

Technische Universität Wien

Seminararbeit

Java Portlets

Die Spezifikation und deren Anwendung

ausgeführt am Institut für Softwaretechnik und Interaktive Systeme der
Technischen Universität Wien

durch

Peter Gerstbach
Mat.Nr.: 0125397
Alexander Trieb
Mat.Nr.: 0127384

Mai 2004

Abstract

Unternehmen setzten im Inter- und Intranet seit Jahren Portale ein, um die Informationsflut zu bewältigen, doch das Fehlen von Standards machte Portal-Projekte inkompatibel und kostspielig. Seit Oktober 2003 existiert nun der JCP-Standard *Java Portlets*, der dieses Problem in der Java-Welt beseitigen soll. Diese Arbeit gibt einen Überblick über die Konzepte der Spezifikation, beschreibt die Integration in vorhandenen Anwendungen und gibt einen Überblick über freie und kommerzielle Angebote, die den Standard implementieren.

Inhaltsverzeichnis

1	Einleitung	1
2	Spezifikation	3
2.1	Portal Definition	3
2.1.1	Portlet	3
2.1.2	Portlet Container	3
2.2	Portlet Life Cycle	3
2.2.1	Instanzierung	4
2.2.2	Initialisierung	4
2.2.3	Portlet Window	5
2.2.4	Anfrage-Behandlung	5
2.2.5	Beenden	6
2.3	PortletURL	6
2.3.1	Portlet URL Modes	7
2.3.2	Portlet Parameter	7
2.3.3	Portlet URL Erweiterungen	7
2.4	Portlet Modes	8
2.4.1	VIEW Mode	8
2.4.2	EDIT Mode	8
2.4.3	HELP Mode	8
2.4.4	CUSTOM Mode	8
2.5	Portlet Requests	9
2.5.1	Request Parameter	10
2.5.2	Request Attribute	10
2.5.3	Request Properties	11
2.5.4	Request Security	11
2.5.5	Weitere Eigenschaften	11
2.6	Portlet Response	12
2.6.1	Response Properties	12
2.6.2	Action Responses	12
2.6.3	Render Responses	13
2.6.4	URL Encoding	13

Inhaltsverzeichnis

2.7	Packaging und Deploying	13
2.7.1	Packaging	13
2.7.2	Portlet Deployment Descriptor	14
2.8	PortletConfig	14
2.9	PortletContext	15
2.10	PortletPreferences	15
2.11	PortletSession	16
2.12	UserInformation	17
2.13	PortletCaching	18
2.14	PortletSecurity	18
2.14.1	Security Constraints	18
2.15	TagLibs	19
3	Integration	21
3.1	Servlet Integration	21
3.2	JSP Integration	22
3.2.1	ActionURL	22
3.2.2	RenderURL	23
3.3	Web Service Remote Portlet Integration	23
3.3.1	Prozessablauf	24
4	Systemumgebung	27
4.1	Kommerzielle Anbieter	28
4.1.1	OracleAS 10g Portal	28
4.1.2	Sybase Enterprise Portal 6.0	28
4.1.3	Sun Microsystems Java Enterprise Solution	28
4.1.4	IBM WebSphere Portal 5.0	28
4.1.5	BEA Systems WebLogic Portal 8.1	29
4.2	Freie Anbieter	29
4.2.1	Jakarta Pluto	29
4.2.2	Jetspeed	29
4.3	Empfehlung	29
5	Case Study	31
5.1	Überblick	31
5.2	Installation und Konfiguration	31
5.3	Architektur	32
5.4	Test Suite	33
6	Fazit	35

1 Einleitung

Seit Anfang des Internets war die Notwendigkeit gegeben, diversifizierte Informationen für die verschiedenen Anwender effizient, übersichtlich, personalisiert und organisiert zur Verfügung zu stellen. Daraus ergab sich der Bedarf für Portalsysteme. Dieser Bedarf war durch einzelne Haushalte bis hin zu Unternehmen, die produkt- und dienstleistungsbezogene Informationen anbieten wollten, gegeben.

Durch die Notwendigkeit, maßgeschneiderte Informationen anzubieten, wurden verschiedenste Technologien entwickelt, die den Aufbau von solchen Systemen effektiver und kostengünstiger gestalten. Erst seit kurzem wurde von dem *Java Community Process* ein *Java Specification Request (JSR)* erstellt, der genau diesen Bedarf abdecken soll. Jetzt als *JSR 168* bekannt, wurde die Spezifikation als eine Erweiterung der bekannten Java Servlets Technologie erstellt. Diese neue Spezifikation beschreibt eine neue, auf Java basierende Technologie namens Java Portlets.

Um ein besseres Verständnis für die technischen Möglichkeiten des Java Portlets Frameworks zu gewinnen, werden wir im ersten Kapitel eine detaillierte Analyse der Spezifikation durchführen. Im zweiten Kapitel werden wir die verschiedenen Integrationsmöglichkeiten, die Java und das Portlets Framework anbieten, analysieren (z.B. Java Server Pages und Servlets, Web Service Integration).

Erst wenn innerhalb der IT Community ein neues Framework akzeptiert wird, entsteht der Wert dieser neuen Technologie. Deswegen werden wir in einem weiteren Kapitel nach existierenden bzw. zukünftigen Systemumgebungen forschen, die das Java Portlets Framework unterstützen. Um auch einen besseren Einblick über das Potenzial dieses Ansatzes zu bekommen, werden wir nicht nur die kommerziellen Produkte (wie z.B. BEA WebLogic, IBM WebSphere, etc.) sondern auch die Open Source Ansätze (z.B. der Apache Software Foundation) betrachten. In einem weiteren Kapitel werden wir anhand der Referenzimplementierung die Integration eines einfachen Portlets zeigen.

Nach einer genauen Analyse der Spezifikation, einem Vergleich mit anderen Lösungsansätzen und Technologien, einer Präsentation verschiedener existierender bzw. zukünftiger Systemumgebungen und einem Implementierungsbeispiel, werden wir unsere Ergebnisse und Erkenntnisse diskutieren. Daraus wird dann eine Schlussfolgerung die Potenziale dieses neuen Ansatzes zeigen.

1 *Einleitung*

2 Spezifikation

2.1 Portal Definition

Ein Portal ist eine web-basierte Applikation, die auf Aggregation personalisierter Inhalte beruht. Das Portal erlaubt einem Benutzer bei einmaligem Anmelden, von mehreren Informationsquellen Informationen zur Verfügung gestellt zu bekommen. Ein Portal kann in anspruchsvoller Weise an verschiedene Benutzer und Geräte angepasst werden.

2.1.1 Portlet

Ein Portlet ist ein Teil einer Portal Seite. Genauer gesehen ist ein Portlet ein auf die Präsentationsebene gerichtetes Objekt, das entweder statischen oder dynamischen Inhalt erzeugen bzw. anzeigen kann. "Portlets werden von Portal Systemen als eine 'pluggable user interface' Komponente gesehen, die als Präsentationsebenen zu existierenden Informationssystemen dienen." [1, Seite 13]

Der Inhalt, der von einem Portlet generiert wird, wird als *Fragment* bezeichnet. Ein Fragment bezieht sich auf einen Teil einer Markup-Sprache (z.B. HTML, XML) und kann wieder aus mehreren Fragmenten bestehen. Die Aggregation von diesen Portlets bildet eine Portal Seite.

Portlets werden durch Request/Response-Methoden angesprochen und reagieren dementsprechend darauf. Die Eigenschaften eines Portlets (z.B. Window State) sowie der Inhalt können verändert werden.

2.1.2 Portlet Container

Der Portlet Container, ähnlich wie der Servlet Container, kontrolliert und verwaltet die einzelnen Portlets. Die Requests, die von einem bestimmten Client stammen, werden vom Portlet Container an das notwendige Portlet weitergeleitet. Der Portlet Container verwaltet auch die Erzeugung so wie die Zerstörung der einzelnen Portlets.

2.2 Portlet Life Cycle

Ein Portlet kann sich in verschiedenen Phasen befinden, diese Phasen werden durch den Lebenszyklus definiert. Wir werden diesen Zyklus kurz beschreiben, um einen besseren Überblick des gesamten Ansatzes zu gewährleisten. Dieser Zyklus besteht

2 Spezifikation

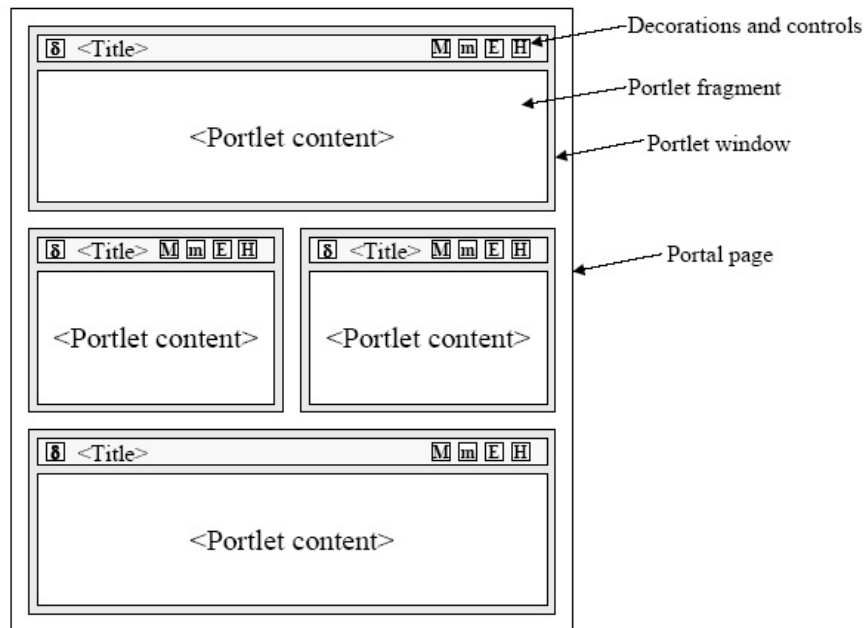


Abbildung 2.1: Elemente eines Portals [1, Seite 19]

grundsätzlich aus dem Aufruf der `init`, `processAction`, `render` und `destroy` Methoden.

2.2.1 Instanziierung

Der Portlet Container ist für das Laden und die Instanziierung des einzelnen Portlets verantwortlich. Der Container darf entscheiden, wann das einzelne Portlet geladen werden soll. Entweder wird das Portlet beim Beginn der Portlet Applikation geladen oder erst nachdem eine Anfrage an den Portlet Container geschickt wurde. Durch den `ClassLoader` wird das Portlet Object geladen und instanziiert, aber für die Portlet Applikation noch nicht freigegeben.

2.2.2 Initialisierung

Nachdem der Portlet Container das Portlet instanziiert hat, muss das Portlet zuerst initialisiert werden um Anfragen annehmen und ausführen zu können. Dies entspricht eigentlich nur dem Aufruf der `init` Methode. Hier kann dann das Portlet verschiedene von ihm benötigten Ressourcen initialisieren und für die Dauer des Portlets zur Verfügung stellen. Es ist aber zu beachten, dass der Portlet Container ein eindeutiges `PortletConfig` Objekt an das Portlet weitergeben muss. Innerhalb dieses Objekts

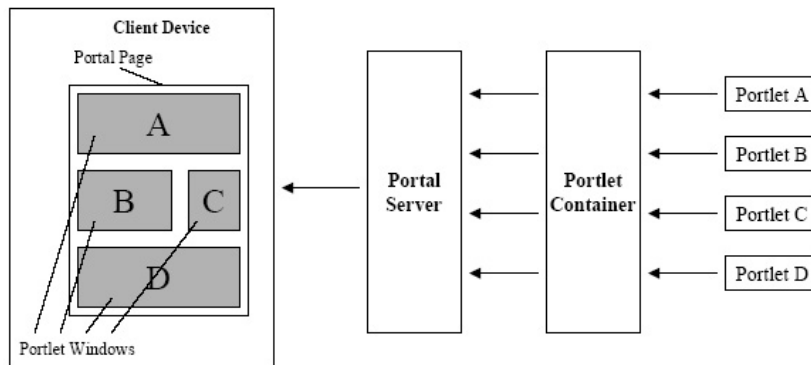


Abbildung 2.2: Generierung der Portalseiten [1, Seite 20]

befinden sich Informationen bezogen auf verwendbare Ressourcen, Initialisierungsparameter und alles andere, was in dem Portlet Deployment Descriptor definiert werden kann.

2.2.3 Portlet Window

Die Definition eines Portlets kann eine Liste von Voreinstellungen enthalten (*Portlet-Preferences*). Das Portlet passt den Aufbau der Informationen oder sein Verhalten an diese Voreinstellungen an und kann sie lesen, modifizieren oder neue Werte hinzufügen. Die Kombination eines Portlets mit dem PortletPreferences-Objekt wird als *Portlet Window* bezeichnet.

2.2.4 Anfrage-Behandlung

Nachdem ein Portlet initialisiert wurde, wird es für die Behandlung der Anfragen (Request Handling) freigegeben. Die Portlet Spezifikation unterscheidet zwischen einem `RenderRequest` und einem `ActionRequest` (genaue Beschreibung befindet sich im Abschnitt 2.5 Portlet Request). Diese verschiedenen Arten von Anfragen erlauben, dass das Verhalten eines Portlets genau gesteuert werden kann. Sie werden durch die `render` und `processAction` Methoden aufgefangen und weiter verarbeitet.

Anfragen werden durch `PortletURLs` (siehe `PortletURL`) ausgelöst. Normalerweise wird zuerst ein `ActionRequest` des betroffenen Portlets ausgelöst, danach wird von jedem der nicht betroffenen Portlets ein `RenderRequests` aufgerufen. Falls Caching eingeschaltet ist, kann der Portlet Container selbst entscheiden, ob die Notwendigkeit die `render` Methode auszulösen gegeben ist oder die benötigte Information einfach aus dem Cache gelesen und zurückgeschickt werden kann. Implementierungen der Response und Request Objekte müssen nicht Thread-sicher sein.

2 Spezifikation

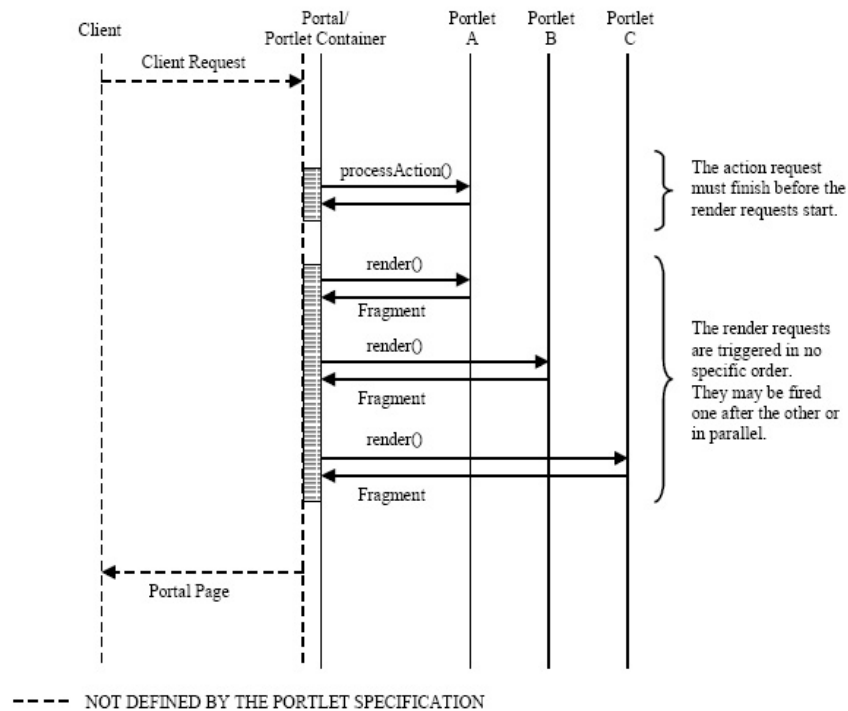


Abbildung 2.3: Portlet Request Handling Sequence [1, Seite 25]

2.2.5 Beenden

Ein Portlet Container ist nicht gezwungen, ein Portlet für einen bestimmten Zeitraum am Leben zu erhalten. Die Lebensdauer eines Portlets kann zwischen ein paar Millisekunden und einem beliebigen Zeitraum variieren. Falls ein Portlet Container entscheidet, ein Portlet beenden zu müssen, wird die `destroy` Methode des Portlets aufgerufen. Nachdem ein Portlet zerstört wurde, darf der Portlet Container keine weiteren Anfragen an dieses Portlet weiterleiten. Der Portlet Container darf nur neue Portlet Instanzen aufbauen, jedoch zerstörte Portlets nicht wiederbeleben. Nachdem ein Portlet zerstört wurde, muss es für den Garbage Collector zur Verfügung gestellt werden.

2.3 PortletURL

Ein wichtiger Aspekt der Portlets ist die Verknüpfung mit sich selbst oder anderen Portlets (z.B. Formulare oder externe Ressourcen). Dies wird, wie in HTML und HTTP üblich, via URLs erledigt. Hier stellt die Java Portlet Spezifikation das Konzept der

PortletUrls vor.

PortletUrls sind einfach neue Client Requests, die von einem URL (z.B. Link oder Formular) innerhalb eines Portlets stammen. Da diese Requests wiederum spezielle Portlet Requests sind, müssen diese URLs, die sich innerhalb eines Portlets befinden, durch ein PortletUrl Objekt erzeugt werden. Hier gibt es verschiedene PortletUrl Modes, die, basierend auf dem Requesttyp, angepasst werden müssen.

2.3.1 Portlet URL Modes

PortletUrl Objekte werden durch das `RenderResponse` Interface erzeugt. Hierbei muss der Programmierer überlegen, was für ein PortletUrl Objekt benötigt wird. Bei der Verwendung von Formularen sollte der Portletsprogrammierer nur die POST-Methode benutzen, da die GET-Methode nicht von der Portletspezifikation unterstützt wird.

- **Action URL's:** Wie schon erwähnt wurde gibt es zwei verschiedene Arten von Requests. Ein Action URL wird durch die `createActionURL` Methode erzeugt. Dies wiederum ruft die `processAction()` und nicht die `render()` Methode auf. Action URL's sollten verwendet werden, falls eine Weiterleitung an ein anderes Portlet oder eine externe Website durchgeführt werden sollte, ebenso, wenn man den PortletMode oder den WindowState ändern möchte, wenn die persistenten Daten geändert, hinzugefügt oder gelöscht, so wie die Render-Parametern gesetzt werden sollen.
- **Render URL's:** Render URL's werden durch die `createRenderURL` Methode erzeugt. Dieser URL Typ wiederum ruft die `render()` Methode von einem Portlet auf. Render URL's sollten für nicht-idempotente (nicht immer die Selben) Tätigkeiten verwendet werden, da das Portletverhalten (u.a. Error Caching, Cache Expirations) den generierten Inhalt verändern kann. Weiters sollten Render URL's nicht in Formularen verwendet werden, da es sein kann, dass der Portlet Container Formular-Parameter einfach ignoriert.

2.3.2 Portlet Parameter

PortletUrl Objekte können durch eine vom Programmierer definierte Parameterliste erweitert werden. Die Codierung der Parameter wird für den Programmierer transparent durchgeführt, muss aber vom Portlet Container als "x-www-form-urlencoded" codiert werden.

2.3.3 Portlet URL Erweiterungen

PortletUrl's können auch den PortletMode und den WindowState des Portlets kontrollieren. Um die Sicherheit zu gewährleisten, können Programmierer PortletUrl's

2 Spezifikation

entweder als *https* oder *http* URL's erstellen.

2.4 Portlet Modes

Ein Portlet Mode beschreibt die Funktion, die ein Portlet gerade ausübt, die aber ohne weiteres auch programmtechnisch geändert werden kann. Weiters können Portlet Modes auch benutzerabhängig definiert werden. Es gibt von der Spezifikation aus drei „default modes“, in der sich ein Portlet befinden kann, und die Möglichkeit eigene Modi zu definieren.

2.4.1 VIEW Mode

Die vorgeschriebene Funktionalität eines Portlets, das sich im View Mode befindet, ist Markup anzuzeigen. In diesem Fall ist Markup als HTML oder XHTML Output definiert, könnte aber in weiterer Folge auch andere Ausgabeformate unterstützen. Der Inhalt dieses Markups kann entweder als statische Information dargestellt werden oder mehrere Seiten enthalten, um Interaktion mit dem Benutzer zu ermöglichen. Der VIEW Mode muss von einem Portlet unterstützt werden.

2.4.2 EDIT Mode

Innerhalb dieses Modus kann ein Benutzer die Logik und den Inhalt bestimmen bzw. verändern. Im EDIT Modus kann der Benutzer auch auf mehreren Seiten Änderungen vornehmen. Typischerweise können innerhalb des EDIT Modus die Einstellungen des Portlets vom Benutzer verändert werden. Im Gegensatz zu dem VIEW Mode muss der EDIT Mode nicht vom Portlet unterstützt werden.

2.4.3 HELP Mode

Dieser Modus erlaubt dem Benutzer nähere Informationen über das Portlet zu erhalten. Die Information kann entweder ein statischer Text sein oder ein context-sensitiver Text. Wie beim EDIT Mode muss der HELP Mode vom Portlet nicht unterstützt werden.

2.4.4 CUSTOM Mode

Portlet Lieferanten können auch eigene Modes definieren. Dies erlaubt eine gewisse Flexibilität, die besonders für den Programmierer einen Vorteil darstellt. Diese CUSTOM Modes müssen innerhalb des Deployment Descriptors definiert werden. Die Portlet Spezifikation beschreibt einige vordefinierte CUSTOM Modes, die nicht unbedingt von einem Portlet Anbieter implementiert bzw. angeboten werden müssen. Folgende Beispiele für CUSTOM Modes sind gegeben:

- **About Mode:** Stellt Information über ein Portlet dar.
- **Config Mode:** Administrative Funktionalität für Portlets, die als „non-modifiable“ im Deployment Descriptor definiert wurden.
- **Preview Mode:** Erlaubt dem Entwickler in der Design-Phase das Portlet zu rendern, ohne dass eine Back-End Verbindung zu den Daten besteht.
- **Print Mode:** Erlaubt dem Benutzer, den Inhalt in einem druckerfreundlichen Format anzuzeigen.

Um Portlet Modes für das jeweilige Portlet zu definieren, müssen alle erlaubten Modes im Deployment Descriptor angegeben werden. Da der VIEW Mode für jedes Portlet erlaubt werden muss, wird es für die Definition nicht benützt. Die verschiedenen Modes eines Portlet werden im Deployment Descriptor folgendermaßen definiert:

```
<portlet-app>
...
  <portlet>
    ...
    <supports>
      ...
      <portlet-mode>print</portlet-mode>
    </supports>
  </portlet>
  ...
  <!-- Falls ein Custom Mode -->
  <custom-portlet-mode>
    ...
    <name>print</name>
  </custom-portlet-mode>
</porlet-app>
```

2.5 Portlet Requests

Eines der wesentlichen Bestandteile der Portlet Infrastruktur ist die Art und Weise, Client Requests zu handhaben. Requests werden als Request Objekte interpretiert, wobei alle notwendigen Informationen (u.a. Parameter, Portlet Mode, Window State, etc.) innerhalb des Objekts vorhanden sein müssen. Das Request Objekt wird zu der `processAction()` und der `render()` Methode eines Portlets geschickt.

Grundsätzlich gibt es innerhalb des Portlet Frameworks zwei Arten von Request Objekten, die wiederum vom PortletRequest Interface abgeleitet werden:

2 Spezifikation

- ActionRequests
- RenderRequests

2.5.1 Request Parameter

Falls ein Client Request existiert, der sich auf das eigene Portlet bezieht, müssen alle Request Parameter innerhalb des erstellten `PortletURL` Objekts definiert werden. Es dürfen keine `ActionRequest` Parameter an ein `RenderRequest` propagiert werden. Die `RenderRequest` Parameter können entweder durch ein `ActionRequest` mit den `setRenderParameters()` Methoden erstellt werden oder innerhalb des `RenderURLs` Objekts.

Render Parameter werden innerhalb eines `PortletRequests` als ein Name-Value Paar gespeichert. Diese können dann durch folgende Methoden verwendet werden:

- `getParameter`
- `getParameterNames`
- `getParameterValues`
- `getParameterMap`

Zusätzlich erlauben Portlets selbstdefinierte Parameter zu setzen, um den `PortletURLs` weitere container-routing oder Request-Handling-Möglichkeiten zu gewährleisten. Für die Kodierung der erweiterten Parameter ist der Portlet Container zuständig. Diese müssen mit dem Präfix `javax.portlet.` gekennzeichnet werden.

2.5.2 Request Attribute

Request Attributes erlauben dem Portletprogrammierer die Möglichkeit mit externen JSP oder Servlets zu kommunizieren. Es darf aber nur ein Wert pro Attribut gespeichert werden. Folgende Methoden werden für die Verwendung dieser Attribute bereit gestellt:

- `getAttribute`
- `getAttributes`
- `setAttribute`
- `removeAttribute`

2.5.3 Request Properties

Der Portlet Programmierer kann durch die `Request Properties` Informationen über Portlets/Portlet-Container oder sogar HTTP Headers bekommen. Der Zugriff auf HTTP Header Properties ist aber Voraussetzung der Spezifikation und kann deshalb bei verschiedenen Anbietern variieren. Mit den folgenden Methoden kann ein Portlet Programmierer Zugriff auf Portlet Properties bekommen:

- `getProperties`
- `getProperty`
- `getPropertyName`

2.5.4 Request Security

Da in allen heutigen IT-Projekten Rücksicht auf die Sicherheit genommen werden muss, gibt es portlet-spezifische Sicherheitsmaßnahmen. Innerhalb eines Portlet Request Kontext werden folgende Methoden angeboten:

- `getAuthType`
- `getRemoteUser`
- `getUserPrincipal`
- `isUserInRole`
- `isSecure`

Es gibt 4 grundlegende Authentifizierungsschemas. Diese basieren auf existierende default Typen (`BASIC_AUTH`, `DIGEST_AUTH`, `CERT_AUTH`, und `FORM_AUTH`). Die Möglichkeit anbieterspezifische Authentifizierungsmethoden implementieren zu können ist auch gegeben.

2.5.5 Weitere Eigenschaften

Die Anzahl der Erweiterungen der Portlet Requests ist zu groß um hier komplett aufgelistet zu werden. Wichtige Erweiterungen werden unten kurz beschrieben.

- **Internationalization:** Die `getLocale` Methode ist innerhalb des `PortletRequest` Objekt vorhanden und dient somit dem Portlet Programmierer die Internationalisierung des Portlets besser zu gestalten.

2 Spezifikation

- **Portlet Modes und Window States:** Durch die angebotenen Methoden `getPortletMode` und `getWindowState` ist es möglich, innerhalb eines Requests den Mode und Window State vom Portlet zu bekommen.
- **Upload Data:** Portal Programmierer können auch Dateien jeglicher Art von einem Benutzer durch die `getPortletInputStream` oder `getReader` Methode bekommen.

Der Lebenszyklus eines Portlet Requests besteht nur innerhalb des `processAction` oder `render` Rahmen. Falls ein Portlet Programmierer das `Request` Objekt ausserhalb dieses Rahmen verwenden möchte, kann es zu nicht deterministischen Nebenwirkungen kommen. Jedoch lehnt die Spezifikation dem Programmierer diese Implementierungsentscheidung ab.

2.6 Portlet Response

Das `PortletResponse` Objekt dient dazu, alle Informationen (z.B. Titel, Inhalt, Weiterleitung, Portlet Mode Änderung usw.) an den Portal-Container bzw. an ein Portlet zurück zu schicken. Dieses Objekt wird an die `processAction` oder `render` zurück gegeben. Das `PortletResponse` Objekt dient als Interface für die `ActionResponse` und `RenderResponse` Objekte.

2.6.1 Response Properties

Wie bei den `PortletRequest` Objekten kann der Programmierer eigene Eigenschaften an das Portal bzw. den Portlet-Container übergeben. Diese werden mit folgenden Methoden gesetzt:

- `setProperty`
- `addProperty`

2.6.2 Action Responses

Um konsistent mit dem Portlets Framework zu sein gibt es unterschiedliche Response Typen. Das `ActionResponse` Objekt dient dazu, um Weiterleitungen durchzuführen oder Window States zu ändern so wie Render Parameter zu setzen. Weiterleitungen werden mit der `setRedirect` Methode ausgeführt. Portlet Modes und Window States werden mit den `setPortletMode` und `setWindowState` Methoden festgelegt. Dabei muss beachtet werden in welcher Reihenfolge die verschiedenen Methoden aufgerufen werden, ansonsten wird eine Exception ausgelöst.

2.6.3 Render Responses

Dieser Response Typ erlaubt dem Portlet Programmierer den Portlet Titel so wie den Portlet Inhalt zu gestalten. Dabei muss sichergestellt werden, dass die `setContentype` Methode aufgerufen wird, um den Inhaltstyp zu definieren.

Außerdem ist es möglich, jegliche Information an den Client zu schicken. Durch die Verwendung von entweder einem `OutputStream` oder einem `Writer` Objekt können verschiedene Inhalte an den Client geschickt werden. Der Response kann zuerst in einen Buffer (dies ist dem Programmierer überlassen, wird aber aus Effizienzgründen bevorzugt) geschrieben werden, wobei folgende Methoden zur Verfügung stehen:

- `getBufferSize`
- `setBufferSize`
- `isCommitted`
- `reset`
- `resetBuffer`
- `flushBuffer`

Die Buffergröße kann vom Programmierer festgelegt werden, wobei der Buffer den Inhalt an den Client schickt, entweder wenn er voll ist oder wenn die `flushBuffer` Methode aufgerufen wird.

2.6.4 URL Encoding

Da Portlets mit JSP oder Servlets erweitert werden können, ist es möglich, verschiedene implementations-spezifische Daten an die verschiedenen Komponenten zu schicken. Dies wird durch die `encodeURL` Methode verwirklicht. Dadurch ist es möglich, Session ID's oder Portlet spezifische Daten an Servlets oder den JSPs zu übergeben.

2.7 Packaging und Deploying

2.7.1 Packaging

Portlets sind vollständige Applikationen, die ohne zusätzliche Ressourcen lauffähig sein sollen. Deswegen werden alle Ressourcen in einem Web Application Archive (WAR) gespeichert, aus dem der Portal-Server alle benötigten Konfigurationen und den Quellcode erhält.

Das Web Application Archive ist genauso aufgebaut wie in der Servlet-Spezifikation [2] beschrieben:

2 Spezifikation

- einzelne Ressourcen, wie JSPs und Bilder
- ein WEB-INF Verzeichnis mit
 - web.xml
 - lib-Verzeichnis mit zusätzlichen JARs
 - classes-Verzeichnis mit den Servlet und Portlet-Klassen

Wie schon in den vorigen Kapiteln häufig erwähnt, gibt es zusätzlich zur Konfigurationsdatei web.xml auch noch die Konfigurationsdatei portlet.xml. Sie heißt Portlet Deployment Descriptor und enthält nur die portletspezifischen Einstellungen.

2.7.2 Portlet Deployment Descriptor

Der Portlet Deployment Descriptor `portlet.xml` kann zwei Arten von Definitionen enthalten:

- Definitionen der gesamten Portalanwendung
- Definitionen der einzelnen Portlets

Zu den Portal Definitionen gehören einige schon besprochene Konfigurationsdaten, wie `custom-portlet-mode`, `user-attribute` und `security-constraint`. Neben diesen Einstellungen, die für das gesamte Portal gültig sind, gibt es Einstellungen für ein oder mehrere Portlets, die ebenfalls teilweise schon besprochen wurden: `portlet-name`, `init-params` oder auch `portlet-preferences`.

Der Portlet Deployment Descriptor unterstützt auch Localization, so können beispielsweise unterschiedliche Sprachen für die Beschreibung von Portlets verwendet werden:

```
<display-name xml:lang="en">Time Zone Clock Portlet</display-name>
<display-name xml:lang="de">Zeitzone-Portlet</display-name>
```

Der Einfachheit halber können diese Informationen auch in eigenen Resource-Bundles abgespeichert werden, die den üblichen Aufbau von Java-Properties haben.

2.8 PortletConfig

Das Interface `PortletConfig` stellt Methoden zur Verfügung, um ein Portlet-Objekt konfigurieren zu können. Es ist ähnlich aufgebaut, wie das Interface `ServletConfig` der Servlet-Spezifikation.

Die Methode `getPortletContext` gibt den `PortletContext` der Portlet-Anwendung zurück.

Mit den Methoden `getInitParameterNames` und `getInitParameter` kann auf Initialisierungsparameter, die im Deployment-Deskriptor angegeben werden können, zugegriffen werden. Die Methode `getPortletName` gibt den Namen des Portlets zurück, der im Deployment-Deskriptor über den `portlet-name` eingestellt werden kann.

Weiters kann auf `ResourceBundles` zugegriffen werden, die im Deployment Deskriptor definiert werden können (siehe Abschnitt 2.7).

2.9 PortletContext

Das Interface `PortletContext` definiert die Sicht eines Portlets auf die Portal-Anwendung, in der es läuft. Es bietet Methoden an, um Ereignisse zu protokollieren, um auf anwendungsweite Ressourcen zuzugreifen und um Attribute zu lesen und zu setzen, auf die andere Portlets und Servlets der Anwendung zugreifen können. Für jede Portalanwendung existiert genau ein `PortletContext`-Objekt.

Da eine Portal-Anwendung eine erweiterte Webanwendung ist, hat auch jede Portal-Anwendung einen `ServletContext`. Intern leitet der `PortletContext` manche Anfragen direkt an den `ServletContext` weiter, denn alle Servlets und Portlets teilen sich die kontextübergreifende Parameter. Beispielsweise müssen die in der `web.xml` definierten Initialisierungsparameter über den `PortletContext` die gleichen Werte zurückliefern, wie über den `ServletContext`. Das gleiche gilt auch für Kontextattribute. Dadurch ist es möglich, Daten zwischen Portlets und Servlets bzw. JSPs zu teilen.

2.10 PortletPreferences

Üblicherweise können Benutzer Portlets an ihre eigenen Bedürfnisse anpassen und diese Einstellungen speichern. Bei einem Wetter-Portlet wäre es z.B. wünschenswert die Einheit der Temperatur, also z.B. „F“ für Fahrenheit oder „C“ für Celsius, pro Benutzer speichern zu können.

Diese Funktionalität übernimmt das Interface `PortletPreferences`. Es kann einfache Konfigurationsdaten speichern, ist jedoch nicht als Ersatz für komplexere Einstellungen gedacht, die nach wie vor in Datenbanken gespeichert werden sollen. Die Speicherung erfolgt über Schlüssel-Wert Paare, wobei sowohl der Schlüssel als auch der Wert vom Typ `String` sein muss. Es können jedoch einem Schlüssel auch mehrere Werte zugeordnet werden. Weiters kann auf Werte auch nur eine lesende Zugriffsmöglichkeit eingestellt werden.

Zugriffsmethoden sind u.a. `getNames`, `getValue`, `setValue`, `getValues`, `setValues`, die die Namen der Schlüssel und den (ersten) Wert bzw. die Werte des Schlüssels zurückliefern.

2 Spezifikation

Die Methode `reset` kann dazu verwendet werden, die Einstellungen auf den Standard-Wert zurückzusetzen. Die Methode `store` speichert das Preferences-Objekt transaktionssicher.

Folgendes Listing zeigt einen möglichen Eintrag von Einstellungen im Deployment Deskriptor:

```
<portlet>
...
  <portlet-preferences>
    <preference>
      <name>units</name>
      <value>C</value>
      <value>F</value>
    </preference>
  </portlet-preferences>
</portlet>
```

Auf die Einträge kann in folgender Weise zugegriffen werden:

```
PortletPreferences prefs = req.getPreferences();
String[] units = prefs.getValues("units", null);
```

2.11 PortletSession

Um sinnvolle Portale bauen zu können, müssen die einzelnen Anfragen eines Clients als zusammengehörig erkannt werden. Um dies zu erreichen, kann das Interface `PortletSession` verwendet werden, das unabhängig von der Implementierung eine Session-Funktionalität zur Verfügung stellt. Jede Portal-Anwendung hat pro User ein eigenes Session-Objekt, das dieses Interface realisiert.

Eine `PortletSession` kennt zwei unterschiedliche Gültigkeitsbereiche:

- Application Scope
- Portlet Scope

Auf ein Objekt, das den Gültigkeitsbereich *Application Scope* besitzt, kann jedes Portlet in der Anwendung zugreifen. Auf Objekte mit dem Gültigkeitsbereich *Portlet Scope* kann nur das Portlet zugreifen, das es auch erstellt hat. Intern wird so ein Objekt allerdings ebenfalls im *Application Scope* gespeichert, nur mit einem speziellen Namensraum.

Das Setzen von Attributen demonstriert folgendes Beispiel:

```

PortletSession session = request.getSession(true);
URL url = new URL("http://www.foo.com");
session.setAttribute("home.url",url,PortletSession.APPLICATION_SCOPE);
session.setAttribute("bkg.color","RED",PortletSession.PORTLET_SCOPE);

```

Weiters hat eine `PortletSession` viele Gemeinsamkeiten mit der `HttpSession` der Servlet-Spezifikation: die von einem Portlet gespeicherten Objekte sind auch für Servlets sichtbar und umgekehrt. Methoden wie `getId`, `invalidate` und `isNew` müssen die selbe Funktionalität wie bei Servlets besitzen.

2.12 UserInformation

Ein grundlegendes Konzept eines Portals ist die Anpassung der Oberfläche und des Inhalts an den Benutzer. Um entsprechende Anpassungen vornehmen zu können, muss die Anwendung Informationen über den Benutzer speichern können. Die Portlet-Spezifikation sieht für diesen Zweck einen Mechanismus vor, mit dem ein Portlet auf Benutzer-Informationen zugreifen kann.

Diese Informationen können in sogenannten Benutzer-Attributen gespeichert werden. Diese werden im Deployment Deskriptor definiert. Folgendes Listing zeigt eine Konfigurationsbeispiel:

```

<portlet-app>
  ...
  <user-attribute>
    <description>Vorname</description>
    <name>user.name.given</name>
  </user-attribute>
  <user-attribute>
    <description>Nachname</description>
    <name>user.name.family</name>
  </user-attribute>
  <user-attribute>
    <description>eMail-Adresse</description>
    <name>user.home-info.online.email</name>
  </user-attribute>
  ...
</portlet-app>

```

Die Namen der Attribute, wie `user.name.given`, können freigewählt werden, die Spezifikation empfiehlt aber die Verwendung vorgegebener Namen für häufig benötigte

2 Spezifikation

Daten, um eine einheitliche Bezeichnung und Semantik zu gewährleisten. Die vorgegebenen Namen orientieren sich an der P3P-Spezifikation des W3Cs [3].

Um von einem Portlet auf die Benutzer-Attribute zuzugreifen, wird ein Objekt vom Typ `Map` verwendet, das als Attribut im Request gespeichert ist. Das folgende Beispiel zeigt den Zugriff auf den im Deployment Deskriptor definierten Wert mit dem Vornamen:

```
Map userInfo = (Map) request.getAttribute(PortletRequest.USER_INFO);
String givenName = (userInfo!=null) ?
    (String) userInfo.get("user.name.given") : "";
```

2.13 PortletCaching

Um die Antwortzeit eines Portals zu erhöhen, können Caching-Mechanismen verwendet werden. Die Spezifikation sieht einen Mechanismus vor, der auf einem Verfallsdatum basiert und eine Zwischenspeicherung pro Portlet und Client vorsieht. Diese Zeitangabe wird in Sekunden angegeben und kann im Deployment Deskriptor angegeben oder im Programm selbst zur Laufzeit verändert werden.

2.14 PortletSecurity

Ein Portalsystem enthält Ressourcen, auf die durch viele User zugegriffen werden kann. In offenen Netzwerken wie dem Internet müssen deswegen Sicherheitsanforderungen beachtet werden.

Der Portlet Container muss die Portlets informieren, welche Rolle ein zugreifender User einnimmt. Die Authentifizierung übernimmt aber nicht der Portlet Container sondern der Servlet Container, wie es in der Servlet Spezifikation 2.3 definiert ist. Das Request-Interface besitzt die selben Methoden-Signaturen wie das ServletRequest-Interface: `getRemoteUser`, `isUserInRole` und `getUserPrincipal`. Diese Methoden haben auch die selbe Bedeutung wie bei Servlets [2].

Für die Verbindung von Portlets mit Enterprise JavaBeans existiert eine Erweiterung, die das Aufrufen von Beans ermöglicht, auch wenn der User dem Portal-System nicht bekannt ist. Dafür kann die Identifizierung an die EJB-Ebene weitergereicht werden.

2.14.1 Security Constraints

Ähnlich wie bei Servlets können auch Portlets bestimmte Einschränkungen auferlegt werden. Diese beziehen sich aber nicht wie bei Servlets auf URL-Pattern und HTTP-Methoden sondern auf die einzelnen Portlets. Es können Anforderungen bezüglich Inte-

grität und Vertraulichkeit gestellt werden, die beispielsweise durch SSL-Verbindungen erfüllt werden müssen.

Ausschnitt eines Deployment-Deskriptors mit Security Constraints, bei dem das Portlet „accountSummary“ sicher dargestellt werden muss:

```
<portlet-app>
  ...
  <portlet>
    <portlet-name>accountSummary</portlet-name>
    ...
  </portlet>
  ...
  <security-constraint>
    <display-name>Secure Portlets</display-name>
    <portlet-collection>
      <portlet-name>accountSummary</portlet-name>
    </portlet-collection>
    <user-data-constraint/>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
  ...
</portlet-app>
```

2.15 TagLibs

Tag Libraries werden eingesetzt um die Programmierung einer JSP zu vereinfachen. Ein Portlet-Container muss 5 Tags implementieren um die Spezifikation zu erfüllen. Diese sollen den Zugriff auf portletspezifische Elemente und Funktionalität ermöglichen:

- **defineObjects** Tag: ermöglicht in der JSP den Zugriff auf die Variablen: `renderRequest`, `renderResponse` und `portletConfig`.
- **actionURL** Tag: erzeugt einen URL, der auf das aktuelle Portlet zeigt und einen *action request* auslöst. Es können Attribute angegeben werden, die u.a. den `WindowState` und `PortletMode` setzen.
- **renderURL** Tag: erzeugt einen URL, der auf das aktuelle Portlet zeigt und einen *render request* auslöst. Es können ebenfalls Attribute für den `WindowState` und `PortletMode` gesetzt werden.

2 Spezifikation

- **namespace** Tag: erzeugt einen eindeutigen Wert für das aktuelle Portlet. Es kann verwendet werden, um z.B. JavaScript-Funktionen eindeutig zu benennen, wodurch keine Namenskonflikte mehr auftreten können.
- **param** Tag: definiert Parameter, die einem **ActionURL** oder *RenderURL* hinzugefügt werden können.

3 Integration

Die Portlet Architektur wurde so spezifiziert, dass die Erweiterbarkeit und Integration von existierenden Frameworks auch ermöglicht wird. Insbesondere werden wir hier Integrationsfähigkeiten des Portlet Frameworks mit existierenden Technologien, so wie JSP, Servlets, Web Services u.ä. präsentieren.

3.1 Servlet Integration

Da das Portlet Framework mit der Java Programmiersprache entwickelt wurde, kann daraus abgeleitet werden, dass Java Portlets durchaus mit anderen auf Java-basierenden Architekturen interagieren kann. Portlets basieren auf die Servlet 2.3 Spezifikation. Trotz der Ähnlichkeiten gibt es eindeutige Unterschiede zwischen diesen Technologien, die eine eigene Spezifikation notwendig gemacht haben [1].

Ähnlichkeiten	Unterschiede
Java basierende Technologie	Portlets erzeugen nur „markup fragments“
Container Verwaltung	Portlets sind nicht direkt an ein URL gebunden
Generierung dynamischer Inhalte	Mehrere Portlets können auf einer Portal Seite existieren
Lifecycle von Container verwaltet	Clients interagieren mit Portlets durch ein Portal System
Interaktion durch Request-Response-Paradigma	Portlets haben raffiniertere Request Handhabung
J2EE Integration	Vordefinierte Portletmodes

Tabelle 3.1: Servlet vs. Portlet Spezifikation

Zusätzlich gibt es weitere Funktionalitäten die von der Portlets Umgebung unterstützt werden:

- Portlets haben die Zugriff- und Speicherungsmöglichkeiten persistenter Konfiguration und Anpassungsdaten
- Portlets haben Zugriff auf Benutzerprofildaten

3 Integration

- Portlets haben die Möglichkeit URLs Portal Server agnostisch zu erstellen
- Portlets können transiente Daten innerhalb 2 verschiedener Scopes speichern: die "Application-Wide" Scope und der "Portlet-Private" Scope

Durch die Verwendung eines `PortletRequestDispatcher` Objekts innerhalb der Ausführung der `render()` Methode kann der Portletprogrammierer Inhalte erzeugen und an ein Servlet oder eine JSP Seite weiterleiten.

3.2 JSP Integration

Durch die starke Verbreitung der Java Server Pages Technologie wird sowohl in der Java Portlets Spezifikation als auch in anderen kommerziellen Portlet-unterstützten Produkten (w.z.B. BEA) die Zugriffsmöglichkeiten auf Portlets durch JSP Tags angeboten.

Um innerhalb einer JSP Seite Zugriff auf ein Portlet haben zu können, muss der JSP Entwickler die korrekte Tag Library deklarieren:

```
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>
```

Somit wird die Tag Library spezifisch für die Integration von Portlets definiert. Dadurch hat der JSP Entwickler die Möglichkeit, verschiedene Portletobjekte zu verwenden. Um diese Objekte verwenden zu können, muss das `defineObjects` tag die `RenderRequest`, `RenderResponse`, und `PortletConfig` Objekte definieren. Folgendes Beispiel würde den Titel eines Portlets innerhalb einer JSP Seite definieren:

```
<portlet:defineObjects/>  
<%=renderResponse.setTitle("My New Portlet Title!")%>
```

3.2.1 ActionURL

Weiters ist es möglich innerhalb JSP ein bestimmtes `actionURL` Objekt zu erzeugen, das dem JSP Entwickler erlaubt, verschiedene Aktionen einem Portlet zu übergeben bzw. ein `ActionRequest` auszulösen. Folgende Parameter können zu einem `actionURL` hinzugefügt werden:

- `windowState`
- `portletMode`
- `var`
- `secure`

3.3 Web Service Remote Portlet Integration

Mittels `<portlet:param/>` tag ist es auch möglich, eigene Parameter an ein `actionURL` hinzuzufügen. Ein Beispiel, welches dem Portlet in den EDIT Modus versetzt und den Windowstate auf MAXIMIZED setzt, würde folgendermaßen aussehen:

```
<portlet:actionURL windowState="maximized" portletMode="edit">
  <portlet:param name="action" value="editStocks"/>
</portlet:actionURL>
```

3.2.2 RenderURL

Da Portlets nicht nur `ActionRequests` unterstützen sondern auch `RenderRequests`, ist es auch möglich, `renderURL` innerhalb einer JSP Seite zu verwenden. Diese lösen wiederum `RenderRequests` für ein gegebenes Portlet aus. Ähnlich wie bei `actionURLs` können folgende Parameter gesetzt werden:

- `windowState`
- `portletMode`
- `var`
- `secure`

Folgendes Beispiel würde einen URL erzeugen, auf welchem die Aktienwerte zweier Firmen angezeigt werden:

```
<portlet:renderURL portletMode="view" windowState="normal">
  <portlet:param name="showQuote" value="myCompany"/>
  <portlet:param name="showQuote" value="someOtherCompany"/>
</portlet:renderURL>
```

3.3 Web Service Remote Portlet Integration

Das Konzept (und der damit verbundene Hype) von Web Services gibt es schon seit einiger Zeit. Web Services sind im Grunde genommen plattform-unabhängige, datenorientierte, übers web-zugreifende Remote Objects. Die Web Service Remote Portlet Spezifikation dient als Lösung für einen presentations orientierten Web Service Ansatz.

Web Services dienen dem Konsumenten von solchen Services nur für datenorientierte Zwecke. Diese Web Services müssten dann wiederum mit einer vom Konsumenten selbst erstellten Präsentationsschicht versehen werden. Weiter gedacht, müssen Applikationsaggregatoren mit einer eindeutigen Schnittstelle kommunizieren [7, Seite 7]. Eine Plug-and-Play Lösung für die dynamische Integration von Geschäftsapplikationen oder Inhalte wurde durch die Web Service for Remote Portlets Spezifikation vorgeschlagen. In Abbildung 3.1 wird die Architektur des WSRP Ansatzes dargestellt.

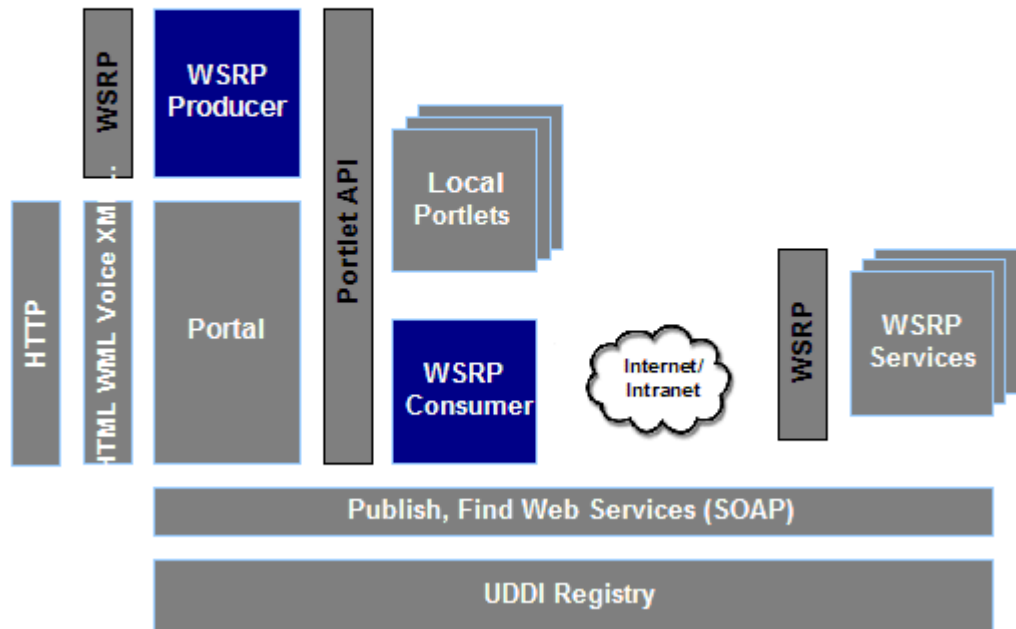


Abbildung 3.1: WSRP Architektur

3.3.1 Prozessablauf

1. Konsument eines WSRP identifiziert, bzw. entdeckt einen Produzenten. Der Konsument findet die URL für den Web-Service Endpoint und eruiert die Metadaten, inkl. Beschreibung des Produzenten, mögliche Registrierungsanforderungen und möglicherweise eine Liste der veröffentlichten Portlets.
2. Etablierung einer Beziehung zwischen Konsument und Produzent. Dies kann auf geschäftlicher oder technischer Ebene fungieren wie auf Sicherheitsanforderungen.
3. Etablierung einer Beziehung zwischen Konsument und Benutzer. Dies erlaubt dem Benutzer den möglichen Zugriff auf den vom Produzenten veröffentlichten Portlet. Der Konsument könnte sogar dem Benutzer die Möglichkeit erlauben eigene Anpassungen der Portlets vorzunehmen.
4. Produktion der aggregierten Seiten. Dieser Prozessschritt wird normalerweise nach der Basisdefinition des Pagelayouts vom Benutzer durchgeführt.

3.3 *Web Service Remote Portlet Integration*

5. Abfrage einer Seite.
6. Benutzerinteraktionen verarbeiten. Manche Benutzerinteraktionen führen zu Abwicklungen einer bestimmten Logik innerhalb der konsumentendefinierten Seiten. Der Konsument führt diese Logik aus, was wiederum einen Einfluss auf andere Portlets haben könnte und somit den vorherigen Schritt auslöst.
7. Beendigung der Beziehung zwischen Produzent und Konsument kann zu jeder Zeit durchgeführt werden, wobei die benutzten Ressourcen danach freigegeben werden können.

Obwohl die Spezifikation schon als Standard akzeptiert wurde, gibt es offiziell noch sehr geringe Unterstützung innerhalb der IT Gemeinschaft. Die Apache Gruppe hat das so genannte WSRP4J (Web Service for Remote Portlets for Java) Framework als Inkubator Projekt, welches wiederum von IBM gestartet und weiterhin unterstützt wird, begonnen [4].

3 *Integration*

4 Systemumgebung

Schon seit einigen Jahren werden Portale verwendet um einen zentralen Zugang zu individuell zugeschnittenen, unternehmensinternen und -externen Informationen und Diensten zu ermöglichen. Diese Integration ist jedoch sehr kostspielig und auch nicht immer durch Erfolg gekrönt. Eine Forrester-Umfrage vom April 2004 unter Angestellten zeigte, dass nur knapp die Hälfte der Angestellten aus dem Webangebot der Firma einen Nutzen gewinnen können [10]. In einer anderen Studie sieht Forrester-Research den Grund des Versagens von Portalen unter anderem in der mangelnden Unterstützung von Standards, wie JSR 168 und WSRP [6].

Heute, ungefähr ein Jahr nachdem die erste Version der JSR 168 veröffentlicht wurde und etwas mehr als ein halbes Jahr nach der Veröffentlichung der finalen Version, unterstützen 7 der 9 größten Anbietern von Portal-Systemen JSR 168 und WSRP:

- BEA Weblogic Portal 8.1
- IBM WebSphere Portal 5.0
- Oracle AS 10g Portal
- Plumtree Enterprise Websuite
- Sun Java System Portal Server 6.2
- Sybase Enterprise Portal 6.0
- Vignette Application Portal 7.0 and Application Builder 4.5

NetworkComputing hat einen Vergleich dieser 9 Anbieter durchgeführt [8], wobei lediglich zwei Anbieter (Microsoft und Tibco) kein zu JSR 168 und WSRP standardkonformes Produkt vertreiben. Nach dem Vergleich landete Oracle 10g Portal auf Platz eins, gefolgt von Sybase Enterprise Portal und Plumtree Enterprise Web Suite. Die Beschreibung der folgenden kommerziellen Anbietern ist eine Kurzfassung dieses Vergleichs.

4.1 Kommerzielle Anbieter

Dieses Kapitel zeigt eine Übersicht einiger Portal-Produkte. Portalsysteme können zu-
meist in zwei Kategorien eingeteilt werden: entweder sie basieren auf einer Infrastruk-
tur, die Enterprise Application Integration zum Bestandteil hat, oder sie verwenden
eine Portal-Standard-Software, die als eigenständige Anwendung läuft.

4.1.1 OracleAS 10g Portal

Das Oracle-Portal ist eines von wenigen Systemen, für die keine Zeile Code geschrieben
werden muss, um es in eine vorhandene Struktur zu integrieren. Neben der obligaten
Unterstützung von JSR 168 unterstützt das Portalsystem von Oracle als Datenquellen
SQL-Datenbanken, XML, HTML und Web Services. Mail-Integration funktioniert mit
Lotus Notes und Microsoft Exchange, zur Benutzerverwaltung kann LDAP und Active
Directory verwendet werden. Weiters bietet es ein visuelles Layout-Werkzeug für die
einzelnen Seiten. Das Oracle-Portal gewinnt auch durch die breite Unterstützung und
Integrationsmöglichkeit mit Produkten von Oracle und anderen Anbietern.

4.1.2 Sybase Enterprise Portal 6.0

Das Portalsystem von Sybase besticht durch gute Integrationsmöglichkeiten. Es läuft
auf jedem zertifiziertem J2EE-Webcontainer, die unterstützten Datenquellen sind
SQL, XML, HTML und Web Services; Metadaten können in LDAP, Oracle und Sybase
Systemen gespeichert werden.

4.1.3 Sun Microsystems Java Enterprise Solution

Die Portalsystem von Sun verwendet den „ONE Directory Server“ und kann auf Ap-
plikationsservern von IBM, BEA und natürlich Sun eingesetzt werden. Die gesamte
Konfiguration kann über ein webbasiertes Tool vorgenommen werden. Ein visuelles
Werkzeug um einzelne Seiten zu erstellen existiert jedoch nicht. Das Portal unterstützt
Instant Messaging-Tools, Authentifizierung über LDAP, Radius, SAML und Unix.
Portlets lassen sich einfach auf User und Clients anpassen, ein eigenes Werkzeug sorgt
für eine einfache Integration von Webservices.

4.1.4 IBM WebSphere Portal 5.0

Das IBM Produkt verfügt von Haus aus über weniger vorgefertigte Portlets, dafür fin-
det man darunter auch Portlets für das Dokumentmanagement von Microsoft Office-
Dokumenten. Als Mail-Systeme können Lotus Notes, Exchange und IMAP/POP3 inte-
griert werden, für Metadaten können u.a Produkte von Microsoft, Oracle und natürlich

IBM eingesetzt werden. Um neue Portlets einfacher entwickeln zu können, integriert IBM die Portal Factory von Bowstreet.

4.1.5 BEA Systems WebLogic Portal 8.1

Die Software von BEA unterstützt den Import von Text, HTML und XML. Authentifizierung kann über Active Directory und LDAP abgewickelt werden, Metadaten können in beliebige RDBMS gespeichert werden. Das enthaltene Entwicklungskit eignet sich jedoch nicht, um in kurzer Zeit Portlets zu integrieren, da die Integration und die Wiederverwendung von Portlets nicht einfach zu administrieren ist.

4.2 Freie Anbieter

4.2.1 Jakarta Pluto

Pluto, ein Subprojekt des Apache Portal Projekts, ist die Referenzimplementierung der Java Portlets Spezifikation JSR 168. Die aktuelle Version 1.0 ist keine vollständige Implementierung eines Portals sondern nur ein Portlet-Container, der der Spezifikation JSR 168 genügt. Aufgrund dieser Sonderstellung als Referenzimplementierung ist das Projekt im Kapitel 5 genauer beschrieben.

4.2.2 Jetspeed

Jetspeed ist eine Open Source-Implementierung eines Enterprise Information Portals der Apache Software Foundation, basierend auf Java und XML mit dem Ziel Netzwerk-Ressourcen Endbenutzern zugänglich zu machen. Dabei ist Zugriff über Webbrowser, WAP-Handy oder andere Geräte möglich. Jetspeed-2 ist konform zur Java Portlet-Spezifikation. Viele externe Datenquellen, wie XML, RSS oder SMTP können integriert werden. Weiters werden Content Publication Frameworks, wie Cocoon, Web-Macro oder Velocity unterstützt. Das Projekt ist in aktiver Entwicklung und hat sich zum Ziel gesetzt, Jetspeed zu einem Werkzeug für Portal-Entwickler und User Interface-Designer zu machen. Mit Jetspeed soll die Erstellung eines XML basierten Portals stark vereinfacht werden.

4.3 Empfehlung

Neben den erwähnten großen Anbietern von Portal Systemen gibt es natürlich auch noch eine große Anzahl an kleinen Firmen, die JSR 168-konforme Portal Systeme entwickelt haben. Es kann angenommen werden, dass durch Open Source-Projekte, wie Jetspeed von Apache, in Zukunft eine große Anzahl an freien und kommerziellen Implementierungen entstehen werden, die auf vorhandenen Implementierungen aufbauen.

4 Systemumgebung

Die schon erwähnte Forrester-Studie [10] zeigt jedoch dass nach wie vor viele Portale - unabhängig von den oft eingesetzten Produkten- nicht erfolgreich sind. Folgende häufige Gründe konnten identifiziert werden (nach einer Zusammenfassung in [9]):

- Keine Unterstützung der Standards JSR 168 und WSRP
- Unzureichende Budget-Planung in frühen Phasen der Einführung
- Aus einer zu großen Auswahl resultieren zu viele inkompatible Produkte
- Schwache Übereinstimmung mit den Geschäftszielen

Da 54% der IT-Entscheidungsträger angeben haben, im Jahr 2004 neue Portal-Technologie einsetzen zu wollen (an zweiter Stelle hinter Sicherheits-Technologie), gibt Forrester folgende Empfehlungen:

- Die Stärken und Schwächen der gewählten Portal-Lösung sollten genau studiert werden
- Die vorhandene Infrastruktur sollte gut etabliert sein
- Neue Projekte sollten nicht unternehmensweit sondern nur in bestimmten Abteilungen durchgeführt werden
- Die Portal-Entwickler sollten System-Integratoren und Entwicklungspartner der entsprechenden Abteilungen konsultieren

5 Case Study

Dieses Kapitel präsentiert die Referenzimplementierung des Java Portlet Standards im Detail. Es wird ein Überblick über das Produkt gegeben, die Architektur kurz beschrieben und anhand von Screenshots das Konzept eines Portal-Containers erläutert.

5.1 Überblick

Jakarta Pluto ist die Referenzimplementierung der Java Portlet Spezifikation. Die aktuelle Version 1.0 ist jedoch keine vollständige Implementierung eines Portals sondern nur ein Portlet-Container, der der Spezifikation JSR 168 genügt. Der Portlet Container stellt eine Laufzeitumgebung für Portlets zur Verfügung und benötigt seinerseits einen Servlet-Container und verwendet dessen Ressourcen und Funktionalität. Pluto implementiert die Portlet API und gibt somit Portlet-Entwicklern die Möglichkeit ihre Portlets zu testen. Die Implementierung ist jedoch einfach und erfüllt nur jene Anforderungen, die die Spezifikation ausdrücklich nennt. Neben dem Portlet-Container gibt es auch eine ebenfalls auf das Wesentliche beschränkte Implementierung eines Portal Treibers, also dem eigentlichen Portal, in dem der Pluto Portlet Container getestet werden kann. Ein weiterer Teil des Projekts ist eine Test Suite, die alle wichtigen Funktionen eines Portlet-Containers testet. Diese Portlets können somit auch verwendet werden, um andere Implementierungen von Portal Servern zu testen.

5.2 Installation und Konfiguration

Pluto verwendet das Projektmanagement-Tool Apache Maven. Neben diesem wird Java 1.3 oder höher, sowie Tomcat 4.1 mit der Servlet API 2.3 zur Ausführung benötigt. Durch Maven kann der Build-Prozess und das Deployment in einem einzigen Schritt einfach ausgeführt werden. Das Resultat ist eine lauffähiges Portal mit installierter Portal Test Suite.

Um Portlets verwenden zu können, müssen sie in der Datei `portletentityregistry.xml` registriert werden. Weiters kann in der Datei `pageregistry.xml` das Portal Layout definiert werden. Das Standard-Layout von Pluto zeigt Abbildung [5.1](#)

5 Case Study

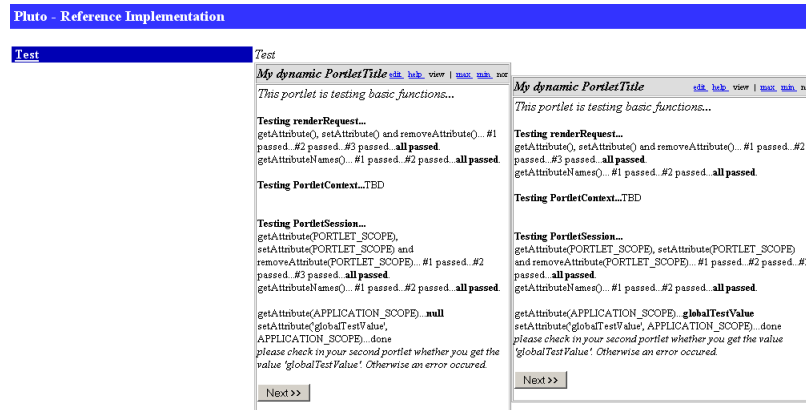


Abbildung 5.1: Pluto Standard Layout

5.3 Architektur

Die Abbildung 5.2 zeigt die Basis-Architektur des Pluto Portals: die *Portal Web Application* verarbeitet den Client-Request, in dem sie den Portlet Container aufruft, der den Inhalt der Portlets darstellt. Das Portal greift auf den Container über die *Portlet Container Invoker API* zu, eine Schnittstelle des Containers aus der Sicht des Portals. Damit der Container Informationen über das Portal erhalten kann, muss es das *Service Provider Interface (SPI)* implementieren. Letztlich ruft der Portlet Container das Portal über die *Portlet API* auf.

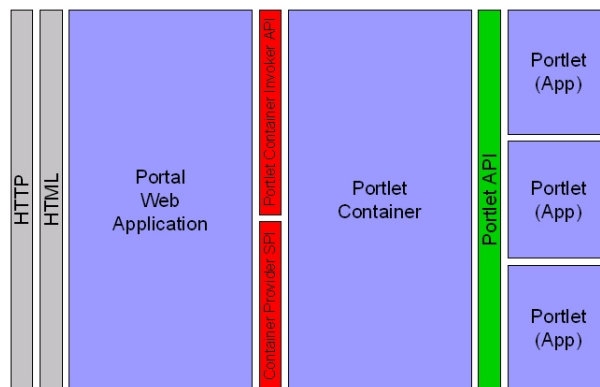


Abbildung 5.2: Pluto Portal Architektur [5]

5.4 Test Suite

Das Pluto-Projekt enthält zwei einfache Portlets, die miteinander zu Testzwecken verwendet werden können. Die Abbildung 5.1 zeigte bereits die Startseite dieser Test-Portlets. Abbildung 5.3 zeigt im Detail, wie man in unterschiedliche *Portlet Modes* und *Window States* wechseln kann.



Abbildung 5.3: Pluto Portlet Modes

Wählt man beispielsweise im linken Portlet den Link *min* wird das Portlet minimiert und das rechte Portlet nimmt den freigewordenen Platz ein (siehe Abbildung 5.4).

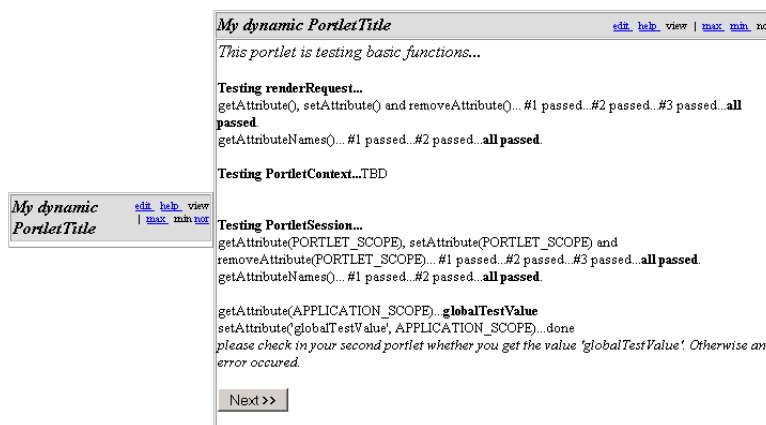


Abbildung 5.4: Pluto Portlet minimiert

Der Inhalt der Portlets zeigt die Testroutinen, dabei werden neben den erläuterten Portlet Modes die einzelnen Methoden der Portlet API, wie z.B. `getAttribute()` und `setAttribute()` oder Zugriffsmethoden der portalübergreifende Attribute getestet.

5 *Case Study*

6 Fazit

In der existierenden IT-Landschaft ist die Java Servlets-Technologie schon sehr stark vertreten. Da die Portlet-Spezifikation auf dem Servlet-Ansatz basiert und ihn erweitert, ist die Einarbeitung und Entwicklung nicht besonders aufwändig. Wer schon mit Servlets gearbeitet hat, kann nach dem Erlernen einiger grundsätzlicher Portal-Konzepte schnell Portale bzw. Portlets erstellen. Die Spezifikation wurde durch eine Zusammenarbeit von allen namhaften Portalanbietern erarbeitet, dadurch war es innerhalb kürzester Zeit möglich, vorhandene Produkte in standardkonforme Implementierungen zu erweitern. Viele kommerzielle Produkte unterstützen bereits die Entwicklung der Portale durch automatisierte Prozesse und visuelle Werkzeuge.

Java Portlets sind wiederverwendbare Komponenten, die entweder durch interne oder externe Anbieter entwickelt werden und in Portalsysteme integriert werden können. Zusätzlich können Portlets als Webservices angeboten werden, was wiederum die Erstellung einer Portalseite effizienter gestaltet. Diese Flexibilität macht in der Kombination mit der Standardisierung Java Portlets zu einer zukunftssträchtigen, nachhaltigen Technologie.

6 *Fazit*

Literaturverzeichnis

- [1] ABDELNUR, ALEJANDRO und STEFAN HEPPER: *Java Portlet Specification*, 2003.
- [2] COWARD, DANNY: *Java Servlets Specification, Version 2.3*, 2000.
- [3] CRANOR, LORRIE, MARC LANGHEINRICH, MASSIMO MARCHIORI, MARTIN PRESLER-MARSHALL und JOSEPH REAGLE: *The Platform for Privacy Preferences 1.0 Specification*, 2002.
- [4] FOUNDATION, THE APACHE SOFTWARE: *WSRP Architecture*.
<http://ws.apache.org/wsrp4j/arch/arch.html>, 2003.
- [5] FOUNDATION, THE APACHE SOFTWARE: *Pluto Integration Guide*.
http://portals.apache.org/pluto/Howto_use_the_Pluto_container.html, 2004.
- [6] HOLMES, BRADFORD J.: *Benefits Portals Don't Deliver On Their Potential*, 2004.
- [7] KROPP, ALAN, CARSTEN LEUE und RICH THOMPSON: *Web Services for Remote Portlets Specification*, 2003.
- [8] MACVITTIE, LORI: *Enterprise Portals Suites*.
<http://www.nwc.com/showArticle.jhtml?articleID=18900467>, 2004.
- [9] PANDEY, PUNIT: *Portlets Development for Java Portals*.
<http://portlets.blogspot.com/>, 2004.
- [10] RAMOS, LAURA: *Portal Projects In Search Of A Purpose*, 2004.

Literaturverzeichnis

Abbildungsverzeichnis

2.1	Elemente eines Portals [1, Seite 19]	4
2.2	Generierung der Portalseiten [1, Seite 20]	5
2.3	Portlet Request Handling Sequence [1, Seite 25]	6
3.1	WSRP Architektur	24
5.1	Pluto Standard Layout	32
5.2	Pluto Portal Architektur [5]	32
5.3	Pluto Portlet Modes	33
5.4	Pluto Portlet minimiert	33