Universität Leipzig
Institut für Informatik
Lehrstuhl für Angewandte Telematik / e-Business
Prof. Dr. Volker Gruhn

Proseminar „User Interface Engineering für mobile und web-basierte Anwendungen“

Java-Portaltechnologie

Portale mit dem Java-Portlet-Standard JSR168, WSRP und Jetspeed 2

Autor: Thorsten Berger mail@thorsten-berger.net

Studiengang: Informatik (8. Semester)

Betreuer: Dipl. Inf. Matthias Book

Eingereicht am:

Inhaltsverzeichnis

1 Einleitung

Portale sind zu einem wichtigen Bestandteil des World Wide Web geworden. Sie bieten Benutzern durch einen zentralen Einstiegspunkt und eine einheitliche Oberfläche einfachen Zugriff auf verschiedene Anwendungen und Informationen.

Viele ehemals statische Webseiten entwickelten sich zu komplexen Systemen, die ein breites Gebiet an heterogenen Informationen und Applikationen zur Verfügung stellen wollen. Eine Informationsflut, die manchmal für den Benutzer kaum in den Griff zu bekommen ist. Portale können hier Informationen sinnvoll strukturieren bzw. selektieren und so dem Benutzer genau das bereitstellen, was er auch wirklich benötigt. Historisch gesehen waren Suchmaschinen-Betreiber und Online-Dienste die ersten, deren Webseiten man als Portale bezeichnen konnte.

Heute verwenden viele Seiten diese Bezeichnung ohne ihr wirklich gerecht zu werden. Im Folgenden soll daher der Portal-Begriff definiert sowie ein Überblick über grundlegende Konzepte und Zielstellungen von Portalen gegeben werden. Detailliert wird auf den Java-Portlet-Standard JSR 168 als eines der wichtigsten herstellerübergreifenden API im Abschnitt 2 sowie auf das darauf basierende Open-Source-Portal Jetspeed2 im Abschnitt 4 eingegangen. Weiterhin bietet Abschnitt 3 eine Einführung in die WSRP-Spezifikation, welche parallel zum JSR168 entwickelt worden ist.

1.1 Portal-Begriff

Für den Anwender fassen Portale Inhalte und Anwendungen verschiedener Quellen unter einer einheitlichen Web-Oberfläche zusammen. Unter inhaltlichen Gesichtspunkten betrachtet haben sich die folgenden drei Bezeichnungen herausgebildet. So umfassen vertikale Portale Inhalte zu mehreren Themengebieten und versuchen durch ein reichhaltiges Angebot eine Einstiegsfunktion in das Internet bereitzustellen, horizontale Portale konzentrieren sich hingegen auf ein bestimmtes Thema und Unternehmensportale sollen schließlich Workflows optimieren sowie Mitarbeitern interne Anwendungen zentral zur Verfügung stellen [TH03].

Portale bestehen aus vielen einzelnen Anwendungen oder Informationseinheiten. Diese können beliebiger Komplexität sein und werden häufig mit Begriffen wie *Channels* (was den inhaltlichen Aspekt hervorheben soll), *Portlets* (Java) bzw. *WebParts* (Microsoft) bezeichnet.

Portale gewährleisten Anwendern Zugriff auf Ressourcen mit unterschiedlichen Rechten, was durch entsprechende (rollenbasierte) Sicherheitsmodelle realisiert werden muss. Die Authentifizierung erfolgt zentral am Portal und nicht mehrmals an jeder integrierten Anwendung einzeln („Single Sign On“). Weiterhin sollte es für Benutzer möglich sein, die Umgebung an eigene Bedürfnisse anpassen zu können („Customization“). Gute Beispiel sind hier myYahoo[1] oder myNetscape[2], bei

[1] http://my.yahoo.com
[2] http://my.netscape.com

denen man seine Startseite mit Informationseinheiten selbst zusammenstellen kann, siehe Abbildung 1.

Einer der häufigsten Einsatzmöglichkeiten für Portale ist die Bereitstellung einer Schnittstelle zu Content-Management-Systemen (CMS). Benutzer greifen über das Portal auf Inhalte des CMS zu, während wiederum andere Inhalte erstellen und modifizieren. Viele Hersteller bieten bereits integrierte CMS-Lösungen für ihr Portal. Abschnitt 4.2.2. bietet einen Überblick über das CMS Apache Graffito und seine Integration in Jetspeed-2.

Web-Services bieten neuartige Kooperationsmöglichkeiten von lose gekoppelten Systemen über das Internet auf einer B2B-Basis. Das steigende Wachstum des Internets bildete die Grundlage für immer neue Kombinationsmöglichkeiten von Diensten aller Art. Portale bieten mit der WSRP-Spezifikation [WSRP04] eine Architektur, um entfernte Portlets über Web-Service-Schnittstellen auf einfache Art und Weise Endbenutzern zur Verfügung zu stellen. Abschnitt 3 gibt eine Einführung in den Standard.

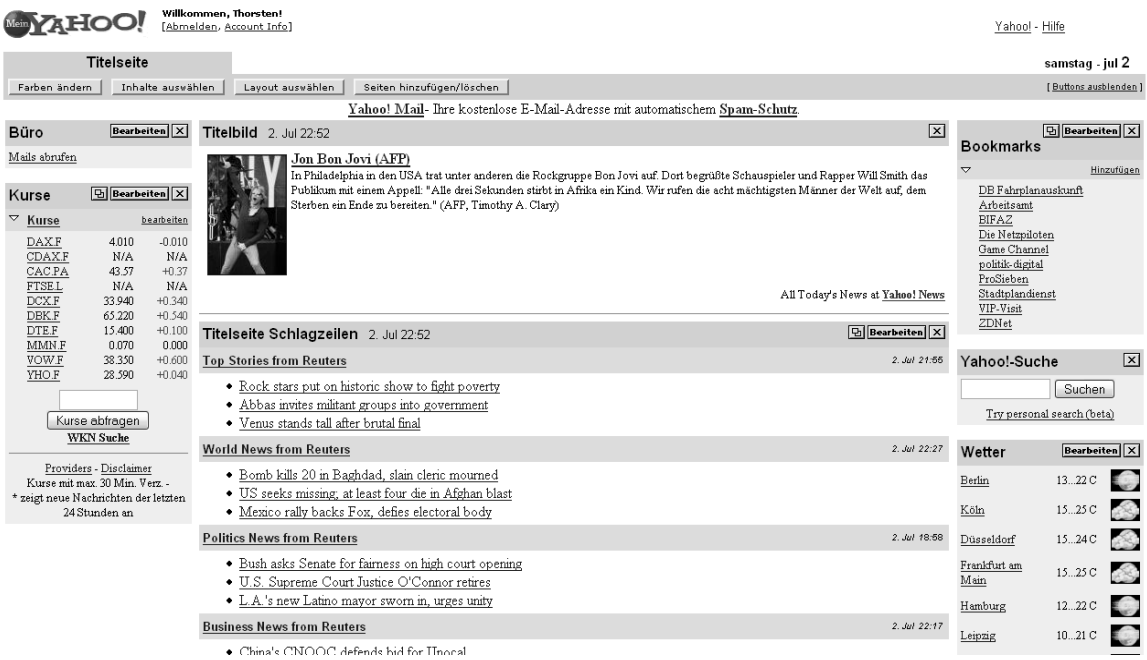

Abbildung 1: myYahoo

1.2 Grundlegende Konzepte und Funktionen

Der folgende Versuch einer Definition soll technische Aspekte von Portal-Software charakterisieren und weniger auf inhaltliche Gesichtspunkte eingehen. Zusammenfassend lässt sich mit [PLT03] und Abschnitt 1.1 festhalten:

Ein (Web-)Portal ist eine Webapplikation, die mindestens folgende Konzepte/Funktionen umsetzt bzw. implementiert:

- „Content Aggregation“: Integration/Zusammenfassen von (heterogenen) Inhalten unter einer Präsentations-Oberfläche
- Security, „Single Sign On“: (rollenbasiertes) Sicherheitsmodell sowie zentrale Authentifizierung
- „Customization“: Anpassung der Oberfläche an persönliche Bedürfnisse

Neben zusätzlichen Administrationsfunktionen können Portale nach [WEG02] auch noch folgende Funktionen bereitstellen:

- „Content Syndication“: Benutzung von Inhalten fremder Informationssysteme, bspw. RSS-Feeds, NewsML oder direkt HTML
- „Multidevice Support“: Aufbereitung von Inhalten für unterschiedliche Ausgabemedien/Markup-Sprachen

1.3 Portal-Software

Es existieren zahlreiche kommerzielle sowie Open-Source-Portal-Lösungen, u.a.:

- IBM Websphere Portal (kommerziell, JSR 168)
- Sun One Portal (kommerziell, JSR 168)
- Apache Jetspeed 2 (Open Source, JSR 168)
- JBOSS Portal (Open Source, JSR 168)
- Microsoft Sharepoint (kommerziell, WebParts)
- Apache Cocoon (Open Source)

Dabei sind insbesondere die Portale, welche den JSR 168 unterstützen interessant, da Portlets zwischen diesen problemlos ausgetauscht werden können, insofern sie nicht spezifische Erweiterungen nutzen.

Bei den beiden anderen Portalen stellt Sharepoint von Microsoft ein ausgewachsenes Dokumentenmanagement-System zur Verfügung. Apache Cocoon hingegen kann man eher als Web-Publishing-Framework mit Portal-Komponente bezeichnen, das aber auch in andere Portale, insbesondere Jetspeed-2, integriert werden kann.

2 Java-Portlet-Standard (JSR 168)

Der *Java Specification Request (JSR) 168* definiert eine herstellerübergreifende API zur Entwicklung von Portlets. Die Spezifikation wurde im August 2003 vom *Java Community Process (JCP)* als Erweiterung der J2EE 1.4 veröffentlicht. An seiner Entwicklung waren hauptsächlich die Java-Größen Apache Software Foundation, BEA, IBM und Sun beteiligt, welche Konzepte aus selbst entwickelten, zueinander inkompatiblen, Portlet-APIs in den neuen Standard haben einfließen lassen.

Für Entwickler und IT-Manager ergeben sich dadurch handfeste Vorteile. Zum einen ist es die Möglichkeit der Unterstützung verschiedener JSR168-kompatibler Portal-Software mit einem Minimum an Änderungen. Zum anderen ergibt sich eine hohe Wiederverwendbarkeit von Quellcode, indem bestehende Anwendungen mit vertretbarem Aufwand zu Portal-Anwendungen migriert werden können.

2.1 Übersicht

Wie bereits erwähnt werden Inhalte bzw. Anwendungen durch Portlets auf Portalseiten repräsentiert. Der JSR 168 definiert eine Containerlösung, in der Portlets Inhalte bzw. Anwendungen darstellen, die auf Dienste des *Portlet-Containers* zugreifen und insbesondere Präsentations-, Sicherheits- und Personalisierungsfunktionen nutzen können. Weiterhin werden durch die Spezifikation die Semantik von Portlets, ihr Lebenszyklus („life cycle“) sowie ihre Interaktion beschrieben.

Der *Portlet-Container* ist verantwortlich für das Entgegennehmen von Requests, die Ausführung von Portlets und das Weiterleiten der Antwort an das Portal, das die Inhalte mehrerer Portlets zusammenfügt und zusätzliche Funktionen für Layout, Navigation oder Administration bereitstellt. Das Portal integriert also den *Portlet-Container*. Dies wird insb. am Beispiel des *Jetspeed-2* – Portals verdeutlicht, das auf die Referenzimplementierung *Pluto* als *Portlet-Container* zurückgreift, siehe Abschnitt 3.

Anzumerken bleibt, dass der JSR 168 nur den *Portlet-Container* definiert und weniger Aussagen über das Portal an sich macht.

Der JSR 168 basiert größtenteils auf der Servlet-Spezifikation [SRV01], wobei Portlets nur auf eine für sie sinnvolle Teilmenge der Funktionen von Servlets zurückgreifen können. Grundlegende Unterschiede zwischen Portlets und Servlets werden im Abschnitt 2.3 dargestellt.

Portlets stellen somit Web-Komponenten dar, die speziell darauf ausgerichtet sind, im Kontext einer Portalseite präsentiert zu werden. In der Regel erfolgen das Rendern mehrerer Portlets sowie die Rückgabe der zusammengeführten Inhalte an den Client in einem Request-Response-Zyklus. Dabei ist der Content-Type nicht auf HTML beschränkt, sondern es können auch andere Markup-Sprachen, wie bspw. WML, zurückgeliefert werden.

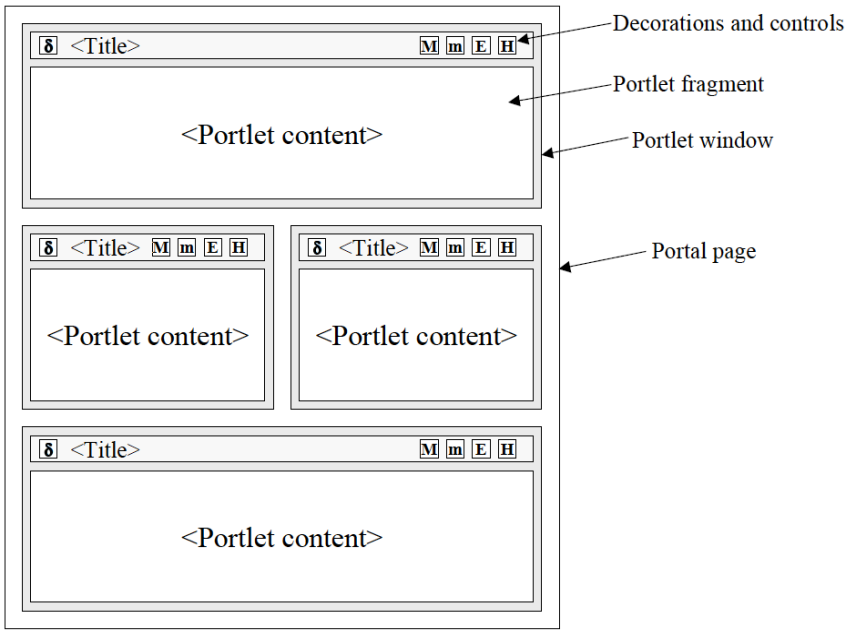

Abbildung 2: Elemente einer Portal-Seite, Quelle: [PLT03]

2.2 Konzepte

2.2.1 Portlet Life Cycle und Request-Sequenz

Portlets sind Singleton-Objekte und implementieren das Interface *javax.portlet.Portlet,* das folgende Methoden, die vom *Portlet-Container* aufgerufen werden, deklariert:

`init()`	Initialisierung der Portlet-Instanz
`destroy()`	Logik, die beim Zerstören der Portlet-Instanz ausgeführt wird
`processAction()`	(Controller-) Logik, reagiert auf Benutzerinteraktionen
`render()`	(Präsentations-) Logik, mit der das Portlet im Portal dargestellt, d.h. *Markup-Code* erzeugt wird *javax.portlet.GenericPortlet* unterteilt `render()` wiederum in `doView(), doEdit(), doHelp()`

An Portlets werden zwei verschiedene Arten von Requests gestellt: *Action-Requests* durch `processAction()` und *Render-Requests* durch `render()`. Daraus wird deutlich, dass Portlets in ihrem Lebenszyklus eher GUI-Komponenten entsprechen als herkömmlichen Web-Applikationen bzw. Servlets, die in einem *Request-Response-Zyklus* ausgeführt werden und Zustände durch *Sessions*

selbst nachbilden müssen. Portlets abstrahieren von diesem http-spezifischen *Request-Response-Zyklus* und können sich auf die Speicherung von *Render-Parametern* über mehrere Seitenaufrufe verlassen, d.h. sie behalten ihren Zustand. Diesem Mechanismus liegen letztendlich natürlich wieder Sessions zugrunde.

Zu beachten ist weiterhin, dass Portlets *thread-sicher* sein müssen, da der Container nur eine Instanz jeder Portlet-Klasse erzeugt. Durch mehrere parallele Request-Threads könnte es sonst zu *race conditions* kommen.

Das Rendern von mehreren Portlets auf einer Seite muss nicht sequenziell erfolgen, sondern es wird häufig eine parallele Strategie bevorzugt, so dass schnelle Portlets nicht auf langsamere warten müssen und folglich die Seite zügiger ausgeliefert wird. Weiterhin können die von `render()` gelieferten Inhalte in einem Cache gehalten werden, was die Effizienz von stark frequentierten Portalen erhöht. Das Caching von Portlets wird im Deployment Descriptor *portlet.xml* (siehe 2.2.5) konfiguriert.

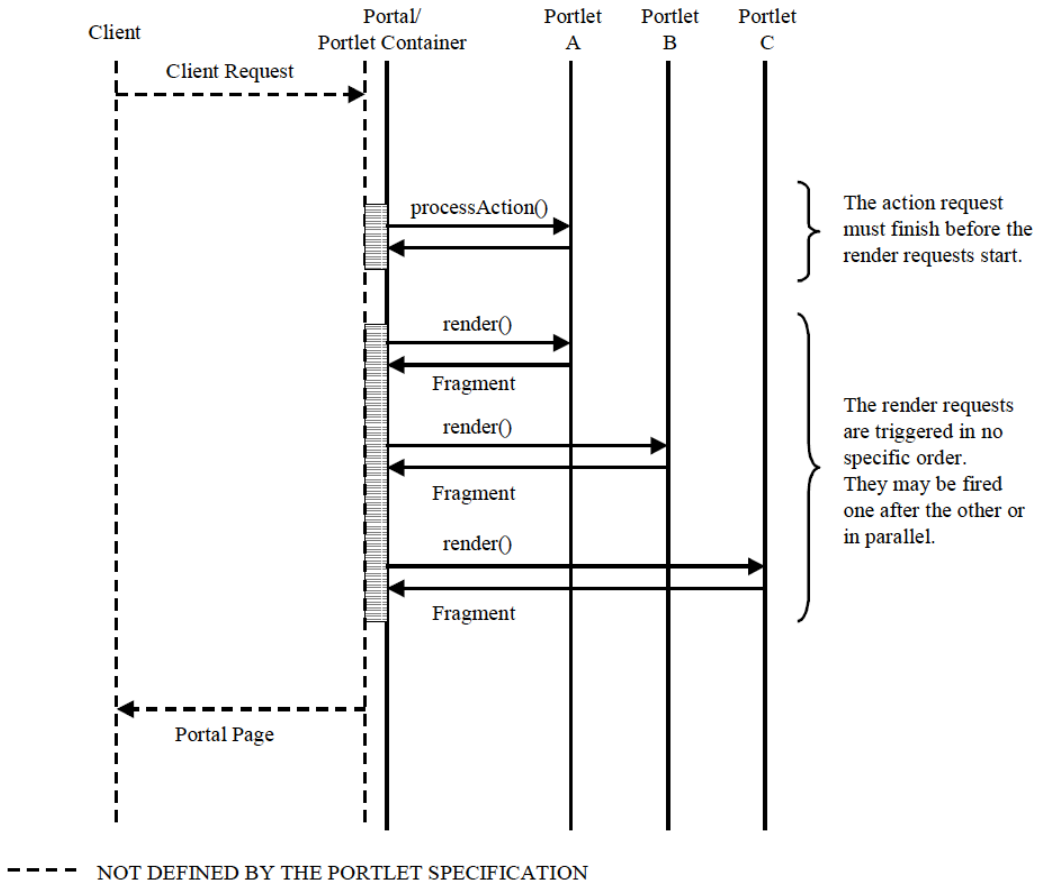

Abbildung 3: Request-Sequenz, Quelle: [PLT03]

2.2.2 Portlet-Modes und Window-States

Abbildung 4: Portlet-Titelleiste

Der Portlet-Container verwaltet zwei verschiedene Zustands-Typen: *Portlet-Modes* und *Window-States*.

Ein Portlet kann sich in verschiedenen Modi befinden, die sich in seiner Titelleiste widerspiegeln. So definiert die Spezifikation drei *Portlet-Modes*: „View“, „Edit“ und „Help“. Der erste bezeichnet den Standard-Modus des Portlets, der auf jeden Fall implementiert werden muss und in dem es seine normalen Funktionen bereitstellt. Im „Edit“-Modus können hingegen Konfigurationsoptionen editiert und im „Help“-Modus kontextspezifische Hilfe angezeigt werden. Zur Darstellung des Portlets werden die in 2.2.1 genannten Methoden `doView()`, `doEdit()` und `doHelp()` entsprechend ausgeführt.

Weiterhin kann sich das Portlet-Fenster, in Analogie zu normalen GUI-Komponenten, in den Zuständen NORMAL, MINIMIZED oder MAXIMIZED befinden. In letzterem nimmt das Fenster den gesamten zur Verfügung stehenden Platz ein und verdeckt somit alle anderen auf der Seite.

Die Portlet-Spezifikation bietet auch noch die Möglichkeit, eigene Zustände zu den oben genannten hinzuzufügen. Sinnvolle Erweiterungen wären beispielsweise Modi zum Drucken oder zur Vorschau von Portlets.

2.2.3 Portlet-Security

Ein großer Vorteil der Portlet-Spezifikation besteht darin, dass eine auf den *Java Authentication and Authorization Services* (JAAS) basierte Sicherheits-Infrastruktur definiert wird, deren Funktionen direkt von Portlets benutzt werden können.

Der Portlet-Container teilt Portlets die Rollen von Benutzern mit, die gerade auf sie zugreifen. Er ist nicht für die Authentifizierung verantwortlich, dafür werden Funktionen des zugrundeliegenden *Servlet-Containers* benutzt.

Programmatische Sicherheit in Portlets ist durch Nutzung der folgenden Funktionen des Interfaces *javax.portlet.PortletRequest* möglich: `getRemoteUser()`, `isUserInRole()`, `getUserPrincipal()`. Diese liefern dieselben Werte wie die gleichnamigen Methoden in *javax.servlet.http.HttpServletRequest,* wobei für `isUserInRole()` noch Zuordnungen zwischen Rollen-Namen im Deployment-Descriptor vorgenommen werden müssen [SRV01].

Deklarative Sicherheit ist durch die Spezifikation von *Security-Constraints* im Deployment-Descriptor *portlet.xml* möglich. Dadurch können für Portlets Integritäts- und Vertraulichkeitsanforderungen gesichert werden. Die Spezifikation schreibt hier mindestens SSL vor.

Weitergehende Sicherheitsmodelle werden in der Regel von der Portal-Implementierung bereitgestellt, insbesondere im Zusammenhang mit Zugriffsbeschränkungen auf Portal-Seiten oder Portlets im Kontext einer Portal-Seite. Die Portlet-Spezifikation geht darauf nicht näher ein.

2.2.4 Preferences und User Attributes

Portles haben Zugriff auf (benutzerspezifische) Konfigurationsinformationen („*Preferences*“) in Form von Schlüssel/Wert-Paaren mit Hilfe des Interfaces *javax.portlet.PortletPreferences*. Sie können sowohl während Render- als auch Action-Requests darauf zugreifen, wobei die Modifikation nur bei Action-Requests möglich ist. Die Persistenz der *Preferences* wird von der Portal-Implementierung gesichert.

Informationen über den Benutzer (*„User Attributes“*) erhalten Portlets als nichtveränderbare *Map* (also wieder in Form von Schlüssel/Wert-Paaren) über die *Request Attributes*. Falls der Benutzer authentifiziert wurde, liefert die Methode `javax.portlet.PortletRequest.getAttribute( PortletRequest.USER_INFO )` die entsprechende *Map*. Alle für Portlets verfügbaren *User Attributes* müssen im Deployment Descriptor *portlet.xml* angegeben werden, sonst ist kein Zugriff möglich. Die Spezifikation enthält weiterhin Vorschläge für Schlüssel bestimmter Benutzerinformationen, bspw. „user.name.given“ für Vornamen oder „user.name.family“ für Nachnamen [CRA02].

2.2.5 Deployment und Portlet Applications

Eine Portlet-Applikation („Portlet Application“) ist grundsätzlich auch eine Web-Applikation, wie sie in der Servlet-Spezifikation definiert ist. [SRV01]. Entsprechend wird sie auch in einem *web application archive* (WAR) zusammengefasst. Darin befinden sich neben weiteren Servlets, JSPs, Bildern und anderen Ressourcen auch die Deployment Descriptor *web.xml* sowie *portlet.xml*. Wurden keine spezifischen Portal-Funktionen benutzt, so kann das WAR problemlos in verschiedenen JSR168-kompatiblen Portalen *deployt*[3] werden.

Der Deployment Descriptor portlet.xml enthält alle Portlets und damit verbundene Einstellungen. Für eine detaillierte Beschreibung wird auf die Portlet-Spezifikation PLT.21 [PLT03] verwiesen. Hier soll folgendes Beispiel aus [SUN03] genügen:

```
<portlet-app>
  <portlet>
    <portlet-name>WeatherPortlet</portlet-name>
    <portlet-class>sample.portlet.WeatherPortlet</portlet-class>
    <init-param>
      <name>weather.url</name>
      <value>java:/comp/env/WeatherProvider</value>
    </init-param>
    <expiration-cache>3600</expiration-cache>
    <supports>
```

[3] von “to deploy“

```
      <mime-type>text/html</mime-type>
      <portlet-mode>EDIT</portlet-mode>
      <portlet-mode>HELP</portlet-mode>
    </supports>
    <portlet-info>
      <title>WeatherPortlet</title>
    </portlet-info>
    <portlet-preferences>
      <preference>
        <name>zip</name>
        <value>95054</value>
      </preference>
      <preference>
        <name>unit</name>
        <value>F</value>
      </preference>
      <preferences-validator>
        sample.portlet.WeatherPreferencesValidator
      </preferences-validator>
    </portlet-preferences>
  </portlet>
</portlet-app>
```

2.3 Unterschiede zwischen Servlets und Portlets

Die Servlet-Spezifikation [SRV01] definiert den Begriff Servlet folgendermaßen:

“A servlet is a Java technology based web component, managed by a container, that generates dynamic content. Like other Java-based components, servlets are platform independent Java classes that are compiled to platform neutral bytecode that can be loaded dynamically into and run by a Java enabled web server. Containers, sometimes called servlet engines, are web server extensions that provide servlet functionality. Servlets interact with web clients via a request/response paradigm implemented by the servlet container.”

Zwischen Portlets und Servlets existieren viele Gemeinsamkeiten. Sie sind in erster Linie Web-Komponenten, die im Kontext eines bestimmten Containers ausgeführt werden. Dieser Container steuert die Lebenszyklen von Servlets und Portlets, deren Methoden-Einstiegspunkte von Interfaces definiert werden. Sie generieren dynamische Inhalte und interagieren mit dem Client auf der Basis eines Request-Response-Zyklus.

Unterschiede zwischen beiden Technologien sind hauptsächlich die folgenden: So liefern Servlets komplette Dokumente an den Client, während Portlets nur Fragmente von Markup-Code erzeugen, d.h. sie interagieren durch ein Portal mit dem Benutzer. Sie besitzen definierte Portlet Modes und Window States, die ihre aktuelle Funktionalität sowie ihren Zustand im Kontext einer Portal-Seite widerspiegeln. Wie bereits erwähnt, haben Portlets einen genauer definierten Request-Lebenszyklus, der mit Action- sowie Render-Requests ähnlich dem von GUI-Elementen ist und somit das MVC-Pattern impliziert. Weiterhin sind sie nicht an eine bestimmte URL gebunden, diese wird dynamisch vom *Portlet Container* erzeugt und Portlets setzen nur Render-Parameter. Die eigentliche (Seiten-) Navigation ist Aufgabe der Portal-Implementierung.

2.4 Zusammenfassung und Ausblick

Die Portlet-Spezifikation stellt das entscheidende Bindeglied zwischen vielen kommerziellen sowie Open-Source-Portalen dar und hat, obwohl es noch ein relativ neuer Standard ist, bereits eine bemerkenswerte Verbreitung erreicht. Um die langfristige Bedeutung des JSR 168 zu verstehen, muss man die Situation für Entwickler vor der Spezifikation betrachten. Neben allen Vorteilen, die verschiedene Portale boten, musste immer auf eine spezielle Implementierung gesetzt werden. Während das generelle Konzept von Portalen allgemein angenommen und gut akzeptiert wurde, gab es sehr viele (inkompatible) Möglichkeiten, eine Portal-Implementierung zu entwickeln.

Der JSR 168 bietet nun für Entwickler Zukunftssicherheit und ein klares Rahmenwerk für die Umsetzung ihrer Portale. Zusammen mit dem WSRP-Standard kann eine komplexe Applikations-Infrastruktur einheitlich und konsistent Endbenutzern zur Verfügung gestellt werden.

Über Eigenschaften zukünftiger Versionen wurden in der Spezifikation bereits Andeutungen gemacht. Zum einen sind dies Portlet-Filter sowie Inter-Portlet-Kommunikation und zum anderen Möglichkeiten, dass Portlets auch Einfluss auf Markup-Code haben können, der außerhalb ihres Fragments liegt. In [DST04] werden zusätzlich weitergehende Verbesserungsvorschläge für die aktuelle Spezifikation genannt. Das sind u.a. fomale Modell-Definitionen für Portlet-Fenster und Portlets im Seitenkontext, mehr Integrationspunkte im Render-Zyklus sowie allgemein bessere Erweiterungsmöglichkeiten von Portlets.

3 Web Services for Remote Portlets – WSRP

Die Web Services for Remote Portlets Specification [KRO03] definiert einen herstellerübergreifenden Standard, mit dem Portale Ein- und Ausgaben von Portlets, die auf entfernten Portal-Servern laufen, steuern können Die Spezifikation wurde auf den JSR 168 ausgerichtet, ist selbst aber sprachunabhängig.

Der Standard hat eine große Bedeutung in der Kooperation von Portalen, so ist es insbesondere auch möglich, dass Java-Portlets in .NET-Portalen und umgekehrt erscheinen. Im Folgenden soll nur ein kurzer Überblick zur Spezifikation gegeben werden, für weitergehende Informationen sei auf [KRO03] oder [LIN04] verwiesen.

Mit WSRP ist ein Portal, oder mitunter auch ein Cluster, für das Hosting von Portlets verantwortlich („*Producer*“), während andere („*Consumer*“) sie so anzeigen können, als ob sie lokal laufen würden. Wie der Name schon sagt setzt dieser Standard auf der Web-Service-Technologie bzw. SOAP[4] und WSDL[5] auf, mit der Systeme lose auf einer strukturierten semantischen Basis gekoppelt werden können.

Im Gegensatz zu anderen, meist daten-orientierten Web-Services, ist WSRP präsentations-orientiert, d.h. auf Anfragen von Clients werden Markup-Sprachen-basierte Antworten geliefert. Die WSRP-Implementierung verbirgt Details der Kommunikation zwischen den Portalen. *Consumer* wählen lediglich die Portlets aus, die auf ihrem Portal angezeigt werden sollen. Applikations-Logik verbleibt so komplett beim *Producer*.

Abbildung 5 zeigt beispielhaft ein *Consumer*-Portal mit drei WSRP-Portlets, die über SOAP mit ihren *Producer*-Portlets im entfernten Portal kommunizieren.

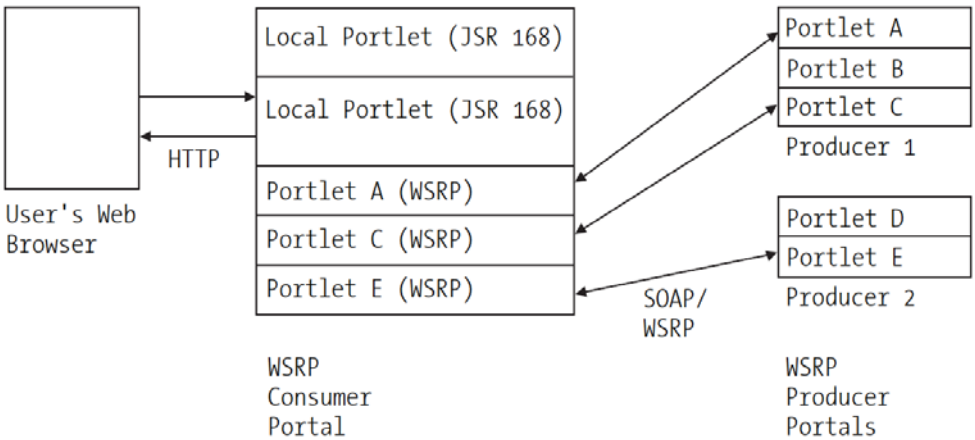

Abbildung 5: WSRP: zwei Producer und ein Consumer, Quelle: [LIN04]

[4] http://www.w3.org/TR/soap
[5] http://www.w3.org/TR/wsdl

3.1 Producer

Producer stellen Container für Portlets dar. Die Spezifikation definiert vier Web-Services, die von *Consumern* benutzt werden können. Dabei müssen die ersten beiden Schnittstellen implementiert werden, die anderen sind hingegen optional.

- “Service Description“: Stellt Informationen und Metadaten über den *Producer* bereit, wie bspw. welche Portlets gehostet werden
- “Markup“: Schnittstelle, die Markup-Code von Portlets bereitstellt und zur Interaktion benötigt wird
- “Registration“: Consumer können mit dieser Schnittstelle dem Producer ihre eigenen Informationen, Fähigkeiten und Metadaten bereitstellen
- “Portlet Management“: Konfiguration und Anpassung von bereitgestellten Portlets durch den Consumer

3.2 Consumer

Consumer kommunizieren mit *Producern* um Portlets aufzufinden, Markup-Code durch *Render-Requests* zu erhalten oder *Action-Requests* an entfernte Portlets weiterzuleiten. Die Markup-Fragmente werden dann im *Consumer*-Portal auf herkömmliche Weise zusammengeführt und dem Benutzer präsentiert.

In der Regel existieren lokale Proxy-Portlets, die alle WSRP-Funktionen über SOAP transparent an den *Producer* weiterleiten.

3.3 Fazit

Der Producer muss nicht unbedingt selbst ein Portal sein, das direkten Zugriff auf Portlets bietet. In einem großen Unternehmen könnte der *Producer* ein zentraler Server-Cluster sein, der Zugriff nur von seinen eigenen, verteilten Portalen erlaubt. Diese Lösung bietet eine nicht zu verachtende Kostenersparnis gegenüber einem Deployment der Portlets auf allen Portalen.

Da weiterhin der JSR168 und WSRP Portlet-Kernkonzepte teilen, können standardkonforme Portlets ohne Änderungen sofort als *Producer*-Portlets benutzt werden. Die Spezifikation bietet somit Herstellern einen Standard, der viele inkompatible Eigenentwicklungen erfolgreich verhindern kann.

4 Jetspeed-2

Eines der interessantesten JSR168-kompatiblen Open-Source-Portale ist zweifelsohne Jetspeed-2 [JS04] aus dem *Apache Portals Project*[6], das sich allerdings mit der aktuellen Version Milestone 3 noch in der Entwicklung befindet.

4.1 Übersicht

Jetspeed-2 stellt eine komplette Neuentwicklung von Jetspeed-1 dar. Neben vielen Architektur-Verbesserungen und Erweiterungen gegenüber dem Vorgänger, bietet es komplette JSR 168-Unterstützung, indem es die Referenzimplementierung *Apache Pluto*[7] als Portlet Container integriert. Eine bessere Skalierbarkeit und eine auf Spring basierende Komponenten-Architektur lassen es mit kommerziellen Portalen konkurrieren. Einziger Wermutstropfen ist die derzeit noch sehr magere Dokumentation und schlechte Strukturierung der Quellen, was einen nicht unerheblichen Mehraufwand in der Einarbeitung bedeutet.

4.2 Features

4.2.1 CMS-basierte Seitennavigation

Die Navigationskomponente befindet sich derzeit in einer kompletten Überarbeitung. Geplant ist, die Seitenstruktur transparent auf einem Content-Repository basieren zu lassen. Diese könnte zum einen auf Dateisystemen oder Datenbanken zugreifen, zum anderen ist aber auch ein CMS als Backend möglich. Durch ein *Profiling* der Inhalte können weiterhin verschiedenen Benutzern unterschiedliche Sichten auf Inhalte ermöglicht werden. *Profiling* bezeichnet die Verwendung eines dynamischen Regelinterpreters, der aufgrund verschiedener Kriterien entscheidet, welche Seiten dem Client geliefert werden.

4.2.2 CMS-Lösung: Graffito

Die bevorzugte CMS-Lösung für Jetspeed-2 wird in dem Apache-Incubator-Projekt Graffito [GRF04] entwickelt. Es stellt eine komplette Plattform zur Erstellung, Verwaltung und Veröffentlichung von Inhalten dar und integriert weiterhin Content-Repository-, Workflow-, Kollaborations-, Versionierungs-, Sicherheits- und Personalisierungs-Funktionen.

Als virtuelles Content-Repository ist es außerdem fähig, verschiedene Standards, insb. JSR 170 (Java Content Repository – JCR) und WebDAV zu unterstützen.

[6] http://portals.apache.org
[7] http://portals.apache.org/pluto

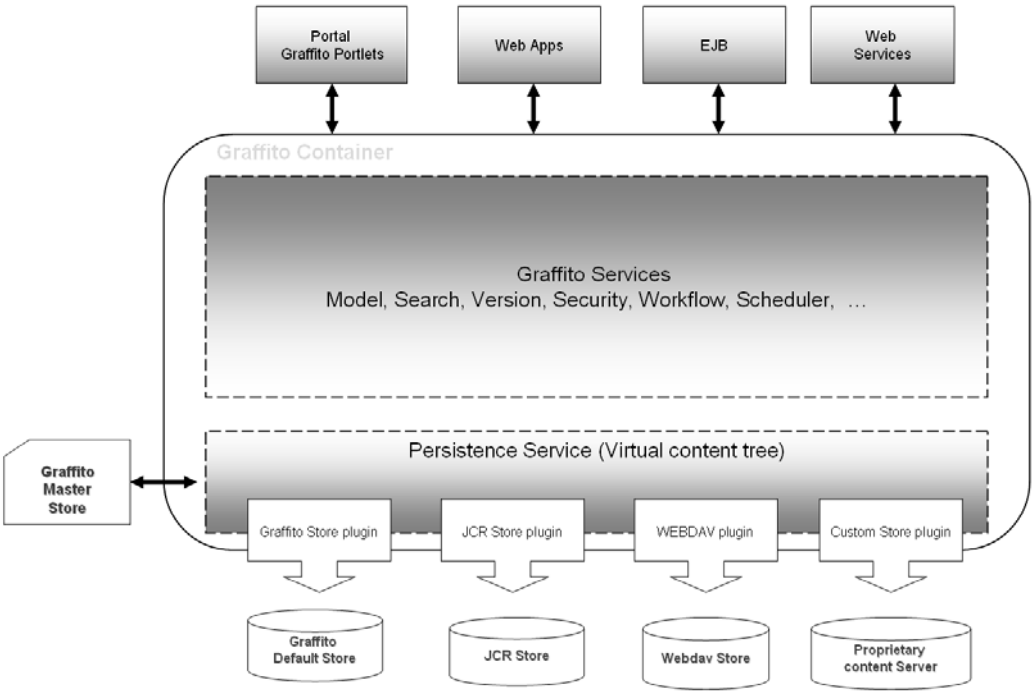

Abbildung 6: Graffito-Architektur, Quelle: [GRF04]

4.2.3 Single Sign On

Zur Integration verschiedener heterogener Systeme existieren Portlets, die deren Ausgaben in einem Portlet-Fenster darstellen. Dafür sind zunächst Änderungen am Markup-Code nötig (nicht erlaubte Elemente entfernen) und zum anderen müssen URLs umgeschrieben werden, sodass sie wieder auf das Portal zeigen.

Ist eine Authentifizierung nötig, so bietet Jetspeed-2 die Möglichkeit, Logindaten dieser integrierten Anwendungen zentral im Portal zu speichern und die Authentifizierung beim Aufruf selbständig vorzunehmen. Diese Funktionalität wird allgemein als *Single Sign On* (SSO) bezeichnet, da für den Benutzer nur ein einmaliges Login am Portal nötig ist.

4.2.4 Customization

Benutzer können sich ihre Seiten mit Portlets selbst zusammenstellen und deren Layout ändern. Dies ist abhängig von ihren zugewiesenen Rechten und wird dauerhaft gespeichert. Hier steckt leider in der aktuellen Version noch der Teufel im Detail. Abbildung 7 zeigt die Veränderungsmöglichkeiten einer Jetspeed-Portal-Seite.

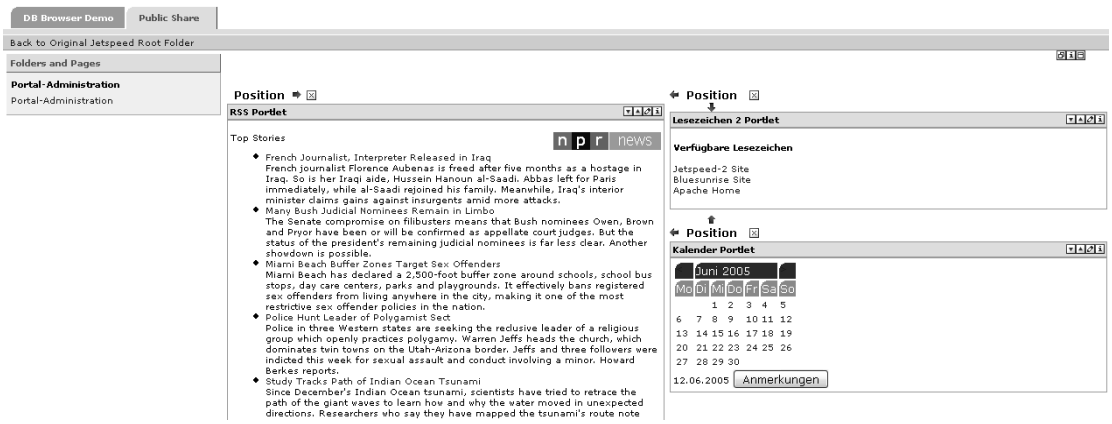

Abbildung 7: Customization

4.2.5 Portal-Administration

Jetspeed-2 liefert komplette Administrations-Portlets, mit denen verschiedene Aspekte des Portals administriert werden können. Dies umfasst u.a. das Benutzer- (siehe Abbildung 8), Rollen- und Gruppenmanagement, aber auch die Verwaltung von Portlet-Applikationen und Seitenstrukturen. Leider befinden sich auch diese noch in der Entwicklung, sodass mit der aktuellen Version doch einiges an Handarbeit notwendig ist.

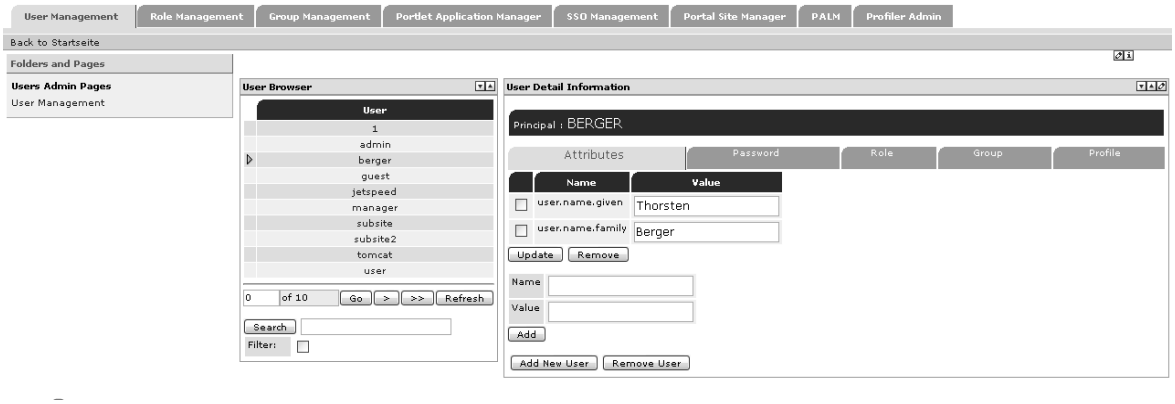

Abbildung 8: Portal-Administration

4.2.6 Portals Bridges

Ein Unterprojekt von Jetspeed-2 stellen die *Portals Bridges* dar, die auch in anderen JSR168-kompatiblen Portalen lauffähig sein sollen. So ist die Integration von bestehenden Anwendungen, die mit Struts, Java Server Faces (JSF) oder sogar mit PHP und Perl geschrieben sind, mit wenig Aufwand möglich.

Ein Hauptproblem, das die *Bridges* dabei lösen müssen, sind die bereits genannten Unterschiede beim Request-Handling von Portlets im Gegensatz zu herkömmlichen Request-Response-Zyklen anderer Webapplikationen.

Zum Beispiel führen Struts-Actions normalerweise ihre Controllerfunktionalität als Reaktion auf Formulareingaben aus und initiieren ein Forward zu JSP-Seiten. Jedoch dürfen Portlets während *Action-Requests* keine Ausgaben erzeugen. Dies ist nur bei *Render-Requests* möglich. Die Struts-Bridge löst dieses Problem, indem sie die Action-Ausführung vor einem Forward unterbricht, den Ausführungskontext sichert und ein *Redirect* an den Browser sendet, damit im folgenden *Render-Request* die Ausgabe der Struts-Action dargestellt werden kann.

Weitere Aufgaben von Bridges betreffen das URL-Handling und das Verändern von Markup-Code zur Anzeige im Seitenkontext.

Abbildung 9 zeigt einen eigenständig lauffähigen Webshop, der mittels Struts-Bridge in Jetspeed ausgeführt werden kann.

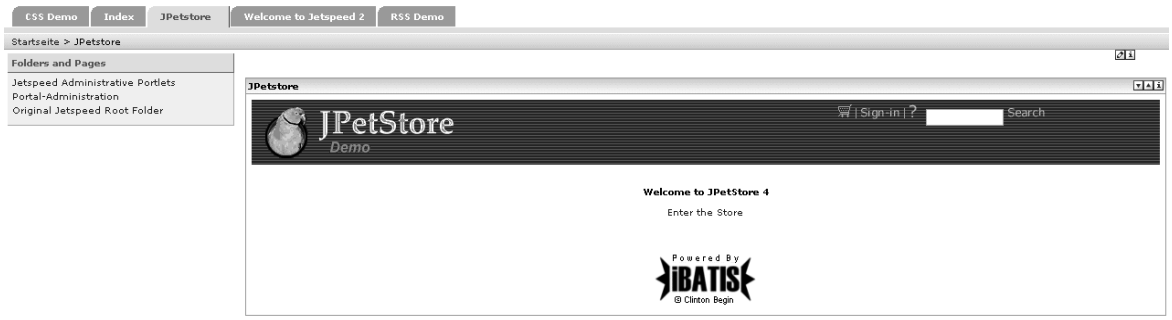

Abbildung 9: Eine Struts-Anwendung als Portlet

4.3 Architektur

4.3.1 Übersicht

Jetspeed-2 ist eine J2EE-Webapplikation und wurde in verschiedenen Servlet-Containern, wie insbesondere Apache Tomcat, JBOSS sowie dem IBM Websphere Application Server getestet.

Eine clusterfähige Architektur gewährleistet hohe Skalierbarkeit auch bei anspruchsvollen Portalen. Mit der „Multi-threaded portlet aggregation engine" kann, wie bereits in Kapitel 2 erwähnt, das Rendern von Portlets parallel erfolgen.

Das Sicherheitsmodell basiert auf JAAS, wodurch u.a. ein problemloser Austausch der Authentifizierungskomponenten möglich wird. Standardmäßig benutzt Jetspeed-2 eine relationale Datenbank als *AuthenticationProvider*, liefert aber auch eine Implementierung, mit der gegen ein LDAP-System authentifiziert werden kann.

Als Persistenzschicht wird schließlich der Objekt-Relationale-Mapper *Apache OJB*[8] genutzt.

[8] http://db.apache.org/ojb

4.3.2 Spring-Framework

Das Spring-Framework [JO02] ist ein J2EE-kompatibles Komponenten-Framework, das von Jetspeed-2 verwendet wird. Es stellt einen alternativen Architekturansatz zu EJB[9] dar, indem es sich mit dem Konzept des leichtgewichtigen Containers wieder auf die Implementierung von *Plain Old Java Objects* (POJOs) konzentriert, die in verschiedenen Umgebungen lauffähig sind. Sie benötigen im Gegensatz zu EJB (bis Version 2) keinen schwergewichtigen Container und müssen auch nicht Home- bzw. Remote-Interfaces implementieren, um lauffähig zu sein.

Weitere Kernkonzepte von Spring sind:

„Inversion of control“: Komponenten können sich auf eine fertige Konfiguration von außen verlassen, d.h. die Konfiguration wird vom Container in sie „injiziert“, was eine hohe Ortsunabhängigkeit zur Folge hat.

Aspektorientierte Programmierung: Alle von Spring verwalteten Objekte können aspektorientiert verwendet werden, so ist es bspw. möglich, Klassen durch Aspekte ohne EJB oder JTA[10] transaktionssicher zu machen oder Loggingfunktionalität hinzuzufügen. Dies geschieht deklarativ an zentraler Stelle.

Persistenz-Abstraktions-Schichten: Spring integriert bspw. Hibernate sowie JDO und stellt u.a. *Data Access Objects* (DAOs) zum transparenten Datenzugriff bereit.

MVC-Webapplikations-Framework: Dies sei nur der Vollständigkeit halber genannt und wird nicht von Jetspeed-2 verwendet.

4.4 Zusammenfassung und Ausblick

Eigene Erfahrungen können Jetspeed-2 ein gutes Potenzial bestätigen um zukünftig eine Alternative neben kommerziellen Portalen darstellen zu können. Insbesondere die anvisierte CMS-Lösung Graffito dürfte nicht unerheblich dazu beitragen.

Der Erfolg steht und fällt allerdings mit der Verfügbarkeit hochwertiger Dokumentation. Zur Zeit kann man diese leider nur als chaotisch bezeichnen. Auch die anvisierte Refaktorisierung des Source-Repositories ist dringend nötig um die Lesbarkeit des Jetspeed-2-Quellcodes zu erhöhen.

Diese Mängel stellen derzeit noch ein großes Risikopotenzial dar, traten in der Entwicklung von Portlet-Applikationen doch manchmal Probleme auf, die nur mit größter Mühe unter Zuhilfenahme des Jetspeed-2-Quellcodes beseitigt werden konnten.

[9] http://java.sun.com/products/ejb
[10] http://java.sun.com/products/jta

Nichtsdestotrotz arbeiten die Jetspeed-2-Entwickler an der Beseitigung dieser Probleme bis zum Final-Release 2.0. Ein Termin dafür steht leider noch nicht fest, jedoch soll die nächste Version M4 der letzte Milestone vor der endgültigen Fertigstellung sein.

5 Fazit

Portale sind eine eigenständige Art von Technologie, die derzeit von der J2EE und einer überschaubaren Anzahl von Herstellern dominiert wird. Obwohl Portaltechnologie immer noch ein relativ junges Feld ist, so zeichnet sich mit den beiden vorgestellten Standards JSR 168 und WSRP eine Konsolidierung des Marktes ab. Insbesondere die Interoperabilität zwischen den Portalen verschiedener Hersteller wird dadurch in sinnvoller Weise gefördert.

Portaltechnologie lässt sich kaum mit anderen vorhandenen Webapplikations-Systemen vergleichen. Es existieren zwar Frameworks, die Webseiten unterteilen und so von verschiedenen Stellen generierten Markup zusammenfügen können, wie beispielsweise *Struts Tiles*, jedoch können diese nicht mit der Mächtigkeit von Portal-Servern mithalten.

Vollständige kommerzielle und offene Portal-Server sollen für Systembetreiber den Einstieg erleichtern. Für zur Verfügung stehende Portlet-Applikationen und -Services kann sich mittelfristig ein Markt entwickeln, der neue Bereiche und Anwendungsmöglichkeiten erschließt. Beispielsweise könnten viele Management-Programme als Remote Portlets interessante integrierte Zugriffsmöglichkeiten für Benutzer bieten.

Weiterhin bleibt abzuwarten, ob es auch abseits der J2EE entsprechende Standards für Portale geben wird. Es existiert noch kein entsprechender Standard für .NET-Portale. Die sprach- und herstellerunabhängige WSRP-Spezifikation bereichert hier zwar beide Welten, ist aber eher auf den JSR 168 zugeschnitten.

Insofern nicht andere Gründe gegen die J2EE sprechen sollten, kann man also, in Ermangelung konkurrierender Standards, derzeit den JSR 168 als Spezifikation der Wahl bezeichnen.

Referenzen

[CRA02] CRANOR, Lorrie et. al. The Platform for Privacy Preferences 1.0 Specification. http://www.w3.org/TR/P3P (Juni 2005). 04 2002

[DST04] TAYLOR, David Sean. What's Happening with Jetspeed-2? http://today.java.net/pub/a/today/2004/11/22/jetspeed2.html (Juni 2005). 11 2004

[GRF04] Apache Software Foundation. Graffito. http://incubator.apache.org/projects/graffito.html (Juni 2005)

[HOE04] HÖLLER, Jürgen. The Spring Framework: Introduction to Lightweigt Architecture. http://www.javapolis.com/JP04DVDContent/talks/SpringInAction2/index.html (Juni 2005). 12 2004

[JO02] JOHNSON, Rod. The Spring Framework. http://www.springframework.org (Juni 2005). 2002

[JS04] APACHE SOFTWARE FOUNDATION. Jetspeed-2. http://portals.apache.org/jetspeed-2/ (Juni 2005). 2004

[KRO03] KROPP, Alan et. al. Web Services for Remote Portlets Specification. http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf (Juni 2005). 08 2003

[LIN04] LINWOOD, Jeff; MINTER, Dave: *Building Portals with the Java Portlet API.* Berkeley: Apress, 2004. ISBN 1–59059–284–0

[PLT03] ABDELNUR, Aleandro; HEPPER, Stefan. Java Portlet Specification. http://www.jcp.org/en/jsr/detail?id=168 (Juni 2005). 10 2003

[SRV01] PELEGRI-LLOPART, Eduardo et.al.: Java Servlet 2.3 and Java Server Pages 1.2 Specifications. http://www.jcp.org/en/jsr/detail?id=53 (Juni 2005). 09 2001

[SUN03] SUN MICROSYSTEMS. Introduction to JSR 168 – The Java Portlet Specification. http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168/pb_whitepaper.pdf (Juni 2005). 07 2003

[TH03] THEIS, Fabian et.al.: *Portale und Webapplikationen mit Apache Frameworks.* Frankfurt: Software & Support Verlag, 02 2003, ISBN 3-93504-236-1

[WEG02] WEGE, Christian: Portal Server Technology. In: *IEEE Internet Computing* May/June (2002), Seiten 73-77

[WEL02] WELLS, Stuart C. JSR 168: Standardizing the Power of Portals. http://java.sys-con.com/read/37183.htm (Juni 2005). 10 2002

[WO05] Wolff, Eberhard: Den Frühling im Netz. In: *Javamagazin* 06 (2005), Seiten 48-51

[WO205] WOLFF, Eberhard: „Offenheit ist der Grund für den Erfolg von Spring“. In: *Javamagazin* 5 (2005), Seiten 78-80