlet API

- /WEB-INF/lib/jar files

As mentioned before, portlet.xml contains the portlet application descriptor and web.xml contains the web application descriptor. The layout.xml defines a categorization for how portlets are initially grouped when added to a users collection. More information on the layout structure is provided in the next section. Once completed, the portlet WAR may be installed and made ready for use via the Portlet Manager Portlet described in the Core Portlets section.

5 Portal Presentation

As described earlier, portlets consist of visual interfaces in the form of windows such as those used in any operating system window manager. A portlet frame represents the bounding box of a portlet and may also include within it a portlet title bar. Naturally, the GridSphere framework is concerned with the layout of potentially many portlets to the clients web browser or handheld device. The chief requirements of the layout framework are listed below:

- Must support multiple client devices e.g. web browser, PDA, handheld, cellular phone
- Must allow user customization e.g. look and feel and portlet organization
- Must be flexible enough to support many different kinds of layout models

The first requirement requires that whatever classes are used for rendering they must support output in HTML, and possibly cHTML, WML, or other markups supported by mobile devices.

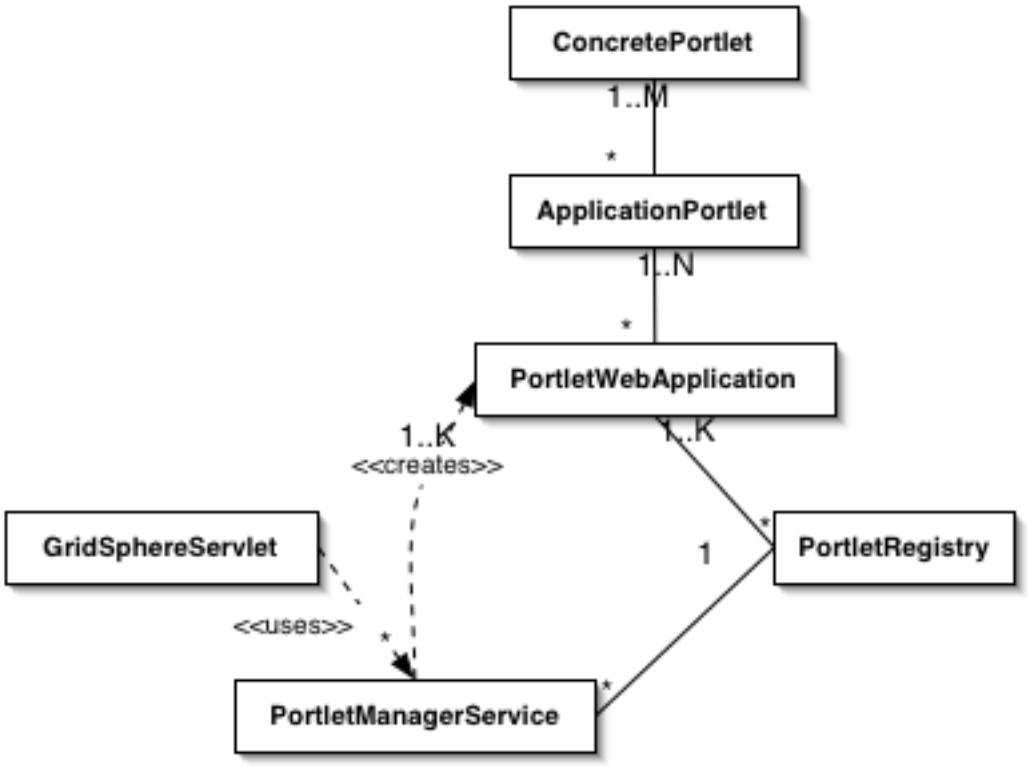

Figure 7: GridSphere Portlet Container

The Adapter pattern can be useful here as a way of providing an abstraction from the rendering interface and the required objects responsible for rendering that are dependant on the client device. The second requirement dictates that the layout components must maintain some state infomation based upon the users preferences. If a portlet is minimized than the layout components required for displaying the portlet must keep track of this information. Another issue is how persistent the state must be. The component classes must save state for the duration of the users interaction with the portal, and presumably they could be serialized to disk such that when a user logged back in they would see the same layout when they had logged out. The third requirement dictates that the component objects should not be tied to any particular type of layout, but rather should work together to be able to create a variety of different template layouts. For this requirement, the Composite design pattern is quite useful and is used in many window toolkits including Java's own AWT and Swing libraries. In the Composite pattern, graphic components can be containers for other graphic components resulting in a nested tree like structure of various graphics components that all satisfy the same base component interface (§9). For instance, a portlet container contains various portlet components. Some of these may be tabbed panes, frames, title bars, panels, and other visual interface that will be needed in the portal. Components such as a panel may contain additional components that may be organized in a particular layout.

5.1 Portlet Layout Components

The portlet components are designed to be lightweight classes and are essentially object wrappers for generating markup. The servlet request and response are passed through each component

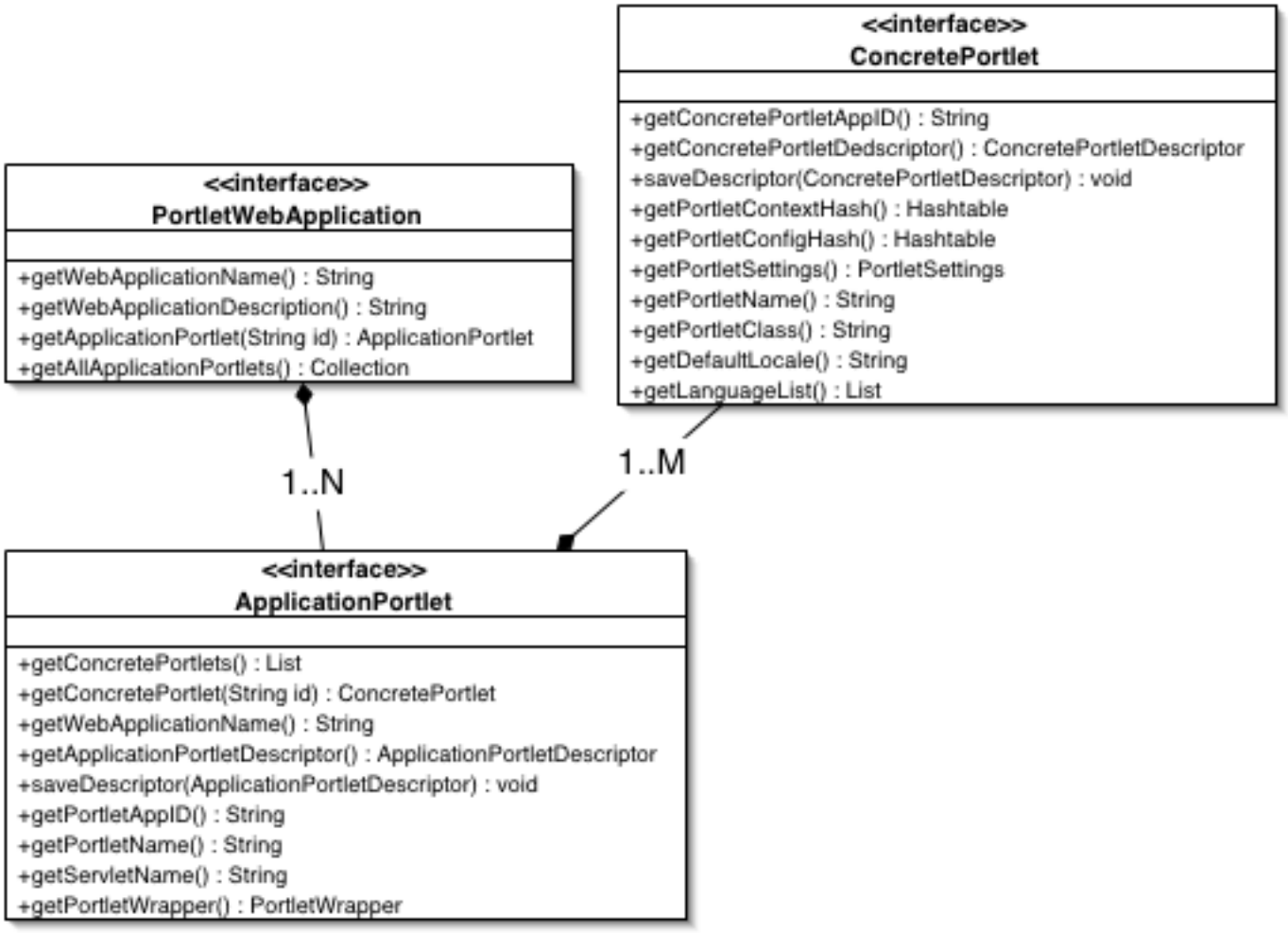

Figure 8: Portlet Container Representation of Portlets

in the tree structure in the doRender method that is the ComponentRender interface and implemented by all layout components. Each component is responsible for deciding how to render itself depending on the client device accessing the portal which can be determined from the servlet request. In general, the hardcoded scripting code uses included CSS files to present a common look and feel, In this way, the CSS file can be modified to present a different look for the corresponding component. In the first version of the layout framework, the following component classes are provided:

- PortletContainer - the root container of all other layout components
- PortletImage - a simple wrapper for including any graphics files
- PortletFrame - the enclosing boundary of a portlet, may also include a PortletTitleBar
- PortletTitleBar - used to display the title of a portlet within a PortletFrame
- PortletTabbedPane - used to categorize portlets within tabs
- PortletTab - provides a tab within a tabbed pane that is itself a container for another component

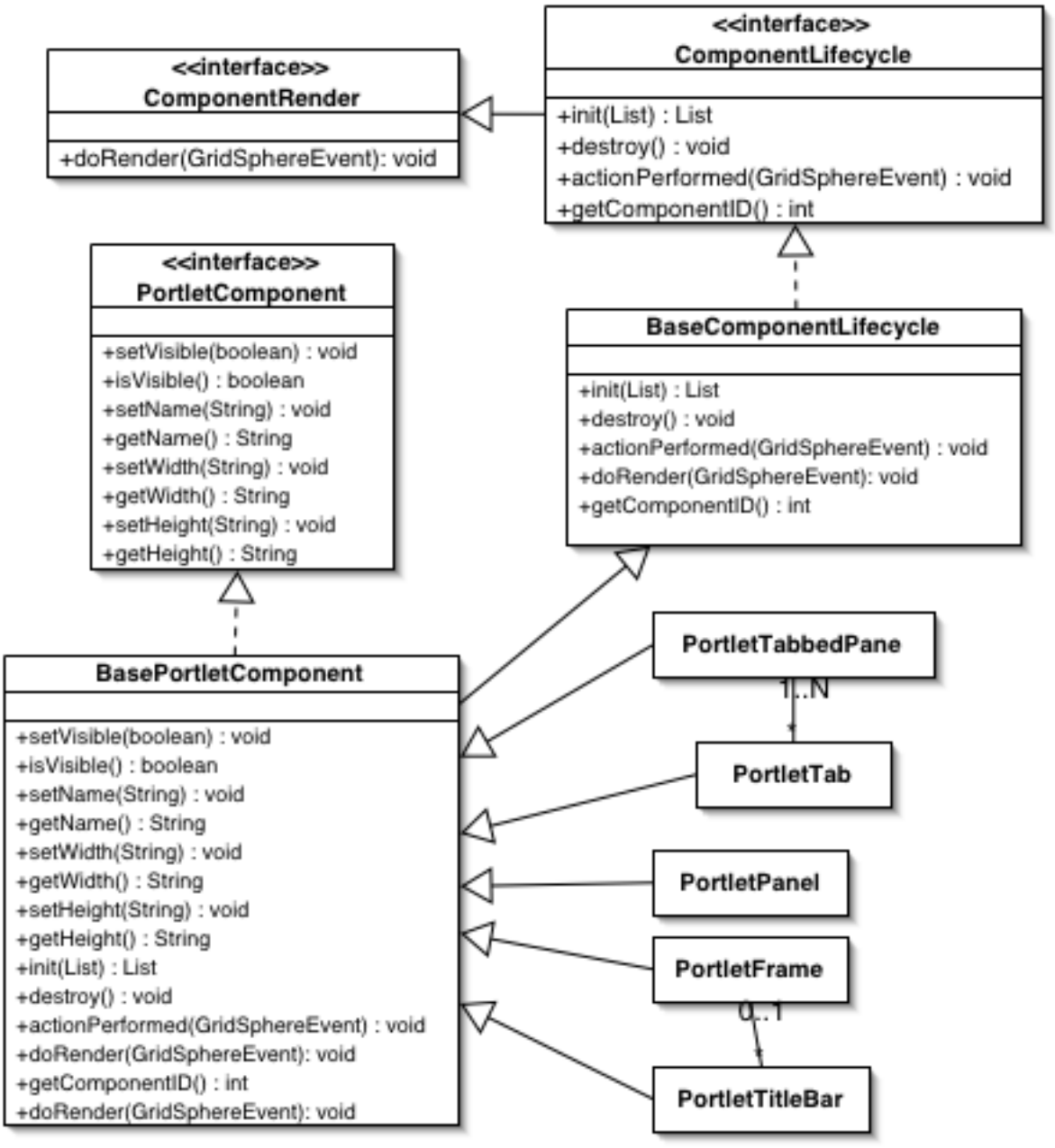

Figure 9: Portlet Layout Components

- PortletPanel (§10) - provides a container for arranging components, generally PortletFrames, in a specified layout

While using the Composite pattern provides a very general approach for constructing and displaying arbitrary layouts, the portal must have a specific look and feel and provide a default pre-defined layout. In the current model, the web site is constructed similarly to many e-commerce sites in terms of navigation. Tabbed panes are employed as a means to categorize portlet groupings. It is imagined that third-party portlet developers will be able to provide a collection of portlets that are packaged as a web application and can be imported by users into

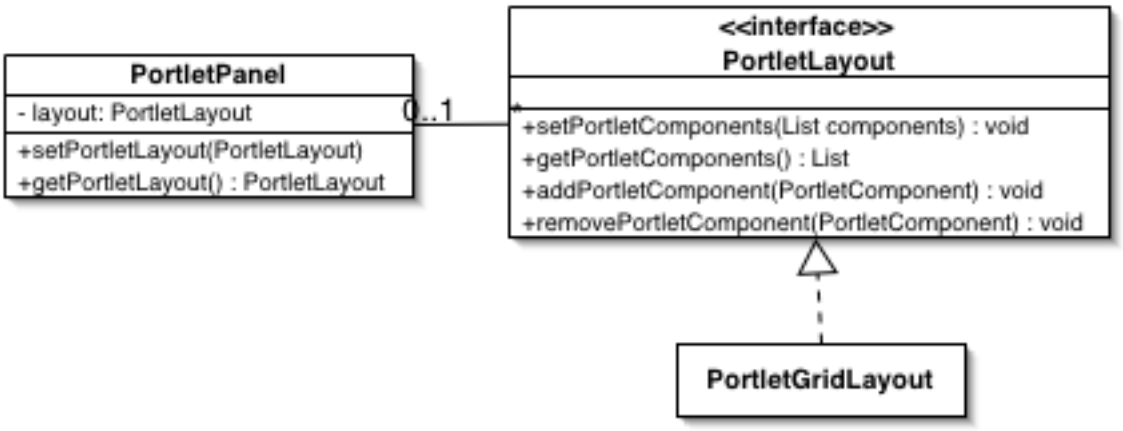

Figure 10: Portlet Panel

their layout resulting in a new tabbed pane containing the portlets. The tabbed pane offers one level of navigability. The flexibility of a generalized tabbed pane component offers the possibility of further adding new tabbed panes to tabs resulting in a greater level of portlet categorization and navigation.

The ComponentLifecycle interface extends the ComponentRender interface to provide support for action handling in the layout components. Because the portlet container is initally deserialized from XML into Java classes using the Castor data binding libraries, the components do not have any association of their children or parents. For this reason, the ComponentLifecycle defines an init() method that gets invoked initially as a mechanism for assigning an identifier to each component in the tree. The init() method serves to create a List from the component tree structure that can then be used to identify a component without having to traverse the tree. Initially, after a portlet container is obtained, the tree is traversed and the doRender method is invoked on every component resulting in markup that is returned to the client. In the markup, the component identifier shows up in any links or forms that are associated with the component. For instance, when the PortletTitleBar generates the links to provide interactivity to the portlet mode or window state icons, the links themselves contain the component identifier so the framework knows which component received an action. Once a user interacts with the portal by generating any event that gets sent back to the portlet container, the component identifier is parsed from the request and the actionPerformed method of the corresponding layout component is invoked. Some events may wish to signal other components as is the case with the PortletTitleBar. In this case, we follow a similar model to the Java event delegation model where components register listeners with the componts they wish to be notified about. For instance a PortletFrame will register a listener with the PortletTitleBar to be notified of window state events. In the case that the title bar receives a ”minimize” action, the PortletFrame gets notified so that it doesn't invoke the portlets doXXX method when rendering, since its in the minimized state. In the case of a maximize window event, not only does the PortletFrame get notified in order to increase its size, but it may also notify its parent if its a PortletPanel containing other PortletFrames so that the other PortletFrames can be placed in the minimized state. In this model, the actionPerformed method is invoked only on the component that triggered the event and events are propagated outward if necessary rather than traversing the entire tree checking for an action in each component.

5.2 Portlet Layout Descriptor

To satisfy the second requirement, components are first unmarshalled from an XML layout descriptor into the actual Java classes, contained by the PortletContainer. An example layout descriptor is shown below, which results in the tree structure in (§11). The users layout in the form of a PortletContainer object may be manipulated via a layout/portlet configuration portlet and the resulting layout descriptor will be marshalled to persistent storage. The PortletLayoutEngine serves as a manager class for PortletContainers and is responsible for loading and saving PortletContainers to a corresponding XML descriptor. Initially, before the user has logged in, the PortletLayoutEngine will return an instance of the "guest" PortletContainer that displays the LoginPortlet. Once the user has logged in, the PortletLayoutEngine will return their customized layout.

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

<container-name theme="xp">GridSphere Portal</container-name>
<portlet-text include="/html/pagehead.html"/>
<!-- Tabbed Panes -->
<portlet-tabbed-pane selected="0" style="menu">
    <portlet-tab>
                <title>Login</title>
                <portlet-panel>
                    <grid-layout columns="40,30,30">
                        <portlet-frame>
                            <portlet-class>org.gridlab.gridsphere.portlets.core.LoginPortlet
                        </portlet-frame>
                    ...
</portlet-layout>
```

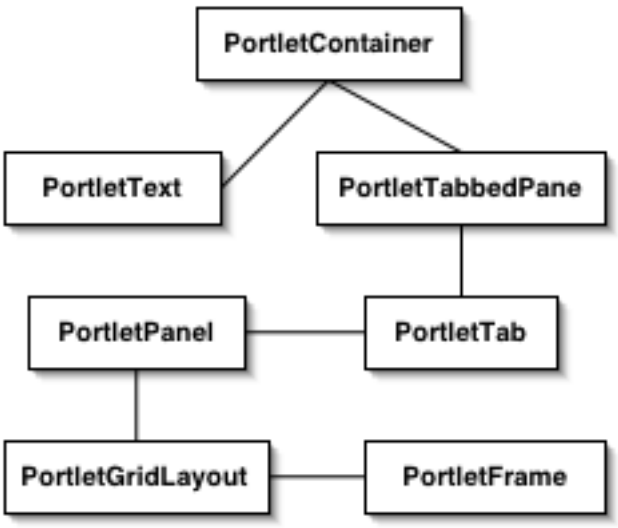

Figure 11: Portlet Layout Descriptor Component Tree

5.3 Portlet Widget Tag Library

Traditionally, HTML is not the most suitable language for the development of user interfaces. The tags as they are supplied are limiting and often browser specific. Moreover, the use of HTML limits the presentation of portlets to web browser that support it. GridSphere provides a value-added web GUI tag library, the Portlet Widget Tag Library (PWTL) for the rapid and elegant development of visual interfaces in new portlets. The goal of hiding HTML in a tag library is two-fold. One, tag libraries can transparently make portlets capable of rendering to multiple devices by taking care of the decision making logic required to format an appropriate display. Second, easy to use tag libraries that offer a rich feature set takes away much of the pain associated with repetitive HTML scripting. Currently, the PWTL offers mostly wrappers around existing HTML elements such as input fields and forms. The form tag, in fact, saves several lines of code by creating the appropriate action attribute in the form tag. More sophisticated tags can be developed using the MVC approach of separting out a model from the visual interface that uses it. For instance, a ListBox Tag can render objects appropiately in a list box based on a ListModel that describes the objects to be displayed.
In addition, if one wants to develop a portlet for a specific device, it can query the device from the *Client* interface defined in the Portlet API. The Client is made available from the PortletRequest and defines *Capabilities* that are intended to classify the device presentation capabilities e.g. HTML version, javascript, tables, etc.

6 Portlet Services Framework

The Portlet services framework defines an architecture for the development of portlet services that provide functionality to Portlets. Portlet services are designed to individuate the functions provided by Portlets from the services with which they need to interact with. In GridSphere, services are used to manage everything from layout preferences, user profiles, user access control, security credentials and Grid services.

7 Portlet Services Framework

The portlet services framework defines an architecture for the development of portlet services that provide functionality to portlets. Portlet services are designed to individuate the functions provided by portlets from the services with which they need to interact with. In GridSphere, services are used to manage everything from layout preferences, user profiles, user access control, security credentials and Grid services.

7.1 Portlet Services API

We borrow from the IBM WebSphere service interface to build a services architecture that we believe will allow for future integration with a standardized Portlet API. The UML object diagram of the services framework is shown in Fig. (§12):
A portlet service is created from a PortletServiceFactory. A PortletService defines a blank interface that is enhanced by the PortletServiceProvider interface which defines a lifecycle consisting of init and destroy methods used when the service is initialized and shutdown. A portlet service XML properties file is used to provide class information and initialization parameters that is accessed via the PortletServiceConfig. The PortletServiceFactory is also responsible for initializing and destroying services. The structure of the services properties file is shown below:

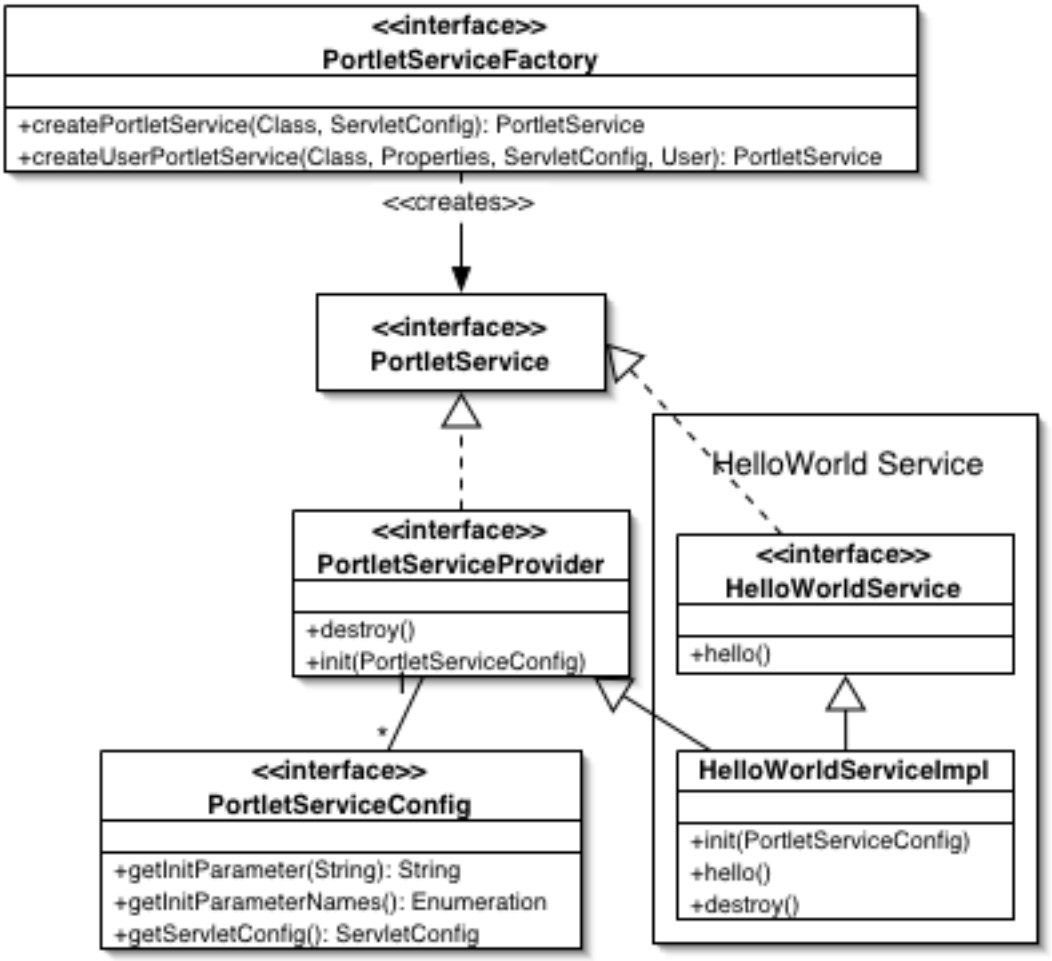

Figure 12: Portlet Services UML Diagram

```
<portlet-services>
    <service>
        <name>Login Service</name>
        <description>Provides Login Capabilities</description>
        <interface>org.gridlab.gridsphere.services.user.LoginService</interface>
        <implementation>org.gridlab.gridsphere.services.user.impl.LoginServiceImpl
</implementation>
    </service>

    <service>
        <name>Access Control Manager Service</name>
        <user-required>true</user-required>
        <description>Provides Authorization Capabilities to Portal Users</description>
        <interface>org.gridlab.gridsphere.services.security.acl.
AccessControlManagerService</interface>
        <implementation>org.gridlab.gridsphere.services.security.acl.impl.
AccessControlManagerServiceImpl
</implementation>
    </service>
 ....
 </portlet-services>
```

Also shown is an example of creating a HelloWorld portlet service. First a HelloWorldService interface that subclasses the PortletService interface would define the methods that are part of the service e.g. hello() and then a HelloWorldServiceImpl would provide the concrete implementation of the service interface. The HelloWorldServiceImpl is responsible for implementing the PortletServiceProvider interface and provides the necessary startup and shutdown routines. In general services are usually instantiated when they are first needed unless they are used by the Portlet container. Portlet developers can obtain an instance of a specified portlet service via the PortletContext object which offers a getService method.

7.2 Portlet User Services

The service framework has also been augmented to provide support for *user services*. Under this model, services can be developed that perform method-level access control checks. As an example, the UserManagerService has a method deleteAccount(). Because the UserManagerService is a user service, The PortletServiceFactory will return a UserManagerService appropriate for the passed in User object, such that only a "super-user" can invoke this method and for other types of users, the method would throw an AccessDeniedException.

7.3 Role Based Access Control

The notion of Role Based Access Control (RBAC)[18] is quite popular in many enterprise application servers. In the RBAC model, users have *roles* that provide various *permissions*. In the GridSphere model we have four predefined roles:

- Guest – a guest is an anonymous user who has restricted access to the portal.
- User – a user is anyone who has an account and has gained admittance to the portal

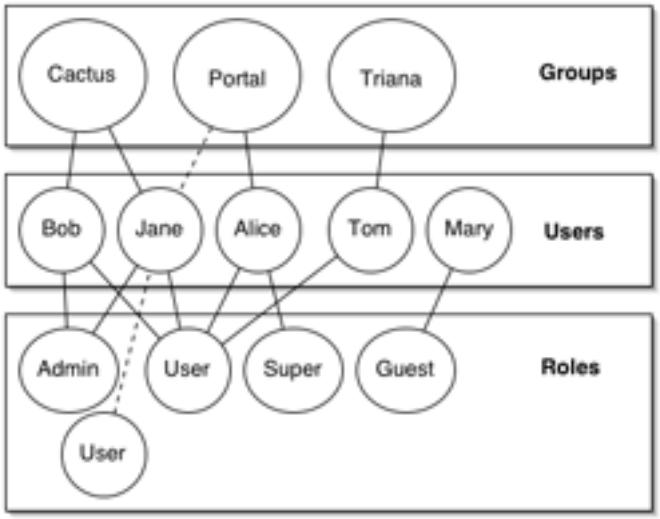

Figure 13: Relation of Users, Groups and Roles

- Admin – an admin is considered the leader of a particular group and can change certain configuration settings for users in the group.
- Super – the super user has "root" equivalent permission on the portal to edit and configure settings and preferences for all users in all groups. Typically, only the portal administrators have this role.

The relation of users, groups and roles is shown in Fig. (§13): A group is a representation of a project that ties together many distributed people wishing to collaborate in a common area e.g. Cactus. Users can belong to one or more groups. In the diagram, Jane is both a Cactus member and a Portal member. Users can have multiple roles within a group. For instance Jane has the role of Admin and User in the Cactus group, but is only a user in the Portal group (as shown by the dotted line). Portlets are defined to be accessible to a particular group in the Portlet configuration file (see Fig. (§**??**)). All information about the relationship between users, groups and roles is maintained by the `AccessControlManagerService`.

7.4 Persistence Management

Many interfaces and classes in GridSphere represent persistent entities. For example, an object implementing the User interface contains persistent information about an end-user while a Resource object may contain information about some particular resource to which that end-user wishes to submit a job.
However, while the Java language provides mechanisms (JDBC) for supporting persistence in SQL based databases, managing persistence within Java can be tedious and repetitive for programmers. In each case create, read, update, and delete (CRUD) operations must be implemented. In developing a persistence framework, the following requirements must be satisfied:

- The ability to marshall objects to/from multiple data formats and storage services for each type of persistent operation we wish to support, and this marshaling should be transparent from the perspective of the class designer.

- The ability to dynamically lookup objects based on object attributes, as opposed to say, using SQL to search for objects based on the column values of their underlying representation in relational tables.
- The ability to cache objects in memory, as performing reads or writes to an RDBMS with every persistence method call can result in poor application performance. And again, we would like this management to be transparent from the perspective of the class designer.
- The ability to maintain persistent data even as class definitions change. In a production environment such as the GridLab Portal, persistent data must not be lost between updates to the GridSphere software.

7.4.1 PersistenceManager

Our persistence manager provides classes and methods for developers to handle objects and their persistence.

7.4.2 JDO

JDO solves the problem of marshalling/unmarshalling objects to some kind of storage. It aims to be transparent to the developer for making different objects persistent and provides methods to create/read/update/delete objects in the persistent storage. Furthermore it handles the caching of objects automatically to increase performance of the application. Every time a object is requested from the storage the PM delivers the object from the cache if it is there and still valid instead of using costly databaseconnections. If the object is invalid or not anymore in the cache it is requested, delivered and stored in cache. JDO even includes the possibility to provide different caching algorithm (like last recently used or timebased). For accessing the objects the Object Query Language (OQL) is available to perform any kinds of queries to the PM. The OQL is much like SQL but it works directly with real javaobjects instead of SQL commands, the advantage here is that the developer does not need to know anything about the non-java SQL code, he just deals with his well known javacode.
We are planning to implement a thin layer of our own persistence. With that we can easily switch the persistence manager itself so we will use Castor for now and later on we will very likely swap to the Jakarta JDO implementation OJB[39].

Objects needing caching and persistence will be stored in a RDBMS. Objects which marshall from other sources like from LDAP or plain XML files are are just cached and not made persistent, since we won't write to LDAP servers (like GIS) and don't want to write to the local filesystem (there might be some cases but we want to minimize them because of performance reasons). All objects which need persistence need to be saved in a RDBMS. We will provide a unified interface for accessing the different kinds of storage types (either read-only or read/write).

7.4.3 Change Management

JDO does not, however, address how to maintain persistent information across changes to class definitions. While we could implement versioning in our class definitions, we would still have to migrate older data representations to correspond to new class definitions. In short, there are two problems we must deal with:

- Coherence between class definitions and underlying data model.

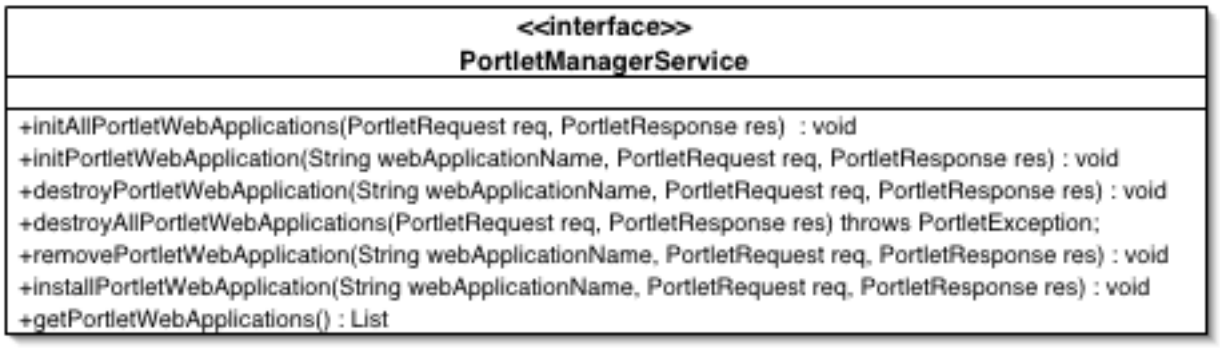

Figure 14: Portlet Manager Service

- How to upgrade existing databases to support newer data models.

We can address the first problem by developing unit tests to verify that persistence operations on objects behave as expected. Addressing the second problem is more complicated. Fortunately, SQL provides mechanisms for modifying table definitions of an existing database and we can use these mechanisms to upgrade the GridSphere database along with its code-base. In our case, we have created a simple procedure for enabling developers to apply modifications to SQL table definitions in step-wise fashion along with new class definitions commited to CVS using the supplied installation script.

7.5 Core Services

Core services are those services that are used by either the core GridSPhere framework including the portlet container or the collection of core portlets described in the next section that provide basic user/account management.

7.5.1 Portlet Manager Service

The `PortletManagerService` provides various portlet management methods for installing, removing, and destroying portlets (§14). The PortletManagerService is used by the frameowrk itself and the Portlet Manager Portlet to manage deployed portlet web applications at run-time.

7.5.2 Login Service

The `LoginService` is essentially the bootstrapping service used to retrieve a User object based on a provided name and password. The LoginService is not a user service since it is assumed that user does not yet exist.

7.5.3 Access Control Manager Service

The `AccessControlManagerService` is used to add users to groups, grant (and revoke) roles and provide other services with the necessary access checks. Since the `AccessControlManagerService` is a user service, methods can invoke the appropriate logic depending on the users role. For example, the createGroup method may only be called by a user who has the super user role, or the approveGroupRequest may be invoked by an admin to provide another user with the role of admin.
The following interface defines the AccessControlManagerService:

Figure 15: Access Control Manager Service

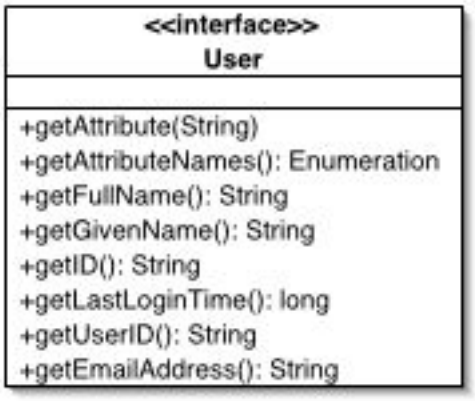

Figure 16: User Interface

7.5.4 User Manager Services

The `UserManagerService` (§17) is used to manage the creation, deletion and loading and saving of user accounts. The UserManagerService is a user service and once instantiated by a particular user will only offer the functionality required by the users role So for example, an ordinary user cannot approve any account requests or retrieve User objects beyond their own.
We choose to adopt the `User` interface provided by the IBM WebSphere Portlet API and shown in Fig. (§16). As in WPS, a `User` may be obtained by a Portlet from the `PortletRequest` (see. Fig. (§5))

7.6 Service Testing Framework

To insure the proper functioning of the developed services, regular unit tests will be performed which will test all service methods to ensure they return expected results. In traditional black box unit testing, it is very difficult to test stateful objects because it assumed that service methods could be invoked in any order and should always return the same verifiable results. While it is generally good software design practice to maintain as little state as possible for these reasons, the services must contain some amount of state information. For instance, the UserManagerService and the AccessControlManagerService assumes that a root, or super user exists. The situation becomes increasingly difficult in the case of a Grid enabled service where it is assumed that a test user exists with a valid credential. The adopted approach is to test services like an onion starting from the inside layer where the first test just tries to create a root user. Assuming that test succeeds, the next test can use the previous tests createRoot method within its testing harness e.g. the setUp and tearDown methods which get invoked before and after a test completes. The testing structure is shown in Fig. (§18). The major points are that certain service tests that assume some state information must inherit from base tests which are responsible for th eproper creation of the state information and can then be continually subclassed by other service tests that wish to extend the tests for more complex functionality e.g. the proper functioning of a Grid enabled service.

8 Grid Services

To be implemented in GridSphere 1.1 2Q03

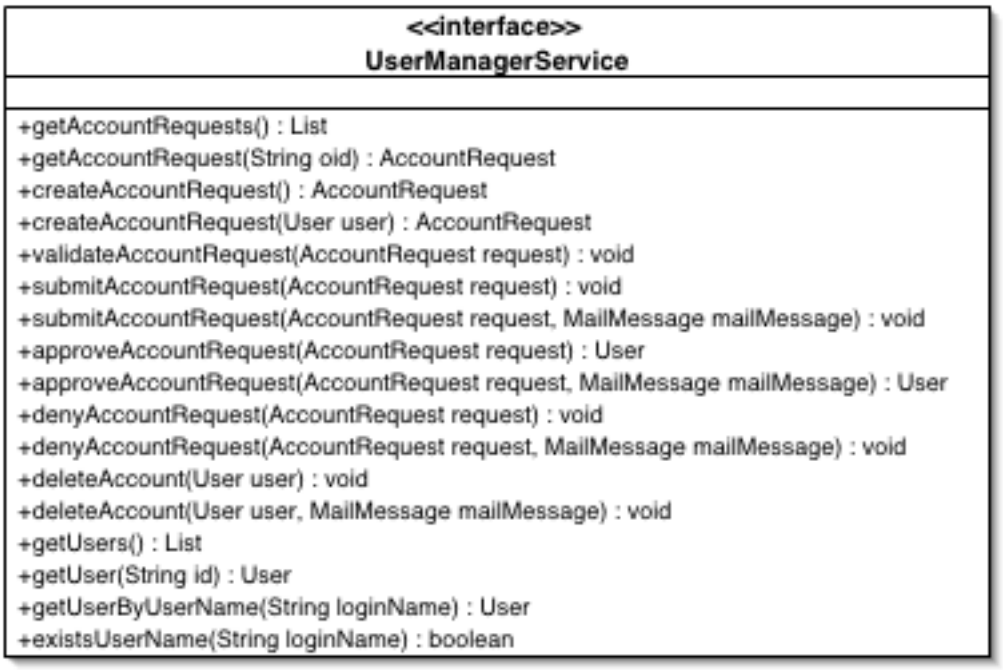

Figure 17: User Manager Service

A major goal of the GridLab project is to develop a Grid Application Toolkit (GAT) API that will provide application programmers with an API for easily accessing Grid services without reference to the underlying Grid fabric. In this model, the GAT API is capable of using adaptors to achieve the necessary Grid functionality. We are commited to providing a Java based implementation of the GAT API that will allow portlet services to access GridLab developed services using the GAT API. In our propopsed GAT implementation, adaptors will be provided that support both Globus 2.2 and GT 3 services.

9 Integration with Web Services/OGSA services

The Open Grid Services Architecure (OGSA) is a collaboration between the Globus development group and IBM to provide an open framework for the development of Grid services based on standards developed within the web services community. Grid services are defined to be web services that support an extended set of interfaces.
Currently, the interfaces defining a Grid service are evolving within the OGSA community and may be extended to support concurrency and authorization. Grid services are also concerned with the creation of transient service instances that need to be created dynamically and cleanly shutdown when no longer needed. A serviceType is a Grid service definition containing a set of one or more portTypes supported by the Grid service along with additional versioning information. Dynamically created Grid service instances also have a Grid service handle (GSH) associated with them that contains a unique name used to refer to that service instance. A GSH is represented as a globally unique URL and can return a Grid service reference. A Grid service reference (GSR) is used to maintain information about a Grid service instance that may change over a service's lifetime and is represented as a WSDL document with extensions to reference a serviceType and expiration information.

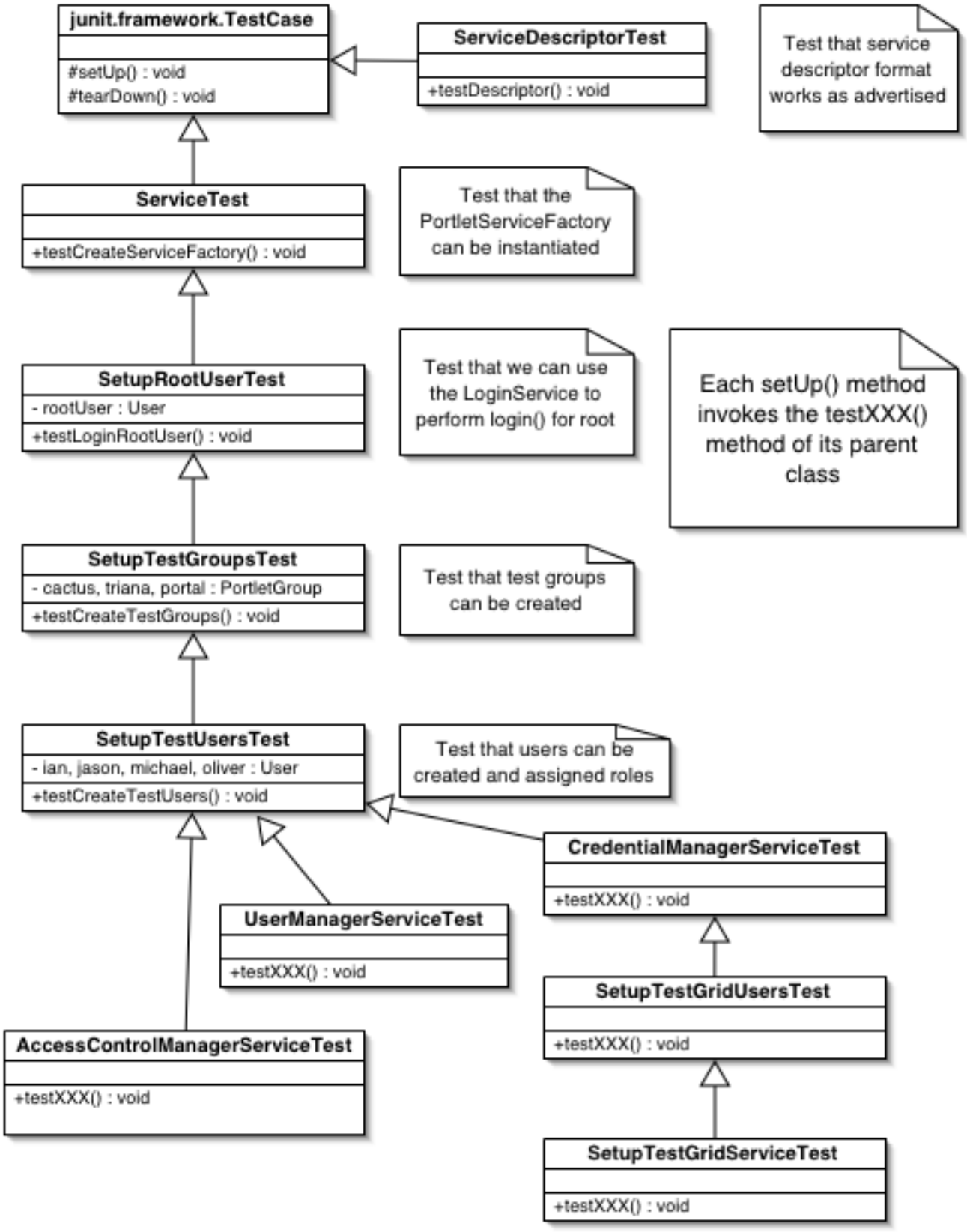

Figure 18: Testing Framework UML Diagram

The Grid Service Specification (GSS) document specifies how clients create, discover and interact with Grid services and provides a detailed technical description for how the GridLab portal will interact with Grid services developed by other Work Packages. The GSS is currently under review and still in the refinement process. The GSS will be adopted as it becomes more standardized.
Currently, an alpha release of the OGSA software exists in Java and is being tested by early adopters. The codebase leverages the Apache AXIS [34] SOAP and WSDL implementation and uses the Tomcat servlet container as a hosting environment. GSI security is enabled in Axis and Tomcat using the non-open source IAIK SSL libraries provided in the Java CoG kit.
Under the OGSA framework, a client first creates an instance of a service that is specified as a URL and then gets back a GSH expressed in the form of a URL that is used to invoke the service instance. The samples supplied with the OGSA code base demonstrate simple service factories which are used to create service instances whose methods can be invoked by sample clients. As of this writing, the supplied user's documentation is quite sparse. Hopefully, as OGSA gains favor among the user community and in the GridLab project, more detailed instructions on creating new services and steps for generating clients will follow. In the GridLab portal model, OGSA client services will be part of the GridSphere service library.

9.0.1 Credential Management Service

Access to Grid resources in the GridSphere architecture is provided by the Grid Security Infrastructure (GSI)[2]. GSI provides support for mutual authentication using X.509 *proxy* certificates. A proxy certificate provides a temporary certificate/key pair that provides single sign-on access and can be used to access resources on a users' behalf. In order to provide access to Grid enabled resources, GridSphere must have access to a user's valid proxy credential. The Myproxy [20] online credential repository provides a secure repository where users can delegate credentials of a limited lifetime for later retrieval via the portal. The credentials are stored and retrieved using a supplied username and password by a Credential Management Service (CMS). The CMS supports multiple credentials per user. In order to support resources both in the GridLab testbed and other resources, users must be able to select a particular credential to be used for accessing the desired resource. We plan on providing a Portlet that will allow a user to configure which credential should be used for a given set of resources. After this initial configuration step, it should be transparent to a portal user which credential is needed for a particular resource. Credentials are loaded directly into memory and promptly destroyed when a user logs out. In the case that users do have access to their credentials, GridSphere will be able to handle passing uploaded credentials to the CMS.
The Credential Management Service UML diagram in Fig. (§19) lists the capabilities provided. The credentialManagementService is also implemented as a secure portlet service allowing it to provide capabilities for portal administrators to set the allowed credential subject names (distinguished names) that are accepted by the portal. Portal administrators can also reconfigure the myproxy server settings and the lifetime of accepted proxies. Finally, a mapping between hosts and user credentials is managed using the assignCredential method and is also made persistent.
In addition to the CMS, tools will be provided making it easier for scientists to manage their credentials including delegating credentials to the Myproxy repository. We plan on enhancing the JMyproxy[36] GUI tool for our purposes. We also wish to work with the Myproxy project to add support for managing multiple user credentials.
We wish to work with the Security WP (WP-6) to ensure that the configuration of the portal and Myproxy server meet the requirements of the GridLab testbed. We also plan on becoming

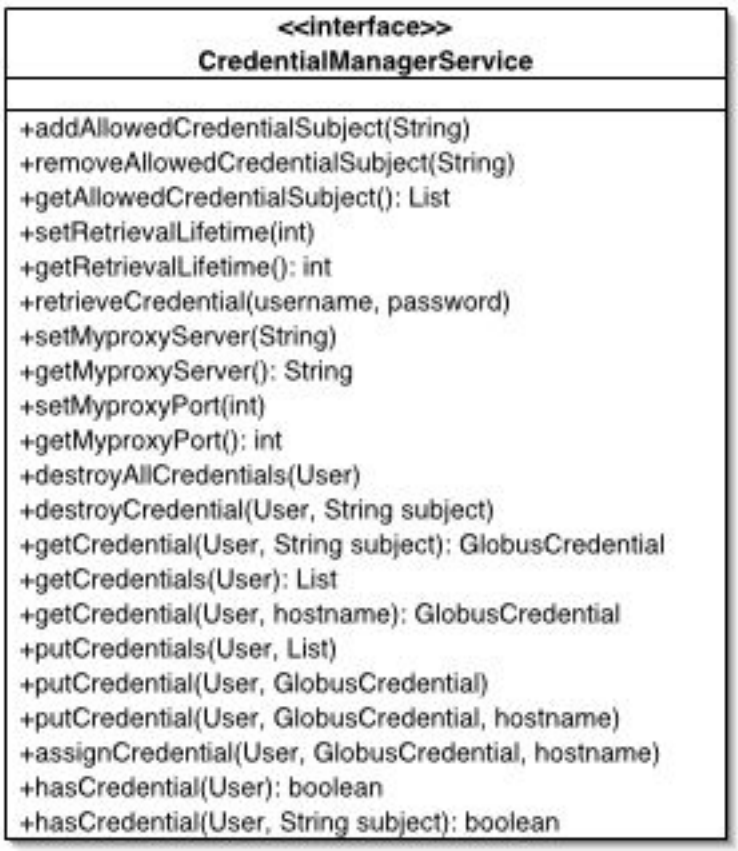

Figure 19: Credential Manager Service UML Diagram

early adopters of the Community Authorization Service) as/when it becomes available.

9.0.2 Resource Management Services

Resource Management services are used to provide job submission capabilities to portal users. Several technologies for the submission and monitoring of running jobs will be incorporated into the portal services framework. Resource Management services will include support for Globus based job submission and the GridLab Resource Management System (GRMS) developed by WP 9. A preliminary UML object class diagram is presented in Fig. (§20):
The Job Submission Service interface provides methods for submitting, stopping and querying jobs. In addition, a background thread can monitor job status periodically and provide the user with email notification when the job completes or allow for the update of job status to a user's profile even after the user logs out.

9.0.3 Resource Description Language

Currently, the Globus middleware relies upon the Resource Description Language (RSL) for specifying parameters required for submitting a job. Some of the more useful parameters are listed:

- executable
- arguments
- count (number of processors to use in a parallel job)
- type (can be single or parallel)
- max. wall clock time
- max. CPU time
- max. memory
- min. memory
- queue name

While RSL will continue to be supported for resources supporting Globus 2.0, other services including the GRMS may choose an XML based resource description language. The Java CoG kit also includes prototype support for the conversion of RSL to an XML based format.

9.0.4 Resource Profiles

The concept of resource profiles is used to describe the supported job submission services on a particular compute resource. For instance, the portal needs to keep information about which resources have Globus gatekeepers deployed or GSI enabled SSH access, which versions, etc. Wherever possible, we rely upon the Information Services provided by WP 10 to maintain this information and make it available to the portal. However, it may not be feasible to query the GridLab information servers every time a request for a job submission is made. For this reason, the portal will cache information as necessary and provide administrators the ability to update the cache. The list of information necessary regarding job submission is listed:

- Service name – e.g. Globus Gatekeeper or GSI SSH

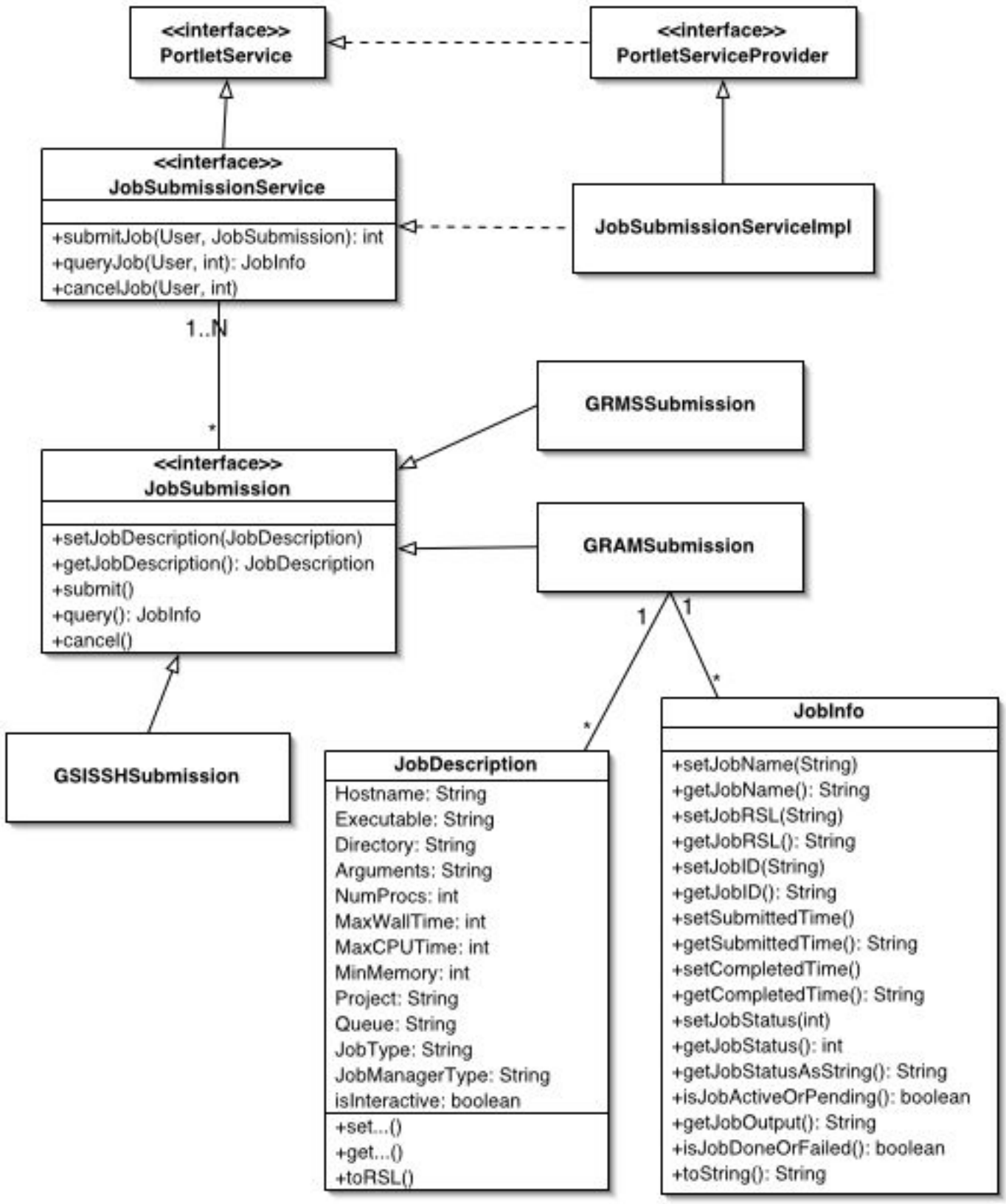

Figure 20: Job Submission Service UML Diagram

- Software version
- Port that service is running on
- Names of queues available on resource
- Additional resource information provided by Globus GRIS

9.0.5 Grid Resource Management System

The GRMS system developed under WP 9 is a resource broker that attempts to submit a provided job on the best resource available based on any criterion supplied by the user. While GRMS is currently based on Java/Corba technologies, it is the goal of WP 9 to develop a web service or OGSA enabled service for accessing GRMS. Currently, WP 9 has developed a portal page to access GRMS and it will be integrated into a Portlet in the GridSphere portal.

9.0.6 GRAM Job Management

Globus Resource and Allocation Management (GRAM)[4] provides an API that uses GSI and is used to submit jobs to Globus gatekeepers. The gatekeeper is responsible for starting up a jobmanager that manages the execution of the job and can handle various scheduling systems including LSF, NQE, LoadLeveler, or PBS to name a few. The goal of the Globus gatekeeper/jobmanager is to hide the differences of local scheduling systems from the user and the portal. The GRAM API also provides mechanisms to cancel jobs and query jobs for status e.g. pending, active, or finished.

9.0.7 GSI SSH Job Submission

GSI enabled SSH allows a standard proxy credential to be used to authenticate to a SSH daemon running on a compute resource. OpenSSH has been modified allowing for the more secure version 2 protocol to be used with GSI. Currently, no open source Java implementations are available of OpenSSH, but a version of MindTerm SSH has been augmented to provide support for GSI.

9.1 Data Management

Data management services will provide users with the ability to efficiently and easily migrate data between resources (third-party file transfer), as well as the ability to download data or a subset of the data to their client machines. The Data management services will use underlying Grid technology such as GridFTP[5] as well as data management services developed by Data Management and Visualization WP 8.

9.1.1 GridFTP

The Grid FTP protocol provides an optimized data transfer library and a specialized set of FTP commands for performing data channel authentication, third-party file transfers, and partial file transfers. GridFTP relies on GSI to perform mutual authentication using X.509 proxy certificates. The Java CoG library provides a Java API for using GridFTP. By using the adaptor pattern to wrap existing CoG GridFTP classes, they can be made more suitable for satisfying the specific needs of GridSphere. The following class diagram, Fig. (§21), illustrates a GridFTPService that will be provided by GridSphere and accessible to Portlets.

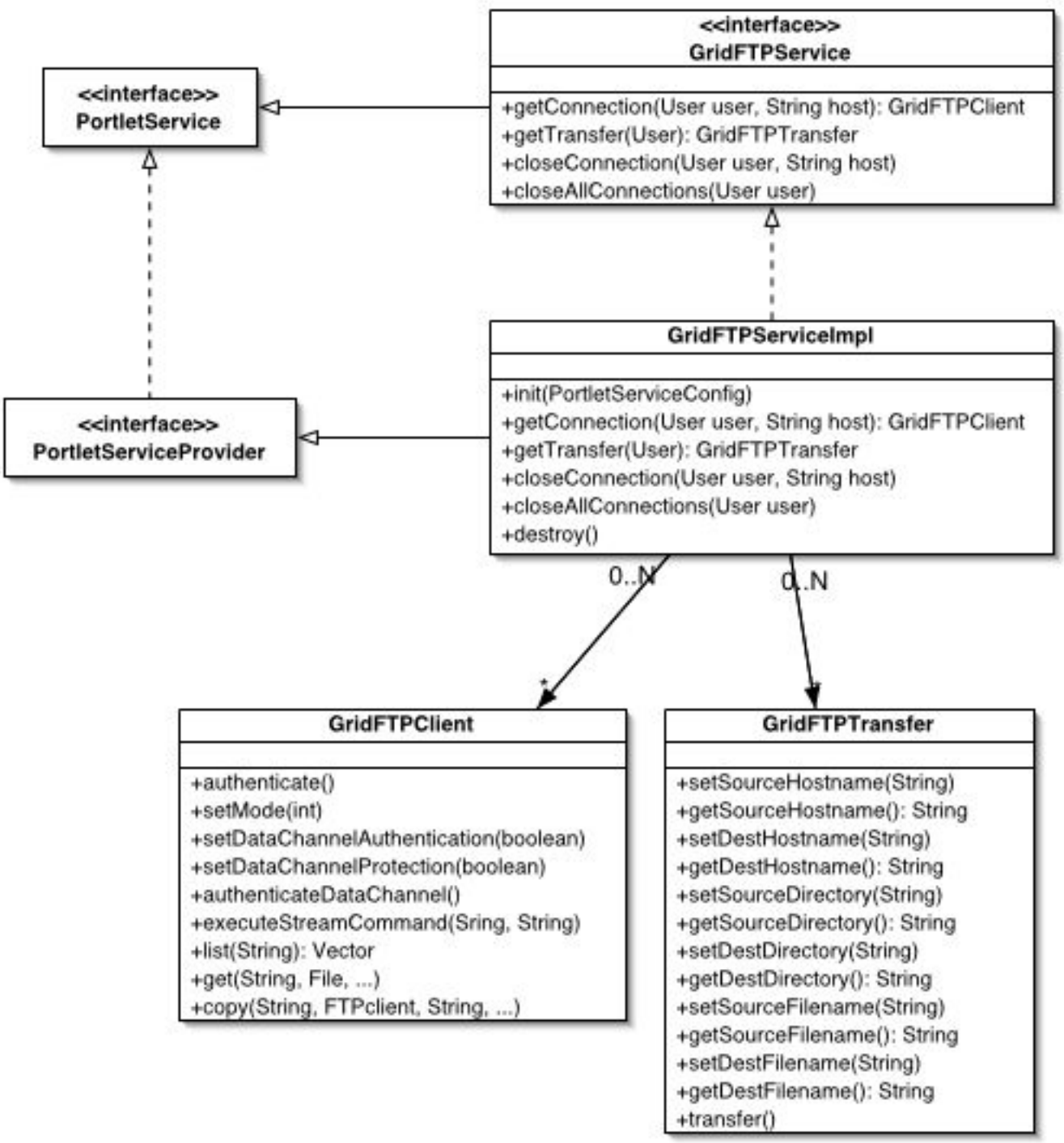

Figure 21: GridFTP Portlet Service

9.2 Information Services

A critical component of GridSphere is to provide users with information about the GridLab testbed and additional machines that may be external to the testbed. The Information Services work package (WP 10) has proposed the Grid Information Services (GIS) architecture for deploying information services on the GridLab testbed. Information services in GridLab will be used to obtain both static and dynamic information on software and hardware resources.

9.2.1 Grid Resource Information Services

The GIS supports the Lightweight Directory Access Protocol (LDAP) as the communication protocol used to query information services. The Globus toolkit provides an implementation of a Grid Information Service, known as the Grid Information Service (GIS) using OpenLDAP, an open source LDAP server. Also, known as MDS-2, the GIS supports secure authentication based upon GSI and the Simple Authentication and Security Layer (SASL). The latest Java CoG toolkit provides support for the secure MDS-2 using either the Java Naming and Directory Interface (JNDI), or the open source Netscape/Mozilla Directory SDK [44]. An open issue is whether connection pools can be maintained for users to reduce resource consumption, or if users need to re-authenticate for every secure query. Additionally, it may be possible to provide both a secure and non-secure directory tree in the LDAP object class hierarchy.
Fig. (§22), illustrates a first pass at a design of a Grid Information Service to be used by the portal. The GIS service provides a single method queryGIS which takes a GISQuery object describing the query to make and returns a GISResults object in the form of an LDAPSearchResults object if the Netscape SDK is used.

10 GridSphere Portlets

The GridSphere portal provides a portlet framework and a set of portlet services described earlier. However, in order for GridSphere to be usable it must also supply a basic set of Portlets that provide some level of functionality for end-users and portal administrators. The various Portlets that will be developed are described below:

10.1 Core Portlets

10.1.1 Portlet Manager Portlet

The Portlet Manager Portlet is essentially the root of all portlets. The Manager allows authorized users to start, stop, remove and reload portlet web applications. Additionally, the Manager portlet provides an interface for uploading and initializing a portlet web application. Using this interface, the portal need not be restarted. Portlets can be deployed an dundeployed as necessary while GridSphere, the portlet container continues to run. This feature is currently Tomcat specific and uses the Tomcat manager features of Tomcat 4.1.18. It is envisioned that a standard API for the dynamic reloading of servlets will make this a common feature in all servlet containers.

10.1.2 Login Portlet

The Login Portlet provides portal users the ability to authenticate to the portal and obtain a valid session, based upon a provided username and password. The name and password are checked by a PasswordManagerService and then the user is either authenticated or denied access.

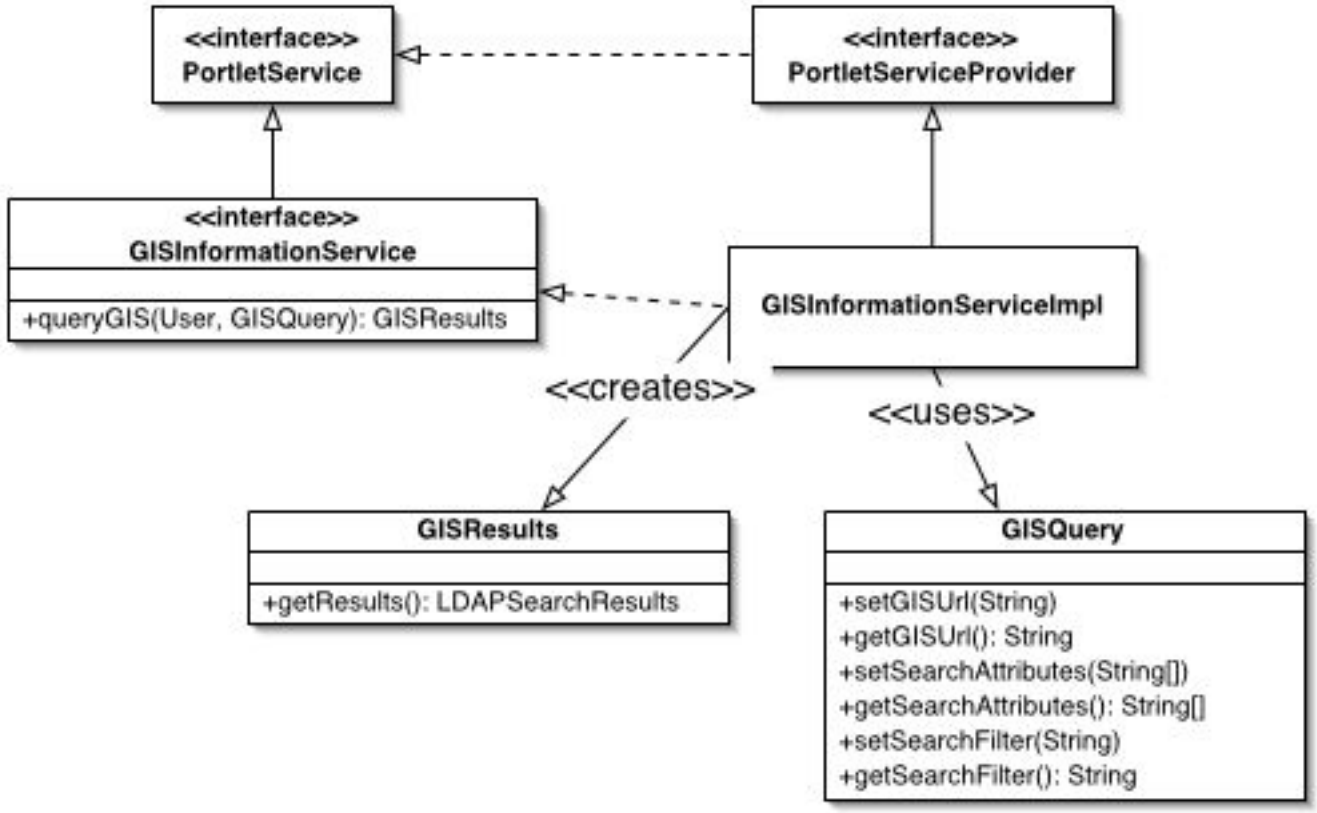

Figure 22: Grid Information Service

It should be mentioned that all access to the LoginPortlet will be restricted to HTTPS to ensure that passwords are not transmitted in the clear.

10.1.3 Account Request Portlet

The Account Request Portlet allows users to register with GridSphere and obtain an account. The process for registering with GridSphere is similar to Yahoo, E-Bay or any other portal. A form is provided that allows the user to enter the following information:

- Full name
- Desired login name
- Organizational affiliation
- Email address
- (optional) Groups they wish to join

Optionally, a new user may request groups they wish to join. A notification will be sent to admins of the groups to allow them to be added. Once a users request is processed, an account will be created (or denied) by the portal administrator.

10.1.4 Account Management Portlet

The Account Management Portlet offers various account management services for a particular role. For the role of super, the following capabilities are provided by the Portlet:

- Modify any users role
- Modify any users groups
- Account deletion
- Account request approval and account creation
- Logout a user
- Display portal activity among all users

For the role of admin, the following capabilities are provided:

- Approve user requests to join group
- Promote other users in group to admin role
- Revoke group access to users
- Display activity of users in group

The user and guest role do not have access to the Account Management Portlet.

10.1.5 Portlet Configuration Portlet

The Portlet Configuration Portlet provides Portlet configuration capabilities. Users with the super role may perform the following:

- Modify Portlet configuration files e.g. change cache parameters, group ownership, title, etc.
- Modify default layout settings including menu bar categories and Portlets within categories

All other users can perform the following:

- Customize personal Portlet settings including locale, text, title, and background color
- Configure layout type
- Edit menu bar categories and add/remove Portlets from categories

10.1.6 Portlet Subscription Portlet

The Portlet Subscription Portlet displays a list of Portlets that a user can subscribe to. For instance, if a new Portlet, CoolPortlet, is deployed to GridSphere, then it becomes possible for a user to select the CoolPortlet to add to their personal portal pages. Similarly, users can also unsubscribe from Portlets that are in their subscription list. Since Portlets are associated with groups, only users in those groups can add the Portlets to their subscription. However, many core Portlets including this one are configured to be available to all users even if they have no group affiliation.

10.2 Grid Portlets

To be implemented in GridSphere 1.1 2Q03
A collection of Grid enabled portlets wil be developed that make use of the underlying OGSA frameowork provided by GridSphere.

10.3 Credential Management Portlet

The Credential Management Portlet provides users with credential configuration capabilities as well as the ability to display credential information available from the Credential Management Service.
The super role has the following abilities:

- Reconfigure the Myproxy server and port used and lifetime of accepted proxies
- Reconfigure the accepted certificate subject names
- Destroy any certificates in the Credential Management Service

The user and admin role can perform the following functions on their own credentials only:

- Display credential information available from the Credential Management Service
- Retrieve and refresh credentials

10.4 Credential Configuration Portlet

The Credential Configuration Portlet is intended to perform simple authentication tests using the user's credentials against various resources and various services. The net result is a mapping of credentials to hostnames that is stored on a per user basis by the Credential Management Service. Other portlets requiring authentication to resources will be able to obtain the correct credential from the CMS.

References

[1] Gamma E, Helm R, Johnson R, Vlissides J. *Design Patterns: Elements of Reusable Object Oriented Software.* Addison-Wesley, 1995.

[2] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. A Security Architecture for Computational Grids. *Proc. 5th ACM Conference on Computer and Communications Security Conference* pp. 83-92, 1998.

[3] Czajkowski K, Fitzgerald S, Foster I, Kesselman C. Grid Information Services for Distributed Resource Sharing. *Proc. 10th IEEE Symp. On High Performance Distributed Computing* 2001;

[4] Czajkowski K, Foster I, Karonis N, Kesselman C, Martin S, Smith W, Tuecke S. A Resource Management Architecture for Metacomputing Systems. *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing* 1998;

[5] Allcock W, Bester J, Bresnahan J, Chervenak A, Liming L, Tuecke S. *Grid Forum Working Draft* March 2001; http://www.gridforum.org

[6] Novotny J. The Grid Portal Development Kit *IEEE Concurrency: Practice and Experience* 2002;

[7] Apache Webserver Project
http://www.apache.org

[8] ModSSL
http://www.modssl.org

[9] Java Servlet 2.3 and Java Server Pages 1.2 Specifications
http://java.sun.com/products/servlets

[10] The Jakarta Tomcat Project
http://jakarta.apache.org/tomcat

[11] The Jakarta Jetspeed Project
http://jakarta.apache.org/jetspeed

[12] The Jakarta Turbine Project
http://jakarta.apache.org/turbine

[13] The Jakarta ANT Project
http://jakarta.apache.org/ant

[14] The Jakarta Log4J Project
http://jakarta.apache.org/log4j

[15] The Jakarta JMeter Project
http://jakarta.apache.org/jmeter

[16] The JUnit Project
http://www.junit.org

[17] The Castor Project
http://castor.exolab.org/

[18] Role Based Access Control links
http://csrc.nist.gov/rbac/

[19] Laszewski G., Foster I, Gawor J. CoG Kits: A Bridge Between Commodity Distributed Computing and High-Performance Grids *Proceedings of the ACM Java Grande Conference* 2000;

[20] Novotny J., Tuecke S., Welch V. An Online Credential Repository for the Grid: MyProxy. *Proc. 10th IEEE Symp. On High Performance Distributed Computing* 2001;

[21] Allen G, Daues G, Foster I, Laszewski G, Novotny J, Russell M, Seidel E, Shalf J. The Astrophysics Simulation Collaboratory Portal: A Science Portal Enabling Community Software Development. *Proc. of the 10th IEEE Intl. Symp. on High Perf. Dist. Comp* 2001;

[22] What is a Portlet?
http://www-3.ibm.com/software/webservers/portal/portlet.html

[23] JSR 168: Portlet Specification
http://www.jcp.org/jsr/detail/168.jsp

[24] Hesmer S., Fischer P., Buckner T., Schuster I. *Portlet Development Guide* April 2, 2002;

[25] Hesmer S., Hepper S., Schaeck T. *Portlet API (First Draft)* April 2, 2002;

[26] Kelley, I., Novotny, J., Russell, M., Wehrens, O. Jetspeed Evaluation *WP4 Internal Document* May, 2002;

[27] I. Foster, C. Kesselman, J. Nick, S. Tuecke The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. January, 2002.

[28] Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C. and Nick, J. Grid Service Specification

[29] Web Services Description Language
http://www.w3.org/TR/wsdl

[30] Simple Object Access Protocol
http://www.w3.org/TR/soap12-part1/

[31] Web Services Invocation Framework
http://www.alphaworks.ibm.com/tech/wsif

[32] Universal Description Discovery and Integration
http://www.uddi.org

[33] Web Service Invocation Language
http://www-106.ibm.com/developerworks/webservices/library/ws-wsilspec.html

[34] Apache Axis
http://xml.apache.org/axis

[35] Legion of the Bouncy Castle
http://www.bouncycastle.org

[36] JMyproxy
ftp://george.lbl.gov/pub/globus/jmyproxy.tar.gz

[37] Model View Controller
http://

[38] Apache Xalan-J
http://xml.apache.org/xalan-j/index.html

[39] Jakarta JDO - OJB
http://jakarta.apache.org/ojb/

[40] Castor O/R Mapper
http://castor.exolab.org

[41] Java Server Pages Standard Tag Library
http://java.sun.com/products/jsp/jstl/

[42] Sun JDO Specification
http://access1.sun.com/jdo/

[43] OpenSymphony JSP Caching Tag Library
http://sourceforge.net/projects/opensymphony/

[44] Netscape Directory and LDAP Developer Central.
http://developer.netscape.com/tech/directory/index.html

Software Dependencies
In keeping with the requirements, all software used in implementing the GridLab Portal is open-source and preferably under a BSD-style license rather than the more restrictive GNU Public License (GPL).

- Jakarta Log4J Logging Library[14]
- JUnit Testing Library[16]
- OpenSymphony JSP Caching Tag Library[43]
- Java Server Pages Standard Tag Library[41]
- Bouncy Castle Security Libraries[35]