GridSphere: An Advanced Portal Framework

Jason Novotny, Michael Russell, Oliver Wehrens

Abstract

In traditional web application development, very few libraries exist to make portal development easy. In general, many homegrown and vertical solutions exist and very little code is shared or reused. Finally with the emergence of some critical new web technologies, web application development is focusing more on reusable solutions and software. The GridSphere portal framework provides a standards based portal for the easy development of modular web components, called *portlets*. Portlets are defined by a standard API and provide a model for developing new portal components that can be shared and exchanged by various portlet containers. GridSphere provides both a portlet container, a collection of core portlets and an advanced user interface library that makes developing new portlets easier for application developers. This paper discusses briefly the the GridSphere portal architecture including the layout engine, support for two portlet API implementations and the portlet services model.

1 Introduction

Although many definitions exist for the term *web portal*, most definitions classify a portal as a gateway web site that offers a set of services. The services provided may range from weather information to stock quotes to the ability to search for other sites and many other features. Web portals may also offer personalization and customization features that provide users with a unique, individualized web experience. Examples of generalized portals include Excite®, Yahoo®, America Online®, or Microsoft Network®. Niche portals tend to offer specialized content suitable for a particluar audience. CNN.com®is an example of a news portal or the increasingly popular Friendster®portal supports building personal connections and communities of people with common interests.

In general, many different technologies may be used to construct a portal. A portal can be delivered using anything from a loosely couple collection of static web pages to using a full-blown *application server*. An application server is viewed as a part of a three-tier application consisting of the following components as shown in the diagram:

1. The first tier is a user's web browser, generally on their desktop but may also be a mobile device or PDA.
2. The second tier web performs business logic and generally combines or work with a web server that can dispatch requests via HTTP from the first-tier web browser.
3. The third tier, or back-end, are resources such as databases, information servers or compute resources.

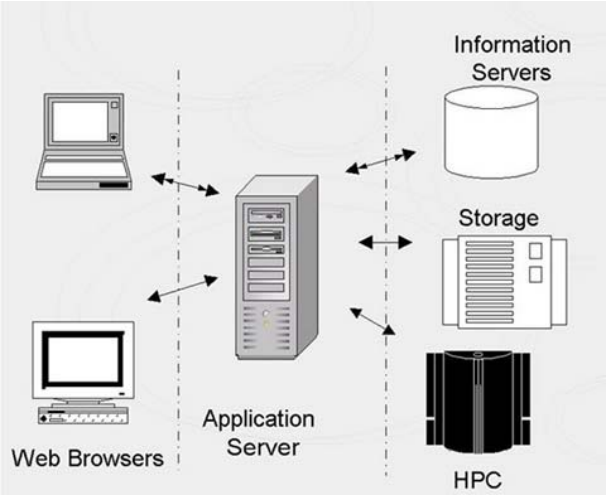

Figure 1: Classic 3 tier application architecture

A wide variety of software packages exist that perform the duties of the application server. All in one solutions such as IBM's WebSphere®, Oracle i9AS Server®, Sun Application Server®or BEA WebLogic®provide monolithic solutions that may be relatively easy to set up and configure "out of the box". Many open source solutions are based on using the popular Apache web server, the Jakarta

Proceedings of the 30th EUROMICRO Conference (EUROMICRO'04)
1089-6503/04 $ 20.00 IEEE

Tomcat[10] Servlet container, in combination with various *web application frameworks*. The vast majority of application server solutions are currently Java™based over any other implementation technology for the reason that server side Java has gained tremendous popularity as a language that provides greater support for the development of component based architectures and can more readily separate presentation from logic. According to a recent Jupiter report[1] of the most widely used application servers, over 95% of the portal market is dominated by Java based technologies.

While a portal can be constructed simply by using a web server and a collection of static HTML pages, most portals capable of any real functionality especially personalization, security and transactions, require an architecture for the development of components that can add new functionality and provide reusability. Web application frameworks have emerged as a collection of tools and best practices to make the development of new components easier. With the last three years the number of application frameworks has ballooned. Currently a list of the most popular web frameworks includes Struts[2], Tapestry[3], WebWork[4], Turbine[12], Barracuda[6], Expresso[7], Spring[8] and numerous others. To the first approximation all frameworks have one goal in common which is to provide a mechanism for the separation of logic and presentation. Quite clearly this separation provides far greater flexibility in the features and functionality that can be provided by the application server and how it gets displayed and possibly customized to end users. In general, all frameworks make use of a very common and popular *design pattern* for this separation called, the Model View Controller (MVC) pattern. In the MVC approach, the model is comprised of the data that is to be displayed and is generally encapsulated in terms of some generic abstract data type. The view refers to the component responsible for the display of the model and can be specially tailored or customized. The controller has the responsibility of dispatching requests from the user to the appropriate set of components that construct the model and presentation. Each of these three aspects represent orthogonal functionality that can be varied independently resulting in a powerful design paradigm. As will be discussed in the next sections, the portlet model also encapsulates MVC design and can be complemented with existing web application frameworks. In addition the portlet model has been ratified through a formal specification unlike existing web application frameworks giving it a broader appeal.

2 The Portlet Concept

Last October 2003, the portal market was shaken up when a new API, known as Java Specification Request (JSR) 168[14] Portlet API, was finally approved by most of the major portal application server vendors including IBM®, Sun®, Oracle®, Plumtree®, BEA®, Vignette®, SAP®, and ATG®under the Java Community Process (JCP). The Portlet API represents an attempt at creating a well defined interface for the development of web components that can be shared and exchanged among portal vendors to provide enhanced functionality. For years many vendors had been developing their own proprietary models for the creation of new components that would work with their specific portals but could not be reused in another vendor's portal. The Portlet API represents a convergence of views that just as operating system lock-in prevents the sharing of applications between say Microsoft™, Apple™and Linux, various third party software providers needed to tailor their specific set of web functionality for individual portals resulting in far greater development effort, lack of any real reusability and overall maintenance headaches. It is a fairly good guess that while portal vendors will continue to vie for customers by providing overall greater ease of use, configuration and bundled software with their portal offerings, the shift has now gone to producing portlet applications that provide "add-on" value for specialized customer bases. Independent software vendors (ISV's) can now concentrate on delivering functionality in particular focused areas and program to a common API.

Portlets[13] define web application components with a well defined set of lifecycle methods much like the Java Servlet API[9] as well as providing a visual interface 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. Visually portlets appear as "mini-windows" within a portal page and have a title bar providing options for setting the portlet mode and window state as shown in the screenshot of the GridSphere portal

below:

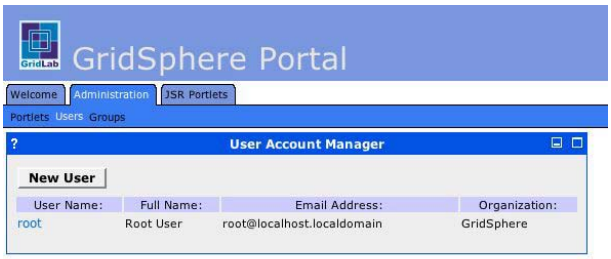

Figure 2: GridSphere user manager portlet

Just as a servlet container provides a runtime environment for the loading and management of Java servlets, a portlet container performs the necessary initialization and management of portlets. The portlet container acts as an intelligent dispatcher to forward client browser requests to the appropriate portlet during a request response cycle. It is the responsibility of the portal and not the portlet container to appropriately render the markup of portlets onto a single portal page. The next sections will discuss the architecture of the GridSphere portal consisting primarily of the portlet container and the portal layout engine.

3 GridSphere Framework Overview

GridSphere provides a portlet implementation, a portlet container and a collection of core services and portlets as shown in the diagram:

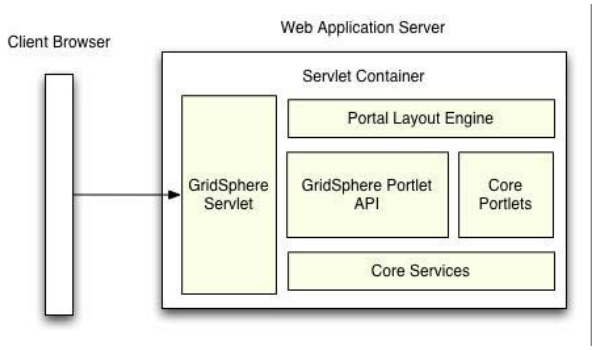

Figure 3: GridSphere Portal Architecture

The architecture diagram shows the primary components that combined with the servlet container comprise the web application server. A request to the portal triggered by a user's web browser invokes the GridSphere servlet which acts as a controlling dispatcher to the layout engine responsible for rendering suitable output to the user's browser. Both the GridSphere servlet and layout engine make use of core services, including the portlet registry, to invoke the appropriate set of portlets in a users portal page. The following subsections discuss each of these components of the architecture in greater detail.

3.1 The Layout Engine

Layouts in the portal are defined as XML descriptor files as shown in the following example snippet which displays the portal page banner:

```
<page-layout theme="xp" title="GridSphere Portal">

<portlet-header>
 <table-layout style="header">
  <row-layout>
   <column-layout width="70%">
    <portlet-content include="/html/pagehead.html"/>
   </column-layout>
   <column-layout width="15%">
    <portlet-frame transparent="true" label="locale">
     <portlet-class>LocalePortlet.1</portlet-class>
    </portlet-frame>
   </column-layout>
   <column-layout width="15%">
    <portlet-frame transparent="true" label="logout">
     <portlet-class>LogoutPortlet.1</portlet-class>
    </portlet-frame>
   </column-layout>
  </row-layout>
 </table-layout>
</portlet-header>
...
```

Figure 4: Layout descriptor

The nested structure of the layout descriptor provides a very handy approach to modeling a templated layout that can be easily modified that acts as an abstraction layer above the underlying display technology, generally HTML. The *Composite* design pattern is employed for the representation of layout components within a nested tree-like structure. Each layout component defined in GridSphere e.g. `PortletFrame`, `PortletTabbedPane`, `PortletTab`, `PortetContent`, etc itself adheres to a `PortletComponent` interface which defines a set of lifecycle methods that closely maps to the lifecycle methods of portlets. The following UML sequence diagram shows how the actual portal page rendering

is accomplished:

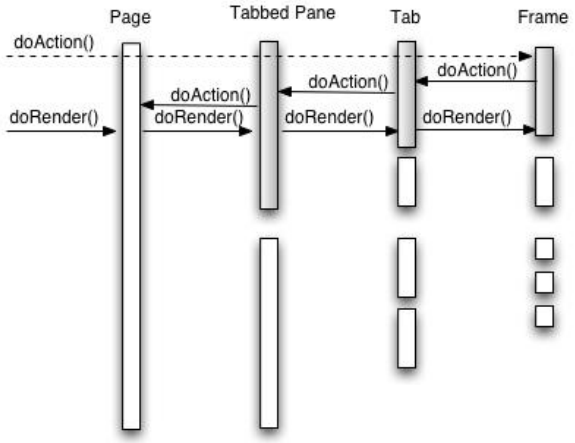

Figure 5: Layout Engine Sequence

The diagram is simplified by showing a page consisting of a tabbed pane which itself contains a few tabs with frames within the tabs. The frames represent the visual appearance of the portlet and are responsible for obtaining a portlet's markup for display in the portal page. In typical GridSphere usage, the portal page uses a double tabbed pane approach, where each tab itself contains an additional tabbed pane with tabs and frames for greater navigability like that supported by many portal sites including the Amazon®, and Apple™websites.

As shown, an event caused by clicking on a layout component performs an action in that component which is then "bubbled" upward to to the root of the layout tree. Afterwards, the component tree is rendered appropriately. Notice that not every component in the tree is rendered. For instance only the visible tab will continue to propagate render events to its children components.

The Layout Engine offers great flexibility and is entirely customizable. New components can simply be added by creating an XML descriptor element and a corresponding component class that implements the `PortletComponent` interface. In fact we have seen our model extended by other projects wishing to build specialized visual components for rendering iFrames, XHTML or that provide additional layouts.

3.2 The GridSphere Portlet Model

When GridSphere was first under development, the ratified JSR 168 Portlet API did not exist and so we had two options. One, we could have chosen to build off of the open source Jakarta Jetspeed project[11]. In fact, at the time, Jetspeed was the most highly evolved open source portal offering a first attempt at developing a Portlet API. However, after further evaluation[17], it was determined that Jetspeed 1 had many dependencies on other large stacks of software such as the Jakarta Turbine project that were quickly changing. In addition, the Portlet API offered was not quite robust enough to support the easy development and packaging of external, or "third-party", portlets. However, upon looking at the IBM WebSphere®4.1 Portlet API we found a richer, better documented API that itself had been based upon Jetspeed in earlier versions of WebSphere®. Believing the finalized API to be somewhat similar to the WebSphere®API, we chose to base our implementation using the IBM WebSphere®4.1 Javadocs[16] and Developer's Guide[15] as a starting point. For the first year of development, we implemented the WebSphere Portlet API and based our core portlets provided by GridSphere on this model. One of the benefits of the IBM®Portlet API was a clearer packaging model that allowed for easy integration of third-party portlets as Web Archive, or WAR files, in the same way the Java Servlet API specifies the packaging of web applications as WAR files.

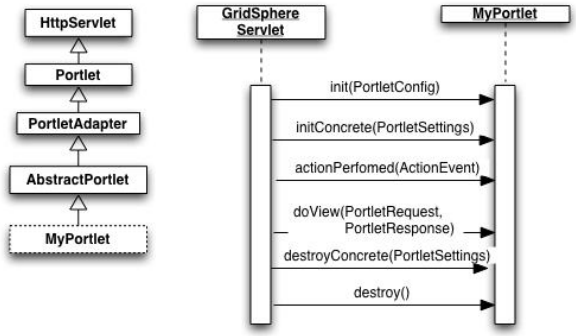

Figure 6: IBM WebSphere portlet hierarchy and sequence

In the WebSphere®portlet model, there are no interfaces defining a portlet. Instead a portlet is represented as a hierarchy of abstract and base classes as shown on the left. A portlet developer would subclass from the `AbstractPortlet` base class to

develop a customized portlet, `MyPortlet` in the diagram.

The WebSphere®portlet model makes a distinction between *Application Portlets* and *Concrete Portlets* where a concrete portlet is a parametrized instance of an application portlet. For instance a stock quote portlet can have multiple concrete instances that are parametrized with different stock quote services from which the quotes are obtained. As shown in the sequence diagram, an application portlet and its corresponding concrete portlets are initialized during `init()` and `initConcrete()` method invocations respectively. Similarly, portlets are destroyed via the `destroyConcrete()` and `destroy()` methods. These methods generally occur only once during a portlets lifecycle, while the `actionPerformed()` and `doView` methods may occur during every client request potentially. The `actionPerfomed` is called only when an action occurs in the portlets, usually as a form submission, button click or portlet hyperlink being invoked. For simplicity only the `doView()` method is shown in the sequence, although portlets in allow for three additional *portlet modes*: `Edit`, `Configure` and `Help`. A portlet developer may implement the `doEdit()`, `doConfigure()` or `doHelp()` methods to support these modes when the mode icon is selected from the portlet title bar. The appropriate render method is invoked for every portlet that is displayed on the page, although the portal may choose to cache rendered output for redisplay.

Just as the portal layout is defined by a layout descriptor, a portlet descriptor file, `portlet.xml`, defines the portlet configuration information and provides the portlet container the necessary details for instantiation of the portlets. Both the `PortletConfig` and `PortletSettings` objects used during portlet initialization contain configuration information obtained from the portlet deployment descriptor.

The JSR 168 Portlet API is relatively similar to the WebSphere®Portlet API. However, in JSR 168, a portlet is defined by a `Portlet` interface which is implemented by the `GenericPortlet` base class as shown in the diagram:

A portlet developer would sublclass from `GenericPortlet` as shown to produce `MyPortlet` as an example. The lifecycle interfaces provided in the JSR 168 Portlet API are similar to the WebSphere®API, however there is no notion of concrete and application portlets. Only one portlet exists and is initialized. Instead of `ActionPerformed()`, the action method that is invoked when a portlet action occurs is replaced by the `processAction()` method. In practice, it is not too difficult to refactor WebSphere®portlets to be JSR compliant, although the WebShere®API provides a few additional interfaces for portlet messaging and better localization support.

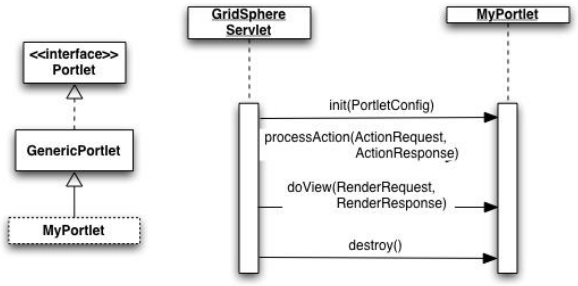

Figure 7: JSR 168 portlet hierarchy and sequence

After JSR 168 had been finalized and ratified we faced the design decision of how to support the JSR portlet API and handle the refactoring of existing portlets that would be necessary. Upon further reflection, a plan was devised that would continue to support the existing WebSphere®API and offer support for JSR 168. The merits of doing this are the following:

1. Allows developers that have already developed WebSphere®portlets to keep existing portlets without having to refactor to JSR portlet API.
2. Continues to support WebSphere®API compliance allowing WebSphere®users to migrate to GridSphere as an open source portal solution.
3. Does not force existing GridSphere framework code to be refactored which would destabilize GridSphere for some period of time
4. Modularity allows JSR portlet API implementation to be developed on top of existing functionality and requires only minimal modifications to the existing framework.

The following architecture diagram shows how support for both WebSphere®and JSR 168 portlet API's is provided. The portlet registry acts as a repository for portlets that is used by the

GridSphere servlet controller and the layout engine. In the case of a WebSphere®portlet application, the portlets are registered in the portlet registry upon startup. To the GridSphere framework, a JSR portlet application looks just like a WebSphere®application with one special portlet, the `PortletServlet` . The `PortletServlet` performs the task of loading and registering all JSR portlets in the application, such that the portlet registry contains both WebSphere®and JSR 168 portlets that can be invoked by the GridSphere servlet and the layout engine. In addition, the GridSphere portlet container includes portlet API implementations of both the WebSphere®and JSR 168 API's, such that portlet development for both models is supported.

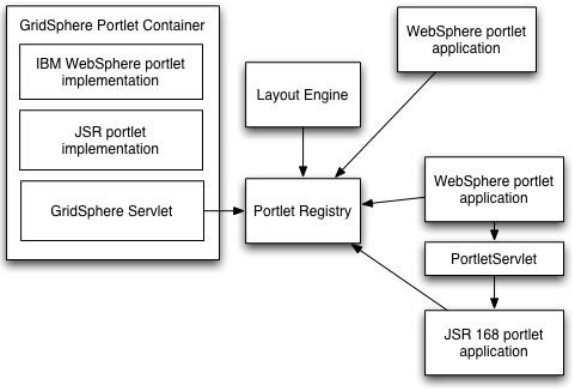

Figure 8: GridSphere support for WebSphere and JSR 168 portlet API's

The following sequence diagram 9 shows how the container is responsible for invoking JSR portlet lifecycle methods via the special `PortletServlet` gateway.

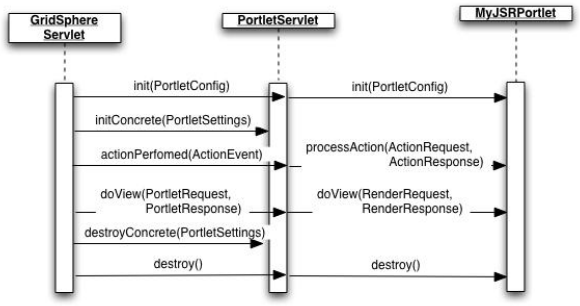

Figure 9: Sequence diagram capturing both portlet API's

Generally the `PortletServlet` is responsible for translating WebSphere portlet lifecycle methods into JSR portlet lifecycle methods. Fortunately, the WebSphere model contains a superset of the JSR portlet methods that translate quite easily. In the case of an `initConcrete()` or `destroyConcrete()` method call, no JSR portlet invocation is performed since that concept does not exist. The `PortletServlet` also does the work of translating the WebSphere portlet API core objects such as `PortletRequest` or `PortletResponse` into the `RenderRequest` or `RenderResponse` objects as required by the JSR 168 portlet model.

3.3 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. Portlet services provide an encapsulation of reusable business logic that may be reused by one or more portlets. In GridSphere, services are used to manage everything from layout preferences, user profiles, user access control as well as the portlet registry discussed previously.

3.3.1 Portlet Services API

The IBM WebSphere®API provides a set of interfaces for defining portlet services and allows developers to get portlet service instances via the `PortletContext` object. Although the JSR portet API makes no mention of portlet services, we can provide access to services by using our own "enhanced" portlet which subclasses from the base `GenericPortlet` . 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 is accessed via the `PortletServiceConfig` object. The `PortletServiceFactory` is also responsible for initializing and destroying services. A portlet service UML class diagram is shown in figure 10:

Like the portlet or layout descriptor, the portlet

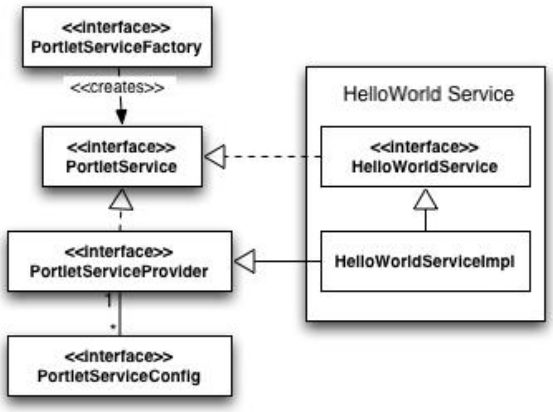

Figure 10: Portlet services UML class diagram

services descriptor is also expressed as an XML file. An example entry is shown in figure 11:

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

Figure 11: Portlet services descriptor

4 GridSphere Core Portlets

The GridSphere framework provides a core set of portlets that offer the basic functionality required for the portal to be usable. The following list itemizes the core set of portlets provided:

- Login - Allows users to login based on a name and password
- Logout - Logs a user out of the portal
- Locale Selection - A user can select from English, French, German, Italian, Hungarian, Czech and Polish
- 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
- User Profile - Provides the ability for users to edit personal information including name, email address and select portlet applications to join
- Layout Configuration - Users can configure their layout including placement of tabs and portlets within tabs
- Portlet Subscription - Provides the ability for users to add and remove portlets from their workspace
- Local File Manager - Provides a virtual filesystem allowing users to edit, upload and download files to the portal
- Notepad - Users can create, edit, delete and search notes
- Text Messaging - Users can text message other users using instant messenger

5 Conclusion

The Portlet API has garnered a lot of support within the web application server community and looks to be a viable model for the sharing and reuse of portal components. The GridSphere portal framework provides both an implementation of the WebSphere®and JSR 168 Portlet APIs as well as a full-fledged portal and portlet container. The collection of core portlets bundled with GridSphere offer the minimum functionality required of a portal including user and group management, user customization and user profile management and more.

6 Acknowledgments

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 thank the support of the European Commission 5th Framework program, (grant IST-2001-32133) the primary funder of the GridLab project and without which none of this work would be possible.

7 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`

References

[1] Jupiter Application Server Report
http://www.nwfusion.com/
news/2003/0214studyoracl.html (Feb 22)

[2] Jakarta Struts framework
http://jakarta.apache.org/struts/index.html
(Feb. 22)

[3] Jakarta Tapestry framework
http://jakarta.apache.org/tapestry/index.html
(Feb. 22)

[4] WebWork
http://www.opensymphony.com/webwork/
(Feb. 22)

[5] Jakarta Turbine framework
http://jakarta.apache.org/turbine/ (Feb. 22)

[6] Barracuda
http://barracudamvc.org/Barracuda/index.html
(Feb. 22)

[7] Expresso framework
http://www.jcorporate.com (Feb 22)

[8] Spring framework
http://www.springframework.org (Feb. 22)

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

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

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

[16] WebSphere®Portlet API Javadocs
http://www-106.ibm.com/developerworks/
websphere/zones/portal/portlet/4.2api/WPS/

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