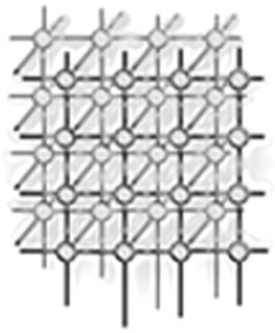
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2004; **16**:503–513 (DOI: 10.1002/cpe.829)

GridSphere: a portal framework for building collaborations



Jason Novotny, Michael Russell and Oliver Wehrens*,†

Max-Planck-Institut fur Gravitationsphysik, Albert-Einstein-Institut, Am Mhlenberg 1, D-14476 Golm, Germany

SUMMARY

Grid-enabled portals are becoming increasingly popular as a platform for providing access to Grid services and resources. Unfortunately, much of the work done in portal development has led to vertically layered solutions that work for a particular project but are difficult to extend or reuse for other projects. The GridSphere portal framework seeks to address these limitations by providing a framework that will offer external developers a model for easily adding new functionality and hence increasing community collaboration. The GridLab portal will serve as an initial prototype to showcase the GridSphere framework and provide access to services being developed within the GridLab project. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: Grid computing; portals; portlets

1. INTRODUCTION

In the last couple of years, Grid-enabled portals have become increasingly popular as a platform to provide application scientists access to Grid services and resources. The difficulty with bridging the gap between currently deployed Grid services and the would-be user community has been largely due to the unfriendly nature of much of the Grid software available today. Web-based portal environments have sought to provide easy-to-use interfaces by relying on technology that reaches out to the lowest common denominator. The holy grail of Grid usability is to provide on-demand services and make access to computational resources as easy as it is to buy books from Amazon.com. In trying to build support for the Grid user community, the authors have developed and describe a new portal framework, *GridSphere*, based on the many previous lessons and best practices learned from past notable Grid

*Correspondence to: Oliver Wehrens, Max-Planck-Institut fur Gravitationsphysik, Albert-Einstein-Institut, Am Mhlenberg 1, D-14476 Golm, Germany.
†E-mail: wehrens@aei.mpg.de

Contract/grant sponsor: European Commission 5th Framework Program; contract/grant number: IST-2001-32133

Received 1 August 2003
Revised 1 September 2003
Accepted 10 September 2003

portal projects such as the Grid Portal Development Kit (GPDK) and the Astrophysics Scientific Collaboratory (ASC) portal.

The GPDK garnered a lot of attention from the Grid community as an attempt to provide a collection of reusable components for accessing Globus-based Grid services. GPDK provided a functional template portal that could be extended and enhanced by users which offered file staging and job submission, and monitoring capabilities. A core part of the design philosophy of GPDK was the separation of logic from presentation, making it easier to develop new functionality that could plug into the existing framework by following a prescribed recipe. From the GPDK project came many lessons about building a framework and developing reusable components. Ultimately, while the GPDK template/demo portal could be enhanced to create a project specific portal, users had to become familiar with the source code in order to add the features they needed. Another major limitation was the lack of any reusability in the presentation layer. Developers would need to handcraft customized presentation pages to re-use the GPDK services provided to create a new portal instance.

The ASC portal project was aimed at providing a Web portal for the computational astrophysics community that would allow them to compile and execute simulation applications on large-scale computational resources. The ASC portal had a similar design to GPDK, but was ultimately specialized in its functionality and services to suit the needs of a particular user community. However, the ASC portal offered a wealth of lessons on trying to create a production portal environment that would attract more application scientists to using the portal instead of the conventional form of logging on to computational resources and running their simulations from the command-line. In both the GPDK and the ASC portal, an emphasis was placed on providing value-added capabilities that would encourage users to perform their work via the portal. As an example, the portal could track and monitor jobs on behalf of users or store input data that had been used for previous simulations. The portal provided a virtual desktop that sought to simplify the day-to-day operations being performed by computational scientists. In doing this, the ASC Portal devoted considerable attention to the needs and usage patterns of its users. Attempts were made to provide easy-to-use, yet flexible Web interfaces that met the demands of the scientists. Another lesson learned in putting the portal into production use is that the portal is only as good as the services that it uses. In deploying a production portal, one of the greatest difficulties is managing the underlying, quickly changing, Grid software libraries that are used by Grid portals and software version compatibilities. As an example, both the GPDK and ASC portals required a user to obtain a valid credential during the login process. If the credential repository was upgraded, users were often unable to login to the portal resulting in user frustration.

2. THE GridLab PROJECT

The GridLab project funded by the EU has recognized the need for higher level computing environments to entice application scientists in using the Grid. The goal of the GridLab project is to enable Grid application development by supporting higher level libraries and services and the deployment of a Grid testbed encompassing resources from ten institutes around Europe. The authors have teamed up to work together with the GridLab project in developing a new portal framework that will support the GridLab community primarily, yet be extensible to support the needs of other Grid and even non-Grid user communities. GridSphere, the Java-based portal framework being developed to support the GridLab portal, will provide scientific researchers with a single point of entry from which

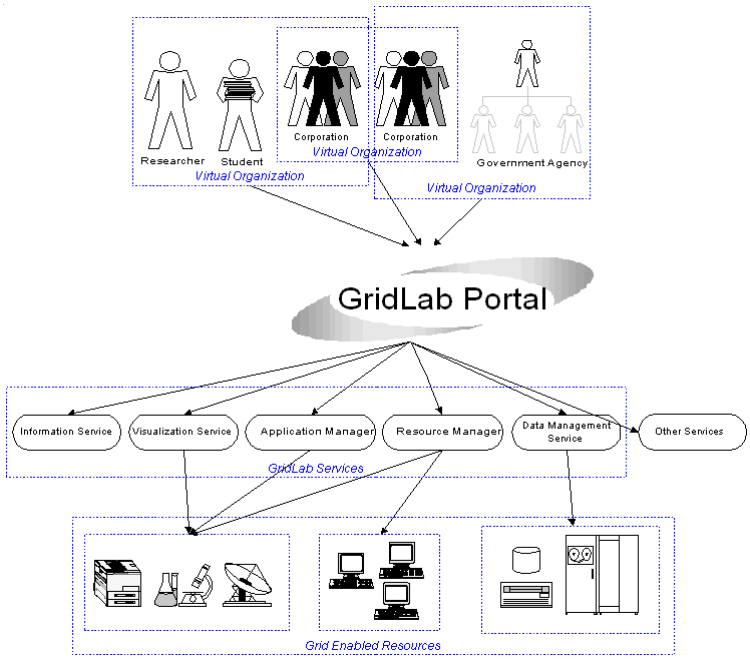


Figure 1. Shows how the portal acts as an intermediary between users of Grid resources and the resources themselves.

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 even a PDA or other mobile device. The hourglass model depicting the role of the GridLab portal as a gateway between end-users and GridLab service providers is shown below.

The design of the GridLab portal, based upon the *GridSphere* portal framework, 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. Indeed GridSphere is designed to be modular enough to also support the needs of a much wider community outside of GridLab. 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 GridSphere architecture outlines both a general portal framework used to assist virtual organizations composed of scientists and project developers, as well as an architecture 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.

3. 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 designed to address this limitation by providing a 'semi-complete' domain-specific application that can be specialized to produce custom applications [1]. 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 [2] 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 *whitebox* framework. Whitebox frameworks rely heavily on object-oriented language features like inheritance and dynamic binding to achieve extensibility. Existing functionality is reused and extended by (1) inheriting from framework base classes and (2) overriding pre-defined hook methods [3]. The difficulty with whitebox 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.

3.1. The Portlet API

Portlets [4] 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 offer 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 model frees the portlet developer from worrying about presentation layout of portlets and Web page composition and to focus solely on providing functionality and a usable interface. Currently, the Portlet API has received widespread adoption in a number of vendor specific application server products including IBM's WebSphere, Oracle's PortalServer, and the Sun iPlanet Server. Because of the importance of portlets to the industrial community, the Portlet API is in the process of being reviewed within the Java Community Process (JCP) and a Java Specification Request (JSR) [5] is currently in the process of being standardized.

Currently, the Jakarta Jetspeed project [6] provides an implementation of the evolving Portlet API and a complete portal framework based upon another Jakarta project, Turbine [7]. Believing that the

Jetspeed/Turbine framework would suit most of the GridLab portal requirements, an evaluation [8] was made. Unfortunately, it was determined that Jetspeed did not provide a suitable development platform in its current state and the dependencies upon existing large over-extended software were too high. In the evaluation, a comparison was made with a commercial portlet implementation offered by IBM's WebSphere Portal Server. While we lacked the time and money to evaluate WebSphere software, the WebSphere Portlet Development Guide [9] offers a very solid description of the Portlet API that comes bundled with WebSphere and appears far more functional than the constantly 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. The GridSphere Portlet API is based loosely on the portlet interfaces provided by the WebSphere Portlet API and is designed to offer only the base functionality required to implement the portlet concept in practice. The GridSphere Portlet API will conform to the standardized API when it is ratified by the JCP.

3.2. Portlet design

In standard Java-based Web application development, *servlets* are developed that function as server-side components responsible for processing browser requests. In the GridSphere Portlet API, portlets inherit much of their functionality from servlets, including existing lifecycle methods such as `init`, `service` and `destroy`, as well as providing additional lifecycle methods that can be implemented by portlet developers.

Initially when the portal is started, the portlets are all initialized and the `service` method is invoked for every client request until the portal is shutdown, at which point the portlets are all `destroy`'ed to allow portlets to perform any shutdown operations.

One of the core concepts associated with portlets is the notion of portlet modes. As described earlier, a portlet is essentially a window, or mini-application running within the portlet container. Portlets may be displayed in the following four modes.

- 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.

The portlet `service` method is responsible for invoking the proper `doView`, `doConfigure`, or `doEdit` method based on the portlet mode dictated by the set of icons on the portlet's title bar. One of the most important design requirements when developing new portal functionality is the separation of presentation from logic which the Portlet API does quite nicely. While the `doXXX` lifecycle methods are responsible for presentation, portlet events are handled by either the `actionPerformed`, `messageReceived`, or `window{Minimized, Maximized, Resized}` methods. It is up to the portlet developer to override these methods to provide functionality to handle the following kinds of portlet events.

- 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.

3.3. Core portlet interfaces

Just as portlets themselves inherit functionality from the Servlet API [10], the Portlet API also specifies decorator classes that wrap most of the standard servlet interfaces to provide a more portlet-centric view of the portal. For instance, the `PortletRequest` object 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). The key additions to the `PortletRequest` interface are the `getUser` method which simply returns the `User` object associated with this request, the `getData` method which returns the user-specific persistent data for the active portlet, and the `getMode` and `getWindow` methods which return the current portlet mode and window state of the active portlet. Similarly, `PortletResponse` and `PortletSession` subclass `HttpServletResponse` and `HttpSession` to provide portlet specific functionality for generating responses and handling sessions. `PortletConfig` and `PortletContext` inherit from `ServletConfig` and `ServletContext` respectively and provide methods for obtaining portlet container information, portlet configuration information, and obtaining instances of portlet services which will be described in more detail later. The PortletSettings object, obtainable from the PortletRequest, provides dynamic portlet customization information and can be modified by authorized users, thus dynamically modifying a portlets configuration information.

3.4. Portlet packaging and deployment

In addition to reusing functionality provided by the Servlet API, portlets are packaged as Web archives (WAR) files, just as servlets are, as defined in the Sun Java 2.3 Specification [10]. By reusing the servlet packaging model, portlets can be developed and packaged cleanly to facilitate easy sharing and deployment of portlets between portal providers. A portlet Web application consists of portlet source code, presentation pages including static HTML, or Java Server Pages (JSPs) and a portlet deployment descriptor which defines a portlet's capabilities and configuration parameters in XML format. The following list itemizes the parameters that a portlet descriptor may define.

- Supported portlet modes.
- Allowed window states.
- Initialization parameters in the form of key value pairs.
- Portlet localization and descriptive information including title.
- Accessibility information including which users can have access to a portlet.

4. 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


```
<service>
    <name>LoginService</name>
    <description>Provides Login Capabilities</description>
    <interface>LoginService</interface>
    <implementation>LoginServiceImpl</implementation>
</service>
```

Figure 2. Example portlet service descriptor.

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.

4.1. 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. In the current model, 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 configuration file is used to provide class information and initialization parameters that are accessed via the `PortletServiceConfig` object. The `PortletServiceFactory` is also responsible for initializing and destroying services. Like the portlet descriptor, the portlet services descriptor is also expressed as an XML file. An example entry is shown below.

The GridSphere portlet 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 of its usage, 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`.

4.2. Core services

The primary core services are the `LoginService`, the `UserManagerService` and the `AccessControlManagerService`. The `LoginService` is used by the GridSphere framework and is not really exposed or useful to portlet developers. The `UserManagerService` provides methods for the creation, querying and deletion of users and groups. The `AccessControlManagerService` provides methods for managing users' roles and assigning/revoking user to/from groups. Access control is not only a core service but used internally by the framework. For instance, when a user service is instantiated the `AccessControlManagerService` is also used to determine which methods the user can successfully invoke in the user service. Our notion of access control is based

upon the Role Based Access Control (RBAC) model [11] in which users can belong to one or more groups. Within a group a user is assigned a role. A generalized RBAC model would allow users to have multiple or even new roles within a group, but for now we have chosen to limit the number of roles to the following.

- Guest—a portal user that has not logged into the portal.
- User—a standard user can access portlets but may have restricted access to particular functionality.
- Administrator—a user that is a manager of a particular group. Administrators can grant users access to a group.
- Super—the super user can do everything from approving new user account requests to adding/deleting groups.

5. GridSphere CORE PORTLETS

The GridSphere framework provides a core set of portlets that offer the basic functionality required for the portal to be usable. Essentially the only portlet required at initialization time is the Portlet Manager Portlet. The manager portlet allows packaged portlets to be deployed, removed and installed 'on the fly' from the portal itself by authorized users.

The following list itemizes the core set of portlets provided by default.

- Login—allows users to login based on a name and password.
- Logout—logs a user out of the portal.
- Account request—provides interface for a new portal user to request an account and optionally choose groups to join.
- Account management—provides the super user and admin users the ability to assign/revoke users' roles.
- User management—provides the super user the ability to approve/deny account requests, and admin users the ability to approve/deny group requests.
- Portlet subscription—provides the ability for users to add and remove portlets from their workspace.

5.1. OGSA and Grid services

While Grid services are not necessary for the proper functioning of the framework or the core portlets, many advanced portlets that wish to offer some Grid functionality will make use of Grid services as represented under the outlined services framework. These services will not only provide access to existing Grid middleware such as Globus gatekeepers, GridFTP services and MDS information services, but will also provide access to the GridLab developed services which are designed to be higher level than the existing Grid fabric layer. The GridLab services include resource brokers, monitoring and data management services to name a few. These services will use the respective GridLab client APIs as well as the Java CoG [12] and the Open Grid Services Architecture (OGSA) [13] to provide access to Globus-based Grid services.

Figure 3. The GridSphere portal.

Integration of GridSphere and OGSA has just begun and already portlets have been developed that can use OGSA clients to connect to OGSA services hosted elsewhere, as described in the IBM developerWorks articles [14]. Even more interesting for future development is the planned integration of the GridSphere portal framework and OGSA services. Under this model, the portal will also act as a container for OGSA services that can be accessed either via the portal itself or other clients. By building off the notion of portlet services discussed earlier we hope to make these portlet services OGSA compliant by providing a wrapper around the portlet services to make them accessible via OGSA clients. For instance, the current `LoginService` which is a portlet service could benefit from being exposed as an OGSA service to allow portal interactivity from other clients beyond a user's Web browser, such as a PDA or other mobile device.

6. STATUS AND FUTURE DIRECTIONS

A stable release of the GridSphere portlet container is now available and future work will focus on developing more Grid-enabled portlets using OGSA/Globus and other underlying middleware. We are actively collaborating and seeking collaborations to enhance the overall usefulness of GridSphere to the general portal community. As of this writing, the JSR 168 Portlet Specification has just been announced and we are hoping to be compliant with the 'blessed' Portlet API as soon as possible. Another area of future direction is to enhance the GridSphere portlet container to develop a general 'content management system' (CMS) to allow portal users to dynamically create and update personalized content. Other portlets that we wish to develop to increase the overall usability and end-user experience are game portlets and a variety of chat, SMS, e-mail portlets to promote collaboration and enthusiasm among portal users.

7. CONCLUSION

The GridSphere Portal Framework is evolving to support a wide variety of applications that can be easily plugged into the portal by adopting the Portlet API. The GridSphere framework offers many value-added features such as a services framework and core portlets that make it easy for new portlet developers to immediately provide portlet interfaces that can be easily deployed to the portal. Core portlets are provided that manage users, access control and portlet subscription. In addition, manager portlets provide the capabilities to dynamically install and remove portlets without having to shutdown the application server.

8. AVAILABILITY

The GridSphere portal framework has been released under an open-source software license and is available from the project Web site at http://www.gridsphere.org.

ACKNOWLEDGEMENTS

The authors wish to thank all the members of the GridLab project for their assistance and discussions that helped make the GridSphere framework more usable. We wish to acknowledge the support of the European Commission 5th Framework program (grant IST-2001-32133), the primary funder of the GridLab project, without which none of this work would be possible.

REFERENCES

1. Johnson R, Foote B. Designing reusable classes. *Journal of Object-Oriented Programming* 1988; **1**(5):22–35.
2. Gamma E, Helm R, Johnson R, Vlissides J. *Design Patterns: Elements of Reusable Object Oriented Software*. Addison-Wesley: Reading, MA, 1995.
3. Schmidt DC. Applying design patterns and frameworks to develop object-oriented communication software. *Handbook of Programming Languages*, vol. I, Salus P (ed.). MacMillan Computer Publishing: Reading, MA, 1997.
4. What is a Portlet? http://www-3.ibm.com/software/webservers/portal/portlet.html [11 December 2003].
5. JSR 168: Portlet Specification. http://www.jcp.org/jsr/detail/168.jsp [11 December 2003].
6. The Jakarta Jetspeed Project. http://jakarta.apache.org/jetspeed [11 December 2003].
7. The Jakarta Turbine Project. http://jakarta.apache.org/turbine [11 December 2003].
8. Kelley I, Novotny J, Russell M, Wehrens O. Jetspeed evaluation. *WP4 Internal Document*, May 2002.
9. Hesmer S, Fischer P, Buckner T, Schuster I. Portlet development guide, 2 April 2002.
10. Java Servlet 2.3 and Java Server Pages 1.2 Specifications. http://java.sun.com/products/servlets [11 December 2003].
11. Role Based Access Control links. http://csrc.nist.gov/rbac/ [11 December 2003].
12. 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.
13. Foster I, Kesselman C, Nick J, Tuecke S. The physiology of the Grid: An open Grid services architecture for distributed systems integration, January 2002. http://www.globus.org/research/papers/ogsa.pdf [11 December 2003].
14. Zhang L-J, Chung J-Y, Zhou Q. Developing Grid computing applications, Part 1, 1 October 2002. http://www-106.ibm.com/developerworks/grid/library/gr-grid1 [11 December 2003].
15. Foster I, Kesselman C, Tsudik G, Tuecke S. A security architecture for computational grids. *Proceedings of the 5th ACM Conference on Computer and Communications Security*, 1998; 83–92.
16. Czajkowski K, Fitzgerald S, Foster I, Kesselman C. Grid information services for distributed resource sharing. *Proceedings 10th IEEE Symposium on High Performance Distributed Computing*, 2001.

17. Czajkowski K, Foster I, Karonis N, Kesselman C, Martin S, Smith W, Tuecke S. A resource management architecture for metacomputing systems. *Proceedings of the IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.
18. Allcock W, Bester J, Bresnahan J, Chervenak A, Liming L, Tuecke S. Grid Forum Working Draft. http://www.gridforum.org [11 December 2003].
19. Novotny J. The Grid Portal Development Kit. *Concurrency and Computation Practice and Experience* 2002; **14**(13–15): 1129–1144.
20. The Jakarta Tomcat Project. http://jakarta.apache.org/tomcat [11 December 2003].
21. Novotny J, Tuecke S, Welch V. An online credential repository for the Grid: MyProxy. *Proceedings 10th IEEE Symposium on High Performance Distributed Computing*, 2001.
22. 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. *Proceedings 10th IEEE International Symposium on High Performance Distributed Computing*, 2001.
23. Hesmer S, Hepper S, Schaeck T. *Portlet API* (first draft), 2 April 2002. http://cus.apache.org/viewcvs/jakarta-jetspeed/proposals/portletAPI/PortletAPIDraft.pdf [11 December 2003].
24. Tuecke S, Czajkowski K, Foster I, Frey J, Graham S, Kesselman C, Nick J. Grid Service Specification, February 2002. http://citeseer.nj.nec.com/article/tuecke02grid.html [11 December 2003].

