Seminararbeit

Entwicklung eines Portlet Abonnement Plugins für den IBM Workplace Client

Konzeption und Implementierung

Wirtschaftsinformatik 2/ Prof. Dr. Ludwig Nastansky

Contextual Collaborative Workplaces

Betreuer: Dipl.-Wirt.-Inf. Ingo Erdmann

Sommersemester 2005

vorgelegt von:

Torben Rasche

Studiengang Wirtschaftsinformatik

Matrikelnummer: 6120576

Brüderstrasse 36

33098 Paderborn

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation der Arbeit

Unternehmen sind seit geraumer Zeit auf der Suche nach Software-Lösungen, die es gestatten, alle zum täglichen Geschäftsablauf nötigen Funktionen und Programme integriert in einer Applikation vorzufinden. Es soll möglich sein, alle bestehenden Geschäftsprozesse auf dieser Plattform abzubilden, um die Kommunikation und das Datenmanagement zu verbessern und die Produktivität zu erhöhen. IBM hat mit dem Ziel diesen Kundenwünschen gerecht zu werden, eine neue Client-Technologie vorgestellt, die Anwendungen und Daten servergesteuert auf mehreren Clients anbieten soll, den *IBM Workplace Managed Client*. Damit soll eine Brücke von einfachen Web-Applikationen zu leistungsstarken Clients geschlagen werden, die die Vorteile beider Modelle vereint.

Der vermehrte Einsatz von Unternehmensportalen ist ebenfalls ein Trend der sich abzuzeichnen beginnt. In Zukunft wird es immer mehr Portalbetreiber geben, die die angebotenen Daten und Funktionen für Interessenten nach außen verfügbar machen. Unternehmen können die für sie interessanten Dienste zu Erledigung diverser Aufgaben nutzen und müssen somit nicht extra eigene Anwendungen dafür entwickeln. So lassen sich Entwicklungskosten und –zeit einsparen.

Verknüpft man nun diese beiden Ideen, so erhält man eine Geschäftslösung mit großem Potenzial. Der Benutzer arbeitet mit den integrierten Applikationen einer Plattform und kann bei Bedarf weitere Funktionalität von entfernten Portalen beziehen. Das zu entwickelnde Portlet Abonnement Plugin soll die Funktion des Beschaffens von entfernten Portlets in der Umgebung des *IBM Workplace Managed Client* möglich machen.

1.2 Aufbau der Arbeit

Nach der Einführung in Kapitel 1 werden im Zweiten die Grundlagen dieser Seminararbeit behandelt. Dieses umfasst die Themen Eclipse Plugins, *IBM Workplace Managed Client*, Portaltechnologien und Web Services. Auf Basis der *Web Services for Remote Portlets* (WSRP) Spezifikation des Konsortiums *Organisation for the Advancement of Structured Information Standards* (OASIS) wird dann in Kapitel 3 die Konzeption des Portlet Abonnement Plugins erstellt. Hier liegt der Schwerpunkt in dem Entwurf einer geeigneten Architektur und Präzisierung dieser anhand vorgesehener Schnittstellen. In Kapitel 4 wird schließlich auf die Implementierung einer prototypischen Lösung zur Umsetzung der Kon-

zeption eingegangen. Zusätzlich wird die Integration des Portlet Abonnement Plugins in den *Workplace Managed Client* erklärt, sowie Fähigkeiten und Schwächen der Implementierung dargelegt. Abschließend werden im letzten Kapitel die gewonnenen Eindrücke in Hinsicht auf die Funktionalität der entwickelten Lösung zusammengefasst und ein Ausblick über potenzielle Weiterentwicklungen gegeben.

2 Grundlagen

2.1 Eclipse und Plugins

Grundsätzlich wird Eclipse zumeist als eine Plattform zur integrierten Entwicklung gesehen. Seit mit der Einführung der Version 3.0 die *Rich Client Plattform* (RCP) veröffentlicht wurde, hat sich dieses Bild gewandelt. Eclipse ist nicht mehr eine reine Entwicklungsumgebung, vielmehr bildet diese selbst nur noch den Kern, der einzelne Plugins lädt. Die *Rich Client Plattform* basiert auf dem Standard des *Open Services Gateway Initiative* (OSGi) Industriekonsortiums (siehe [OSGi 2000]). Sowohl Eclipse als auch Plugins sind vollständig in Java implementiert.

Ein Plugin beschreibt die eigentliche Funktionalität, die in das bestehende System eingefügt werden soll. Mit Hilfe der Plugin Struktur von Eclipse können wartbare *Rich Clients* realisiert werden, da die Plugin Struktur eine Kapselung von verschiedenen Funktionalitäten und damit eine getrennte Wartbarkeit von Programmteilen ermöglicht. „Unter einem *Rich Client* versteht man Software, die applikationsspezifische Funktionalität direkt beim Klienten (z.B. Desktop oder mobile Plattform) implementiert. Das Gegenteil ist (…) der *Thin Client*, (…) bei dem applikationsspezifische Funktionen vom Server aus gesteuert werden.“(vgl. [Daum 2004] S.551). Ein Framework (Programmgerüst) aufgebaut aus Plugins, bietet eine sehr flexible Struktur und der Funktionsumfang kann durch Einbinden neuer Plugins stetig erweitert werden.

2.2 IBM Workplace Managed Client

„Eine einheitliche, vollständig integrierte Umgebung für elektronische Teamarbeit, die Ihnen die Flexibilität bietet, eine oder mehrere Funktionen in beliebiger Kombination zu implementieren und problemlos von einem Platz aus zu steuern, einschließlich Verwaltung und Richtlinienmanagement“ (vgl. [IBM 2005]) verspricht IBM auf seiner Homepage bzgl. der neuen Produktpalette *IBM Workplace Collaboration Services.*

Das Hauptanliegen des *Workplace* Konzeptes ist es, alle für eine bestimmte Aufgabe notwendigen Anwendungen in einer zentralen Benutzerschnittstelle integriert zur Verfügung zu stellen. Die Architektur ermöglicht es auf die gleichen Inhalte mit verschieden Clients zuzugreifen. Das zugrunde liegende Konzept basiert auf dem Ansatz des *Rich Clients* (vgl. Kapitel 2.1). Im Produktspektrum der *IBM Workplace Collaboration Services* ist eine *Rich Client*-Variante enthalten, die dem Ansatz der integrierten Verfügbarkeit aller relevanten Anwendungen folgt und mit einer funktionsreichen Benutzeroberfläche ausgestattet ist. Dieser *Rich Client* ist der so genannte *Workplace Managed Client* und bietet standardmäßig kollaborative Anwendungen, wie zum Beispiel Mail-, Instant Messaging-, Dokumentenmanagement- und Kalenderfunktionen an. Eine RCP-Anwendung bildet sich aus zusammengefassten Funktionsgruppen, dementsprechend handelt es sich auch bei dem *Worplace Managed Client* um eine RCP-Anwendung, die sich auf die RCP des Eclipse-Frameworks zurückführen lässt. Da es sich um eine erweiterbare Plattform handelt, besteht die Möglichkeit auch eigen entwickelte Anwendungen, in Form eines Eclipse RCP-Plugins, in den *Workplace* zu integrieren und für weitere Anwender bereitzustellen

Die Provisioning Komponente des Servers ist ein gutes Bespiel für den typischen Ablauf der Kommunikation zwischen *Workplace Managed Client* und *Workplace* Server. Nach dem erfolgreichen Anmelden des Clients am Server wird die aktuelle Client-Konfiguration geladen. Sind Updates oder neue Komponenten verfügbar, können diese mit Hilfe des Update-Managers vom Server bezogen werden.

2.3 Portaltechnologien

2.3.1 Portal

Ein Portal ist eine webbasierte Applikation, die einen zentralen Zugriff auf personalisierte Inhalte und Funktionen anbietet. Es erlaubt dem Benutzter bei einmaliger Anmeldung (Single Sign On) über eine einheitliche Benutzeroberfläche von mehreren Quellen Informationen zu beziehen. Zudem sind Portale prinzipiell unabhängig vom Endgerät, d.h. der Inhalt kann auf verschiedene Art und Weise dargestellt werden. Der Content wird z.B. für Browser in HTML und für Handys in WML präsentiert.

2.3.2 Portlets

Ein Portlet ist eine auf Java basierende Webkomponente, die innerhalb des Portals die Darstellung der statischen und dynamischen Inhalte übernimmt. Ein einzelnes Portlet ist ein

Teil des Portals, doch erst die Aggregation mehrerer Portlets bildet die eigentliche Portalseite. Man unterscheidet lokale und entfernte (Remote-) Portlets. Lokale Portlets laufen auf dem eigenen Portalserver und greifen auf die internen Daten und Funktionen zu, während Remote-Portlets sich auf fremden Servern befinden und via spezieller Web-Technologien in das eigene Portal übertragen werden müssen.

Im Laufe dieser Arbeit wird häufig von JSR-168-konformen Portlets die Rede sein. Der *Java Specification Request 168* (JSR-168) ist eine *Java Community Process* (JCP) Portlet Spezifikation (siehe [Abdelnur/Hepper 2003]) zur Interoperabilität zwischen Portal und Portlet, die eine neue Java API (siehe [Wikipedia 2005]) bereitstellt. Damit wurde unter Beteiligung renormmierter Firmen wie z.B. Sun, Oracle und IBM ein Portalstandard geschaffen, der es erlaubt JSR-168-konforme Portlets zu entwickeln, die in jeder Portalumgebung lauffähig sind.

2.3.3 Portlet Container

Der Portlet Container steuert den Lebenszyklus einzelner Portlets und bietet somit eine Ablaufumgebung, die die Kommunikation mit dem Portal abwickelt. Die Portlets werden lokal durch den Container gekapselt und auch durch diesen gesteuert und verwaltet. Eingehende Requests (Anfragen) werden durch den Container an das erforderliche Portlet weitergeleitet. Außerdem steht ein Persistenzmechanismus für die Portlet-Eigenschaften zur Verfügung. Der Portlet Container verwaltet lediglich die Erzeugung und das Zerstören der Portletinstanzen. Die einzelnen Portlets zu einer Gesamtansicht zu aggregieren ist Aufgabe des Portal Servers.

2.4 Web Services

„Ein Web Service ist ein Stückchen Geschäftslogik, das sich irgendwo im Internet befindet und auf das man mit standardisierten Internetprotokollen wie HTTP oder SMTP zugreifen kann. Die Verwendung eines Web Services kann so einfach wie die Anmeldung auf einer Site oder so komplex wie das Unterstützen einer Geschäftsverhandlung zwischen mehreren Organisationen sein.“ (vgl. [Chappel/Jewell 2003] S.1)

Ergänzend zu dieser Definition von Chappel und Jewell, ist die Standardisierung von Web Services eine wichtiges Charakteristikum, die Web Services von anderen Technologien abgrenzt. Folglich bieten Web Services eine Schnittstelle an, über die ihre Funktionen entfernt abgerufen werden können und die lose durch den Austausch von Nachrichten miteinander gekoppelt sind.

Den Kern der heutigen Web Service Technologien bilden die weltweiten Standards SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language) und UDDI (Universal Description, Discovery and Integration). Dabei dient SOAP als Übertragungsprotokoll, WSDL als Dienstbeschreibung und ein Dienstverzeichnis steht in Form von UDDI bereit. Die Inhaltsbeschreibung erfolgt anhand von standardisierten XML Dokumenten.

Der evolutionäre Schritt sind jedoch nicht die einzelnen Technologien an sich, sondern die Möglichkeit der gemeinsamen Interaktion, durch einen kombinierten Einsatz in einer standardisierten Infrastruktur. So ist eine automatische Geschäftsintegration denkbar, die ohne menschliches Eingreifen neue Dienste von Anbietern findet, auf diese zugreift, sie integriert und aufruft.

Zur näheren Erläuterung der einzelnen Dienste sei hier auf Chappel und Jewell mit ihrem Buch „Java Web Services“ [Chappel/Jewell 2003] verwiesen. Die Interaktion der Technologien untereinander wird in Kapitel 3 zur Konzeption vertieft.

3 Konzept zur Umsetzung

Im August des Jahres 2003 wurde von dem OASIS Konsortium der Standard *Web Services for Remote Portlets* verabschiedet und in der *Web Services for Remote Portlets Specification Version 1.0* (siehe [Kropp/Leue/Thompson 2003]) verankert. Auf Basis der WSRP Spezifikation soll nun in diesem Abschnitt eine Konzeption entwickelt werden, die veranschaulicht, wie ein Portlet Abonnement Plugin beschaffen sein muss, um die an sie gerichteten Anforderungen zu erfüllen.

3.1 Warum WSRP?

Das Ziel von OASIS war es, einen Standard für Remote Portlets zu etablieren, der das Ansteuern von JSR-168-konformen Portlets erlaubt, die sich via Web Services in das Portal integrieren lassen. Die Intention der Spezifiktion ist ein klare Abgrenzung der grundlegenden Begriffe und die genaue Trennung der Verantwortlichkeiten von Portal und Portlet Container.

Die Idee von WSRP die Portlets über einen Web Service anzubieten ist neu, da bislang Portlets fest installierte Komponenten auf dem Portalserver waren. Im Gegensatz zu üblichen Web Services übermitteln die Remote Portlets neben dem eigentlichen Inhalt und der

Geschäftslogik auch eine eigene Präsentationsschicht. In diesem Zusammenhang bezeichnet man den Anbieter der Portlets nach außen als WSRP Producer und den Dienst, der diese Portlets abruft und benutzt, als WSRP Consumer. Das zu entwickelnde Portlet Abonnement Plugin soll genau der Funktion eines WSRP Consumers nachkommen und deshalb werden im Folgenden die Begriffe Consumer und (Eclipse-) Plugin synonym verwendet.

In dem Gremium OASIS gehören zahlreiche renommierte Firmen wie beispielsweise IBM, Apache und Microsoft an, so dass von vornherein davon auszugehen war, dass die Spezifikation auf eine breite Akzeptanz stoßen wird. Die Entwicklung des Konsortiums lief koordiniert mit der JCP Gruppe ab, das parallel den JSR-168-Standard entwickelte. Diese Spezifikation liefert eine der Grundlagen für den Erfolg von WSRP, da neben der einheitlichen Übertragung der Geschäftslogik und Präsentationsschicht auch gewährleistet sein musste, dass die Portlets vom Aufbau her uniform sind. Proprietäre APIs verhinderten bis zu diesem Zeitpunkt den universellen Einsatz von Portlets in konkurrierenden Portalsystemen.

Die JSR-168-Spezifikation baut auf den bestehenden J2EE Konzepten auf, insbesondere aus der thematisch verwandten Servlet-Technologie wurden Konzeptionen übernommen und erweitert. API Bestandteile wie Funktionen zur Generierung dynamischer Inhalte, das Response/Request-Prinzip, Sessionverwaltung und ein Portlet Container als Ablaufumgebung sind Beispiele für die enge Verknüpfung untereinander. Neu konzipiert ist aber beispielsweise der Deployment (Bereitstellungs-) Deskriptor *portlet.xml*, der umfassende Optionen für die Applikation und die enthaltenen Portlets liefert (z.B. mehrsprachige Textressourcen und unterstützte Modi).

3.2 Portlet Abonnement Plugin auf Basis des WSRP Standards

Das zu entwickelnde Portlet Abonnement Plugin soll so konzipiert werden, dass es den Anforderungen der WSRP Spezifikation gerecht wird. Deshalb wird nun zuerst eine Soll-Architektur entwickelt und diese dann auf Basis der in der Spezifikation vorgestellten Standard-Schnittstellen präzisiert.

3.2.1 Architektur

Web Services for Remote Portlets basieren auf den Grundlagen der Standards für Web Services. Nachdem in den Grundlagen kurz erläutert wurde, was Web Services sind, soll nun auch dargestellt werden wie die Interaktion funktioniert und wie dieses Konzept in das Portlet Abonnement Plugin einfließt.

Die Web Service Architektur unterscheidet sich vom typischen Client/Server-Ansatz im Internet. Dabei greift der Client auf Daten und Funktionen des Servers zu und stellt die gelieferte Information des Servers im Browser dar. Der Web Service Ansatz beruht allerdings auf der Maschine/Maschine-Kommunikation, also dem Austausch von Daten und Funktionalität zwischen verschiedenen Anwendungen. Abbildung 1 zeigt ein Diagramm mit den drei zentralen Technologien von Web Services und ihren Beziehungen.

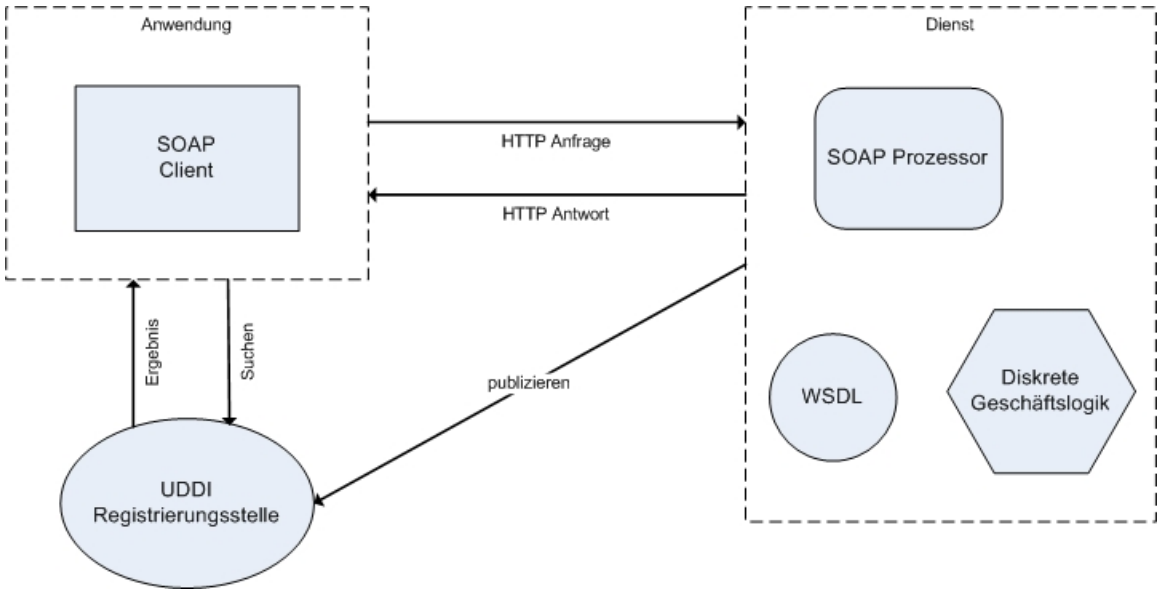

Abbildung 1 - Web Services Grundarchitektur (angelehnt an [Chappel/Jewell 2003] S. 5)

Web Services werden in einem Service-Verzeichnis, der UDDI Registrierungsstelle, aufgenommen und können über diese gefunden werden. UDDI hat den Weg zum Standard geschafft und wird von OASIS gepflegt. Eine Anwendung mit der Rolle eines Web Service Clients der nach einer anderen Anwendung oder einer spezifischen Geschäftslogik sucht, kann über die Registrierungsstelle einen Dienst abfragen. Dabei werden Parameter wie Name, Kategorie, Bezeichner oder die unterstützte Spezifikation als Suchkriterium angegeben. Wurde ein passender Eintrag gefunden, kann der der Dienst über das WSDL Dokument lokalisiert werden. Dieses Dokument beschreibt in Form eines XML Schemas wie der Kontakt aufgenommen wird und wie die Anfragen formatiert sein müssen. Bei der Übermittlung der Nachrichten hat sich mittlerweile SOAP durchgesetzt. Der Client erzeugt eine dem XML Schema aus dem WSDL Dokument korrespondierende SOAP Nachricht und schickt eine Anfrage an den Serviceanbieter. Dieser liefert, sofern der Request akzeptiert wird, die geforderten Informationen. Die weitere Kommunikation erfolgt ebenfalls auf Basis von Frage und Antwort. Das zu entwickelnde Portlet Abonnement Plugin (Consu-

mer) nimmt in diesem Szenario die Rolle des SOAP Clients ein, während der Producer derjenige Service ist, der seinen Dienst über die UDDI Registrierungsstelle publiziert hat und nach außen anbietet.

Diese Architektur muss nun allerdings für den Einsatz der Remote Portlets noch weiter spezifiziert und erweitert werden. Die Abbildung 2 zeigt die in der Literatur gängigste Darstellungsform der Architektur für die Kommunikation zwischen WSRP Consumer und WSRP Producer in einem Portalsystem.

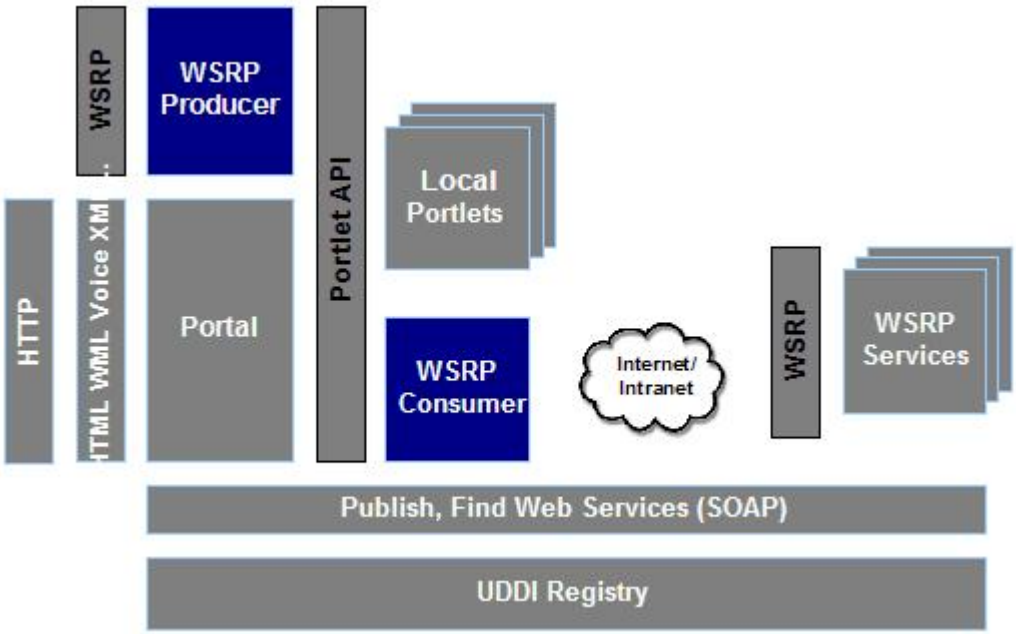

Abbildung 2 - WSRP Architektur (Quelle [WSRP4J 2003 b])

Das Portal identifiziert, wie bei den üblichen Web Services, einen WSRP Dienst über einen UDDI Dienst oder eine andere Methode und verbindet sich zu dem Anbieter über eine URL. Das Portal kann in diesem Model sowohl als Producer als auch als Consumer auftreten und verfügt über lokale Portlets, die entweder selbst im Portal konsumiert werden oder über einen Web Service für entfernte Portale bereitgestellt werden.

Portlets die über entfernte Portale bezogen werden, können über ein so genanntes *Proxyportlet* implementiert und angezeigt werden. Das *Proxyportlet* bietet einen lokalen generischen Platzhalter für Remote Portlets, die im Gegensatz zu den lokalen Portlets dem Portlet Container zu Anfang nicht bekannt sind. Die Kommunikation des *Proxyportlets* mit dem WSRP Service erfolgt mit Hilfe des Übertragungsprotokolls SOAP. So bekommt das *Proxyportlet* vom WSRP Anbieter das Markup zum Anzeigen des Inhalts übermittelt. Die ver-

wendete Markup Sprache ist dabei nicht auf eine bestimmte beschränkt, z.B. benutzt ein WAP Handy als Sprache WML, ein Voice Browser bezieht über Voice XML das Markup und ein normaler PC Browser nutzt HTML.

Der Unterschied zum *Proxyportlet* den das zu entwickelnde Portlet Abonnement Plugin aufweisen wird, ist das es sich nicht in der üblichen Portalumgebung befinden wird. Es wird als eine Art Standalone-Applikation im *Workplace Managed Client* laufen. Es soll allerdings die gleichen Merkmale und Funktionen aufweisen wie ein WSRP Consumer in einem Portal. Die obige Abbildung veranschaulicht auch noch einmal, dass es drei standardisierte Bereiche geben muss, um eine Interoperabilität über alle Schichten zu gewährleisten:

- Die Portlet API muss für unterschiedliche Programmiersprachen standardisiert sein. Unter Verwendung der JSR-168-Spezifikation ist dieses gewährleistet.
- Die Dokumenttypbeschreibung und Regeln bzgl. der Markup Fragmente werden aus der DTD für jede einzelne Markup Sprache abgeleitet.
- WSRP müssen unabhängig von der Programmiersprache basierend auf WSDL, SOAP und UDDI definiert und standardisiert werden.

Der Fokus wird im Folgenden nun im Bereich von WSRP liegen. Es werden die wesentlichen Bestandteile und Konzepte der WSRP Spezifikation von OASIS genannt, erläutert und analysiert. Der Schwerpunkt der Analyse wird darauf ausgerichtet sein, zu erkennen wie die Bestandteile dieser Spezifikation in die Implementierung des Portlet Abonnement Plugins einfließen.

3.2.2 Analyse und Verwendung der WSRP Spezifikation

An dieser Stelle können nicht alle Bestandteile der WSRP Spezifikation mit in die Konzeption einfließen, da sonst der Rahmen der Arbeit gesprengt würde. Darum werden nun nur die wichtigsten Merkmale der Spezifikation, die ausschlaggebend für den korrekten Ablauf des Potlet Abonnement Plugins sind, genau untersucht.

Um die Interaktion zwischen dem Producer und dem Consumer auf WSRP Basis zu standardisieren und vereinfachen stehen vier Schnittstellen (Interfaces) zur Verfügung. Über diese Schnittstellen läuft die gesamte Kommunikation zwischen Producer und Consumer. Diese werden vom Producer implementiert und können vom Consumer über Operationen aufgerufen werden. Im Folgenden werden die Schnittstellen nun näher betrachtet.

3.2.2.1 Service Description Interface

Die *Service Description* macht dem Consumer eine kurze Beschreibung bzgl. der vom Producer angebotenen Dienste publik und vereinfacht die Interaktion der beiden Teilnehmer. Die gesendeten Metadaten des Dienstes sind eine Art Selbstbeschreibung und geben Auskunft über die unterstützten Funktionen und Anforderungen.

Aus diesen Gründen muss der Producer das *Service Description* Interface implementieren und der Consumer kann über die Operation `getServiceDescription()` die Metadaten abrufen. Auf diesen Request antwortet der Producer mit einem `getServiceDescriptionResponse`-Dokument welches die nötigen Informationen enthält. In dieser *Service Description* ist u.a. die Information enthalten, ob eine Registrierung beim Producer notwendig ist. Falls benötigt, muss der Consumer sich erst beim Producer über das *Registration Interface* anmelden, ansonsten kann die Kommunikation ohne Verzögerung weitergehen. Außerdem gibt die Dienstbeschreibung Auskunft über angebotene Portlets. Ein verfügbares Portlet wird über einen eindeutigen `portletHandle` referenziert und über den gesamten Zeitraum der Interaktion zur Identifikation des Portlets benutzt.

3.2.2.2 Registration Interface

„Registration interface provides an in-band mechanism for a Consumer to register with a Producer and let the Producer customize its behavior for each Consumer based on the registration information“(vgl. [Allamaraju/Brooks 2004]).

Dieser Auszug aus der technischen Hilfe zum WSRP Standard verdeutlicht, dass es bei der Registrierung nicht nur um die eigentliche Anmeldung am Dienst geht, sondern auch die Definition eines Gültigkeitsbereichs und das Customizing des Verhaltens gegenüber dem Consumer von Bedeutung ist.

Bei der Registrierung bekommt das Plugin einen `registrationContext` vom Producer zugewiesen, welcher einen `registrationHandle` und einen `registrationState` beinhaltet. Der `registrationHandle` ist eine eindeutige Referenz des Consumers gegenüber dem Producer und bleibt über den kompletten Lebenszyklus unverändert. Fordert der Consumer im Laufe der Sitzung einen neuen Registrierungsstatus an, so wird dieser persistent im `registrationState` gespeichert. Die Abbildung 3 zeigt den Prozess zum Beziehen der *Service Description* bei erforderlicher Registrierung.

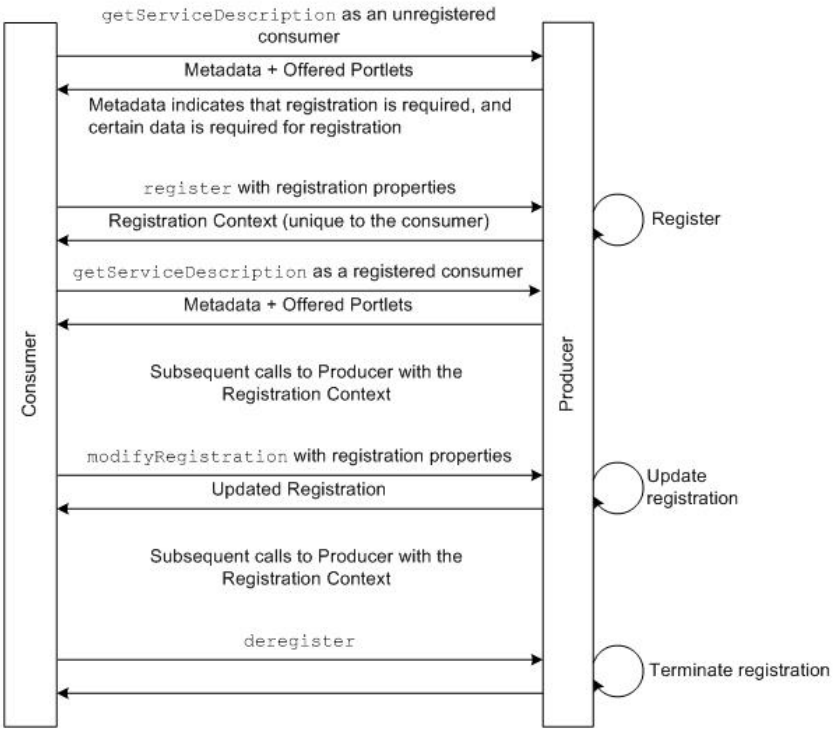

Abbildung 3 - Sequenzdiagramm zum Registrierungsprozess (Quelle [Allamaraju/ Brooks 2004])

3.2.2.3 Markup Interface

Um die Seiten der Portlets zu generieren, muss der Consumer zuerst das Markup der Portlets vom Producer erhalten. Betätigt ein Benutzer beispielsweise einen Hyperlink in dem Portlet, muss der Consumer in der Lage sein, einen Request an den Producer zu senden und die vom Producer gesendete Antwort bzgl. der Anfrage zu empfangen und zu verarbeiten. Das *Markup Interface* bietet zur Erfüllung dieser Anforderungen einige Operationen an. Alle Producer müssen diese Schnittstelle implementieren.

Die beiden wichtigsten Methoden des Markup Interface seien hier kurz erwähnt:

- `getMarkup()` Mit Hilfe dieser Operation kann der Consumer Markup Fragmente für ein bestimmtes Portlet empfangen
- `performBlockingInteraction()` Wird vom Consumer benutzt um vom Anwender ausgeführte Eingaben oder Aktionen zum Hersteller des Portlets zu senden.

Mit Hilfe dieser Operationen wird es dem Portlet Abonnement Plugin dann auch möglich sein mit dem Producer zu interagieren und die nötigen Inhalte und Informationen zu empfangen. Die folgende Abbildung 4 zeigt eine typische Interaktion bei der Übermittlung des Markups anhand der vorgestellten Operationen.

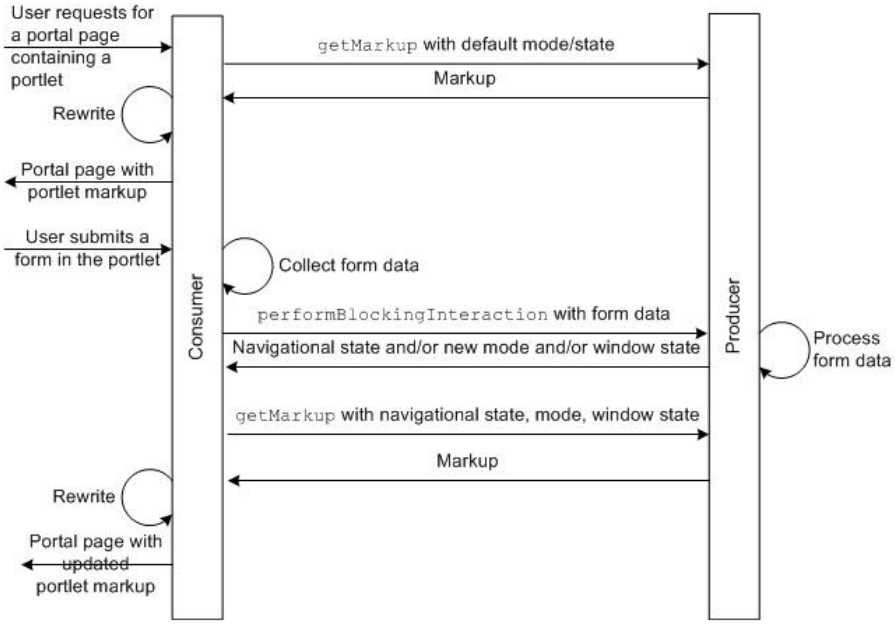

Abbildung 4 - Sequenzdiagramm zum Beziehen des Markups (Quelle [Allamaraju/ Brooks 2004])

a) URL Behandlung

Bei der Behandlung von URLs in Portlets agiert der Consumer als eine Art Vermittler zwischen dem End-Benutzer und dem Producer. Diese Rollenauslegung bietet zwei wesentliche Vorteile:

- Um dem Benutzer ein zusätzliches Maß an Personalisierungs-, Anpassungs- und Sicherheitsfeatures zu bieten, ist es notwendig, dass der Consumer als Vermittler agiert. So ist der Nutzer daran gebunden, den Inhalt der vom Producer angebotenen Portlets über den Consumer zu beziehen. Ein Bespiel für so ein Feature ist das Loggen (Aufzeichen) der Aktivitäten am Client.
- Viele Producer akzeptieren keine direkten HTTP-Request, sondern lediglich SOAP konforme Anfragen.

Die URLs im Markup sollten daher auf den Consumer referenzieren und nicht auf den Producer. Verweist die URL auf Consumer, nimmt dieser die Anfragen entgegen und leitet sie gemäß dem SOAP Protokoll zum Producer weiter. Das Markup Interface stellt zwei Möglichkeiten zum generieren der URLs bereit:

- *Consumer URL Rewriting* (siehe [Kropp/Leue/Thompson 2003] S.61 f.)
- *Producer URL Writing* (siehe [Kropp/Leue/Thompson 2003] S.63 f.)

Beide Techniken liefern sowohl Vor- als Nachteile. Beim *Producer URL Writing* gibt es kein zeitaufwendiges Parsen des Markups, jedoch führt das Übermitteln der Templates (Schablonen) bei jedem SOAP-Aufruf zu einem erhöhten Datentransfervolumen. Ein Vorteil des *Consumer URL Rewritings* ist die Erweiterbarkeit der URLs je nach Anliegen des Consumers. Somit muss von Fall zu Fall entschieden werden, welche Methode angewandt wird.

b) Portlet Modes und Window States

Portlets können sich in unterschiedlichen Modi befinden und unterstützen verschiedene Fensterzustände. Das fertige Plugin soll auch solche Konfigurationsmöglichkeiten unterstützen. Die WSRP Spezifikation liefert für diesen Sachverhalt die technischen Grundlagen.

Die Unterstützten Modi sind *View*, *Edit*, *Help* und *Preview*, wobei bis auf den *View* Modus alle optional sind. Im *View* Modus wird der eigentliche Inhalt angezeigt. Personalisierung und Ändern des Verhaltens können im *Edit* Modus vorgenommen werden. Zusätzlich liefert der *Help* Modus dem Anwender Hilfe-Information zum Portlet. Der *Preview* Modus gestattet dem Benutzer, eventuell vorgenommene Änderungen der Portleteigenschaften, in einer Vorschau zu überprüfen bevor diese persistent übernommen werden.

Desweiteren kann der Zustand eines Fensters *maximiert, minimiert, normal* oder *solo* sein. Der Fensterzustand *solo* indiziert, dass das Portlet das Einzige auf der aggregierten Seite ist, welches angezeigt wird. Alle verfügbaren Zustände sind jedoch optional.

3.2.2.4 Portlet Management Interface

Die vom Anbieter erzeugten Portlets können anfangs ohne Anpassung direkt genutzt werden. Aus Sicht der Benutzer kann es aber nötig sein, diese Portlets für die persönlichen Bedürfnisse anzupassen, beispielsweise ein Portlet zum Anzeigen des Wetterberichts auf das regionale Wetter zu begrenzen. Diese Anpassung der Portlets wird bei jedem Nutzer aber eine unterschiedliche Ausprägung haben und sollte ebenfalls auch in einen persistenten Zustand überführbar sein. Somit wird es nötig eigene Portlet Instanzen zu erzeugen mit der Möglichkeit diese auch zu modifizieren. In diesem Zusammenhang spricht man dann von *Consumer-Configured Portlets*. Das *Portlet Management Interfaces* bietet für eben diesen Zweck eine passende Schnittstelle an, die es ermöglicht den persistenten Status von Objekten und den Lebenszyklus zu steuern.

Producer können mit WSRP eine transparente Sicht auf persistente Zustände als Eigenschaften abbilden. Die Portlet-Eigenschaften können vom Consumer über diese Schnittstelle die Eigenschaften aufrufen und verändern. Der Lebenszyklus von Portlets kann aus drei Zuständen bestehen, die in der Abbildung 5 dargestellt sind. Die Pfeile zwischen den einzelnen Zuständen charakterisieren die Übergänge zwischen ihnen, wobei die dickeren Linien die Übergänge kennzeichnen die explizit vom Consumer durch Benutzen der Portlet Management Interfaces hervorgerufen sind.

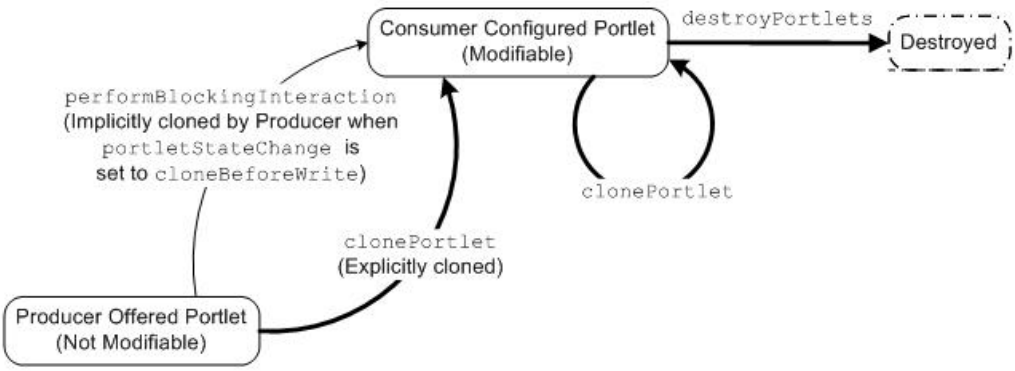

Abbildung 5 - Lebenszyklus von Portlet Instanzen (Quelle [Allamaraju/ Brooks 2004])

Ein *Consumer-Configured Portlet* kann auf zwei unterschiedlichen Wegen erzeugt werden, explizit oder implizit. Ein Producer kann implizit ein solches Portlet während einer `performBlockingInteraction()`-Operation erzeugen, beispielsweise bei dem Entfernen einer Spalte aus einer Tabelle. Nun handelt es sich nicht mehr um ein Standard Portlet, darum erzeugt der Producer einen Klon des Portlets und übermittelt ihm einen neuen *portletHandle* der auf die persönliche Instanz des Anwenders verweißt. Ein Consumer kann auch explizit durch Senden eines Requests an den Producer eine eigene Portlet Instanz anfordern. Dieses geschieht durch die Operation `clonePortlet()` und liefert dem Consumer wiederum einen neuen `portletHandle` zum Referenzieren des Portlets. So kann man die Einstellungen und Metadaten des Portlets auslesen und an die Bedürfnisse des Benutzers anpassen. Sie bleiben persistent bis sie durch die `destroyPortlet()`-Operation wieder zerstört werden.

Diese Schnittstelle ist laut Spezifikation zwar optional, soll aber auch im Eclipse Plugin zum Einsatz kommen um einen möglichst hohen Benutzerkomfort und eine breite Vielfalt an Funktionalität zu bieten. Für weitere Definitionen und Operationen dieser Schnittstelle sei hier auf die WSRP Spezifikation [Kropp/Leue/Thompson 2003] und den WSRP Primer [Allamaraju/Brooks 2004]) verwiesen.

4 Beschreibung der Beispielimplementierung

Das Apache WSRP4J (Web Services for Remote Portlets for Java) Open Source Projekt [WSRP4J 2003] nahm sich der Verantwortung an, eine Referenzimplementierung des WSRP Standards zu entwickeln. Diese Implementierung ist ausgestattet mit einem *Proxy-portlet*, das in einer Portalumgebung WSRP konforme Portlets anzeigen kann und einem *SwingConsumer* der als Standalone Applikation agiert und den Import von WSRP Portlets unterstützt. Die Architektur des Portlet Abonnement Plugins ist eng gekoppelt an die der Referenzimplementierung des WSRP4J Projektes. WSRP4J diente als Grundlage für die Entwicklung des Plugins.

4.1 Referenzimplementierung WSRP4J

Ins Leben gerufen wurde dieses Projekt durch IBM mit der Intention, WSRP möglichst schnell als Standard im Portalumfeld einzuführen und binnen kurzer Zeit viele WSRP Services frei im Internet verfügbar zu machen. Dabei soll sich WSRP4J als Plattform zur Entwicklung und Bereitstellung von WSRP-konformen Web Services etablieren. WSRP4J läuft zurzeit jedoch noch als *incubated*-Teilprojekt unter der Patenschaft der Apache Software Foundation (ASF) im Kontext eines allgemeinen Web Services Projektes. Der Status *incubated* drückt aus, dass es sich im Moment noch um eine Art Beta-Version handelt, die erst noch von der ASF freigegeben werden muss. Ein Projekt bleibt solange in diesem Status bis Kriterien wie Vollständigkeit und Stabilität der Anwendung voll gewährleistet sind.

Die Einzigartigkeit und freie Verfügbarkeit stellt eine gute Basis dar, die Ressourcen, die das Projekt bereitstellt, für das zu entwickelnde Plugin zu benutzen, auch wenn die Fehlerfreiheit von WSRP4J nicht vollständig gewährleistet ist. Der Sourcecode des *SwingConsumers* kann infolgedessen zu größten Teilen übernommen werden und erfordert lediglich eine Anpassung an die Lauffähigkeit in der Umgebung der Eclipse Plattform. So ist beispielsweise dazu zu sorgen, dass in der Applikation nicht mehr zur graphischen Darstellung Java Swing (siehe [Zukowski 2000] S.7 f.) genutzt wird, sondern das bei Eclipse übliche Standard Widget Toolkit (SWT) und JFace (siehe [Holzner 2004] Kapitel 7+8). Die Abbildung 6 zeigt die durch WSRP4J für den Consumer unterstützte Architektur und die Module die das Grundgerüst der Applikation formen.

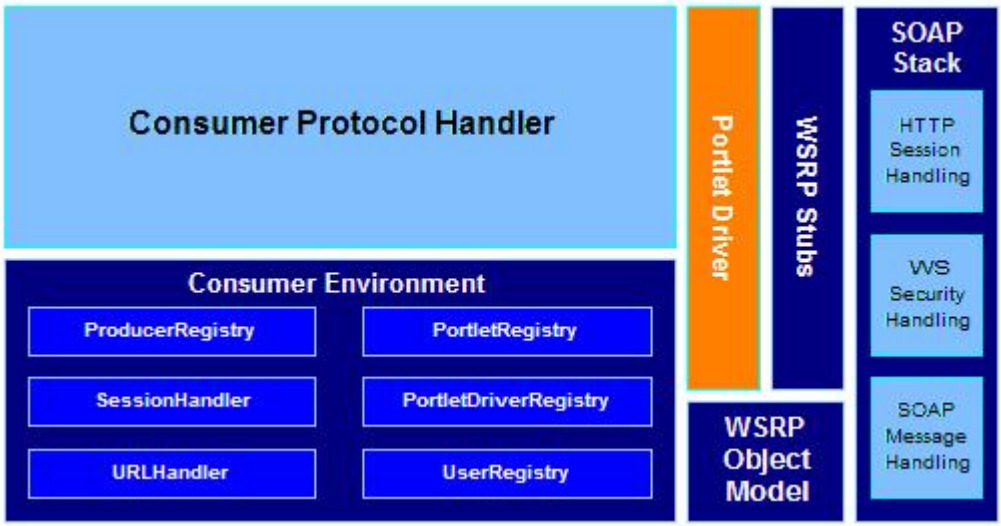

Abbildung 6 - Consumer Architektur in WSRP4J (Quelle [WSRP4J 2003 a])

Der *Protocol Handler* ist eine Standalone Swing basierte Applikation die das konsumierende Portal und die Browser Funktionalitäten implementiert. Es aggregiert die integrierten WSRP Portlets und leitet alle Aufrufe an den WSRP Service weiter. Die *Consumer Environment* wird dazu genutzt um alle benötigten Daten für einen WSRP Aufruf zu sammeln. Eine nähere Erläuterung der einzelnen Module ist in der Sektion Architektur auf der Homepage des WSRP4J Projektes zu finden (siehe [WSRP4J 2003 a]).

4.2 Aufbau der Lösung

Um eine lauffähige Lösung im *IBM Managed Workplace Client* zu erhalten, muss zuerst das Grundgerüst eines RCP-Plugins geschaffen werden und wird im ersten Abschnitt 4.2.1 dargelegt. Danach folgt die eigentliche Implementierung der applikationsspezifischen Programmlogik und endet mit einer kurzen Darlegung der Fähigkeiten und Schwächen der entwickelten Lösung.

4.2.1 Entwicklung des Grundgerüsts via RCP-Plugin

Bevor die Umsetzung des Eclipse Plugins in Angriff genommen werden konnte, war es notwendig sich mit der Eclipse Technologie vertraut zu machen. Daum [Daum 2004] und Brüssau/ Widdler [Brüssau/Widdler 2005] bieten dafür eine gute Literaturbasis.

Das Plugin muss zunächst mit einer Klasse ausgestattet werden, die das Interface `IPlatformRunnable` implementiert. Unter zu Hilfenahme dieses Interfaces kann der Applikations-Eintrittspunkt identifiziert werden. Für das Portlet Abonnement Plugin übernimmt die Klasse `Application` diese Aufgabe. Diese Klasse muss, da sie das Interface `IPlat-`

`formRunnable` implementiert, eine `run()`- Methode bereitstellen. Die Funktion der `Application`-Klasse entspricht der eines *Controllers*, ähnlich des *Controllers* in einer MVC-Architektur (siehe [Zukowski 2000] S.69 f.). Seine Hauptaufgabe liegt in der Erzeugung einer *Workbench* und des zugehörigem *Workbench Advisors*. Die *Workbench* ist das zentrale Grundgerüst einer RCP-Anwendung und liefert die Oberfläche über die alle Aktionen gesteuert werden.

„Die abstrakte Klasse `WorkbenchAdvisor` erlaubt, an verschiedenen Stellen im Lebenszyklus die Konfiguration der generischen *Workbench* in geeigneter Weise einzustellen“ (vgl. [Daum 2004] S.554). Desweiteren existiert noch die Klasse `WorkbenchWindowAdvisor`, die im Gegensatz zur Klasse `WorkbenchAdvisor` nicht die *Workbench* sondern einzelne Fenster in der Applikation konfiguriert. Beispielsweise kann über sie die Fenstergröße vor dem Öffnen des Fensters eingestellt werden. Jedes Fenster besitzt ein oder mehrere *Views* (Anzeigebereich). Dies ist der Bereich in dem die wesentliche Information, die Portlets, angezeigt werden soll. Die *Views* werden durch die entsprechende Klasse `View` initialisiert und können mit Hilfe der Klasse `Perspective` sichtbar gemacht werden. Jeder *View* muss einer bestimmten *Perspective* zugeordnet sein.

Bei der Erstellung eines Plugin-Projektes in Eclipse können vordefinierte Vorlagen genutzt werden um relativ schnell dieses Grundgerüst zu erstellen. Dafür werden dem Projekt die Eclipse-Biliotheken *org.eclipse.core.runtime* und *org.eclipse.ui* hinzugefügt, die für die oben beschriebenen Klassen die benötigten Interfaces enthalten. Die Grundlage zur Entwicklung des Portlet Abonnement Plugins wurde somit gelegt und die Implementierung der Programmlogik kann beginnen.

4.2.2 Implementierung

Die Basisfunktionalität des Plugins konnte durch den Import der WSRP4J Klassen und deren benötigten Bibliotheken in das Eclipse-Projekt übernommen werden. Dadurch waren schon von Beginn an die aus der Konzeption (siehe Kapitel 3) erforderlichen Interfaces und Operationen verfügbar.

Die Namenskonvention *SwingConsumer* wurde innerhalb des Quellcodes nicht geändert, da dieses an sehr vielen Stellen zu Änderungen geführt und auch evtl. neue Probleme aufgeworfen hätte. Somit besteht weiterhin eine Klasse mit dem Namen `SwingConsumer` ohne dass dort Java Swing verwendet wird. Es waren jedoch zahlreiche andere Anpassun-

gen des bestehenden Quellcodes erforderlich und im Folgenden sollen einige davon kurz erläutert werden.

Bei der WSRP4J Version des *SwingConsumers* erfolgt die Initialisierung der Anwendung durch den direkten Aufruf der `main()`-Methode in der Klasse `SwingConsumer`. An dieser Stelle bedurfte es der ersten Änderung. Beim Erzeugen der *Views* für das Plugin wird in der Methode `createPartControl()` zuerst ein Objekt der Klasse `SwingConsumer` instanziiert und danach über den Aufruf der `initLayout()`-Methode das Layout erstellt. Durch die Instanziierung eines Objektes der `SwingConsumer` Klasse werden auch alle anderen relevanten Klassen wie z.B. die `ConsumerEnvironment`, `PageRegistry` und `PortletRegistry` erzeugt.

Die Initialisierung des Layouts geschieht mit Hilfe des *SWT-Browers*. Eclipse stellt einen Webbrowser als *Widget* in Form der Klasse `Browser` im Package `org.eclipse.swt.eclipse` zur Verfügung. HTML-Inhalte können dadurch in SWT-Anwendungen einfach angezeigt werden. Eclipse benutzt allerdings keine eigene Browser-Implementierung, sondern greift auf den Standardbrowser der Ablaufumgebung zu, beispielsweise unter Windows eine OLE-Einbettung des Internet Explorers. Die Fähigkeiten des Browser-Widgets entsprechen dabei denen des Standardbrowsers mit den lokalen Einstellungen.

Zur Behandlung von *Events*, z.B. beim Anklicken eines Links, wurden die benötigten `EventListener` hinzugefügt. Dabei handelt es sich um den `LocationListener` und den `ProgressListener` die speziell für das Abfangen von solchen Ereignissen im Umfeld des *SWT-Browsers* zur Verfügung stehen. Wird beispielsweise ein Link ausgewählt so leitet der `LocationListener` über seine eigene Methode `changed()` den Event an die `onLink()`-Methode des WSRP Consumers weiter. In dieser `onLink()`-Methode wird dann gemäß dem WSRP-Standard die neue Seite generiert.

In der `initLayout()`-Methode wird mit Hilfe der `PageRegistry` die Startseite erstellt. Die `PageRegistry` liest aus XML-Dateien alle benötigten Information zur Lokalisierung der Remote Portlets. Es existieren drei primäre XML-Files die persistent die Speicherstelle des anzuzeigenden Portlets speichern. Eine XML-Datei ist für die Speicherung der benötigten Producerinformationen zuständig. Hier sind die vier zentralen URLs zum `WSRPBaseService`, `WSRPServiceDescriptionService`, `WSRPRegistrationService` und `WSRPPortletManagementService` abgelegt, die nötig sind, um eine Verbindung zum Producer aufzubauen. Es existieren noch zwei weitere XML-Dokumente zur Speicherung

der Portlet- und Seiteninformationen. Die zentrale Information die diese Dateien liefern, sind die Werte zum `portletHandle` (vgl. Kapitel 3.2.2.1).

In der bisherigen Version des *SwingConsumers* mussten, wenn man einen anderen Producer ansteuern wollte, alle Änderungen per Hand jedes Mal neu direkt in die XML-Files eingefügt werden. Das Portlet Abonnement Plugin bietet die Möglichkeit über die Auswahlbox *Change Portlet* verschiedene Producer mit deren Portlets aufzurufen. Die Informationen bzgl. der Anbieter wird in der Datei *portlet.txt* abgelegt und bei der Selektion eines Portlets aus der Auswahlbox dem Portlet Abonnement Plugin zur Verfügung gestellt. Anhand dieser Daten kann dann die neue Seite geladen werden. Diese neue Funktionalität ist eine der augenscheinlichsten Veränderungen gegenüber dem *SwingConsumer* und liefert einen weitaus verbesserten Komfort beim Aufrufen der Portletinstanzen.

Damit das Plugin in der Umgebung des *IBM Workplace Managed Client* lauffähig ist, kann das *IBM Workplace Managed Client Developer Toolkit* benutzt werden. Dieses ist ein auf Eclipse basiertes Hilfsprogramm, das den Aufwand der Entwicklung und des Deployens solcher Applikationen stark reduziert. Ein *Wizard* navigiert den Entwickler durch alle notwendigen Schritte zum Erstellen einer *IBM Workplace Managed Client* Applikation und bietet zudem die Möglichkeit, das Layout zu definieren. Es kann in einer lokalen *Workplace* Umgebung getestet werden ohne es auf dem Server zu deployen.

Weitere Änderungen und Erweiterungen (z.B. Umstellung von Swing auf SWT) des vorhandenen Quellcodes von WSRP4J, werden an dieser Stelle nicht weiter detailliert betrachtet. Dafür sei auf den kommentierten Quellcode verwiesen.

4.3 Fähigkeiten und Schwächen der Lösung

Das fertige Portlet Abonnement Plugin ist nun in der Lage als WSRP Consumer im *Workplace Managed Client* zu interagieren und Remote Portlets darzustellen. Es werden fast alle aus dem Konzept und der WSRP Spezifikation gestellten Anforderungen erfüllt. Die Abbildung 7 auf der nächsten Seite zeigt das Portlet Abonnement Plugin bei der Darstellung eines Apache WSRP Test Portelts.

Nach gründlicher Recherche im Internet konnten allerdings nur sehr wenig frei verfügbare Anbieter von WSRP-konformen Portlets gefunden werden. Über die Auswahlbox *Change Portlet* existieren vier Producer mit den zugehörigen Portlets von IBM, Oracle, Vignette und Sun. Es kann aber nicht gewährleistet werden, dass diese WSRP Services immer korrekt funktionieren, da viele Server häufig offline sind. Während der Testphase des Portlet

Abonnement Plugins erwiesen sich die Producer 1 und 2 der Auswahlbox von Oracle und IBM als die am Stabilsten. Durch Hinzufügen der Auswahlbox *Change Portlet* trat jedoch ein neues Problem auf. Wird das Portlet gewechselt, so funktionieren die Links im Portlet nicht mehr. Bei einem Klick auf den Link bleibt die Seite unverändert. Die Ursache dieses Fehlers ist, dass einige Objekte mit dem Wert null neu initialisiert werden. Bei dem Wechsel zu einem neuen Portlet passiert im Programmablauf das gleiche, wie bei dem eigentlichen Programmstart des Plugins. Es konnte bis dato noch nicht festgestellt werden, warum bei dieser Neuinitialisierung einige Objekte nicht wieder richtig erzeugt werden.

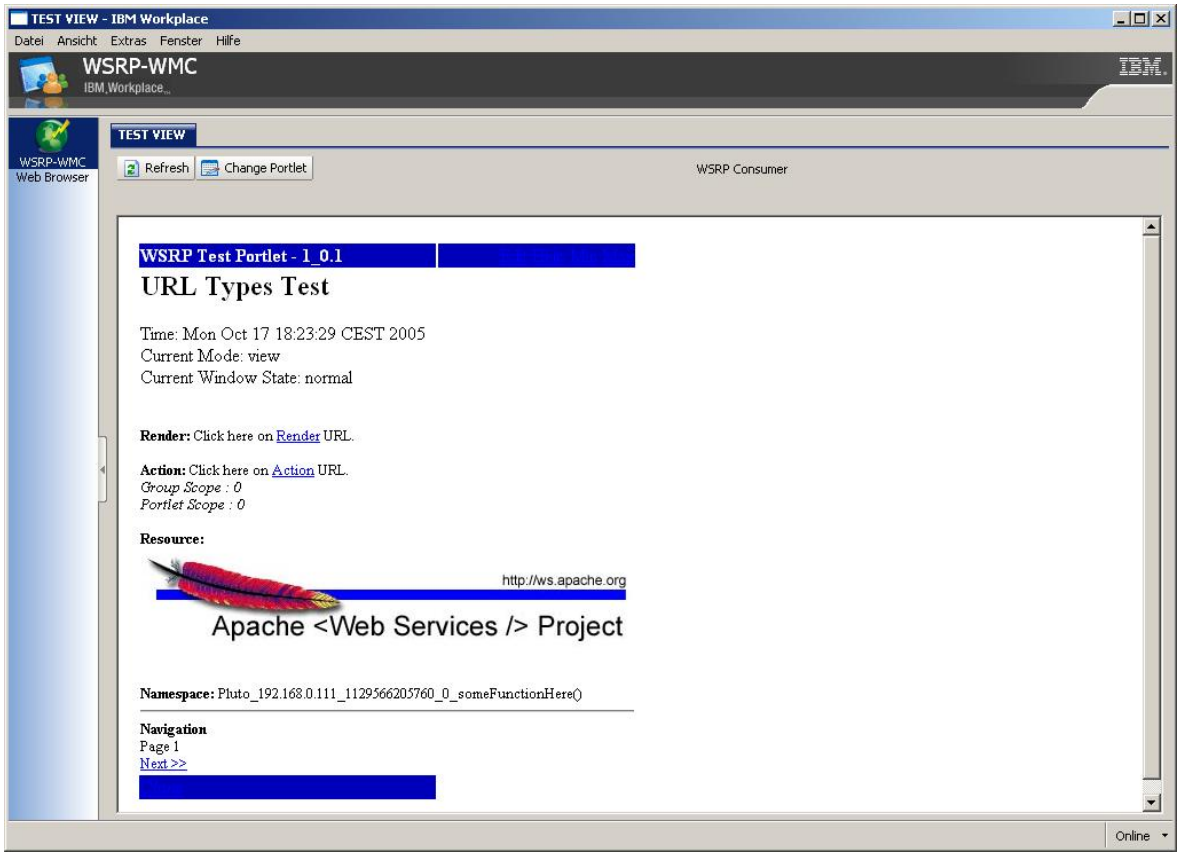

Abbildung 7 – Aufruf des Apache WSRP Test Portlets im Portlet Abonnement Plugin

Das größte Problem der Anwendung stellt die Behandlung von *POST*- und *GET*-Request in *Forms* (siehe [Jones/Nye 1995] S.118 f.) dar. Betätigt ein Benutzer in einem Portlet einen Button oder führt irgendeine andere Aktion aus, so werden die gesendeten Parameter nicht korrekt übertragen. Die Seite bleibt in dem gleichen Zustand wie vor dem Auslösen der Aktion. *Forms* werden in dem WSRP4J Projekt für die Version des *SwingConsumers* nicht unterstützt. Daraus folgt auch, dass die Funktionalität des Portlet Management Interfaces bisher kaum getestet werden konnte. Werden Einstellungen am Portlet über den *Edit*-Link

geändert, so können diese nicht an den Producer übermittelt werden, solange *Forms* nicht unterstützt sind. Das Portlet Management Interface kann nur über den Link *Clone Portlet* angesteuert werden, welche eine zweite Instanz des aktuellen Portlets erzeugt (siehe Kapitel 3.2.2.4). In dieser zweiten Instanz kann dann beliebig navigiert werden, wird jedoch der *Workplace Managed Client* am Ende einer Sitzung geschlossen, so werden die geklonten Instanzen wieder zerstört.

5 Fazit

Die oben genannten Schwächen des Plugins, macht es zu einer Applikation mit sehr eingeschränkter Funktionalität. Als Resume muss man feststellen, dass WSRP4J ein anfängliches gutes Grundgerüst für die Umsetzung liefert, dieses jedoch noch sehr fehlerhaft ist. Somit erhält man eine prototypische Implementierung der WSRP Spezifikation in Form eines Eclipse Plugins, welche jedoch für den operativen Einsatz noch nicht geeignet ist.

WSRP4J als Pilotprojekt im Jahre 2001 mit zu Anfang großem Interesse von allen Seiten, ist nie aus dem *incubator*-Status herausgekommen. Da sich mit IBM der eigentliche Initiator von WSRP4J fast vollständig von der Mitarbeit zurückgezogen hat, ist der Projektfortschritt quasi zum Erliegen gekommen. Zurzeit sind nur noch wenige Personen dabei, den Quelltext nach persönlichen Belangen auszubauen. Somit ist auch zukünftig nicht damit zu rechnen, dass die festgestellten Probleme in absehbarer Zeit gelöst werden. Infolgedessen muss bei eventuellem Interesse daran, die bestehende Lösung des Portlet Abonnement Plugins zu verbessern, überlegt werden, ob es günstiger ist das Problem direkt in der Plugin-Lösung zu beheben oder einen ganz neuen Ansatz zu entwickeln.

Jedoch können auch zahlreiche positive Aspekte aus dieser Arbeit gezogen werden. Es bietet jedem Leser einen Erfahrungsbericht bzgl. der vielen neuen Technologien die in dieser Arbeit behandelt werden, wie etwa der *IBM Workplace Managed Rich Client*, Eclipse RCP-Plugins und WSRP. Die neuen Potenziale die sich aus der Entstehung der neuen Technologien ergeben sind sehr groß, speziell der minimierte Aufwand im Bereich der Entwicklung ist einer der Hauptantriebsfaktoren. Der WSRP Standards und auch der des damit verbundene JSR-168, findet in dem Umfeld der Portalprovider einen immer größer werdende Breite an Akzeptanz. Somit ist damit zu rechnen, dass in naher Zukunft der Anteil an WSRP- und JSR-168-konformen Portlets steigt und eine breite Palette an freien Diensten im Internet verfügbar ist. Dieses würde auch dem Portlet Abonnement Plugin einen Mehrwert an Funktionalität innerhalb der *Workplace* Umgebung zukommen lassen.

Wenn in Zukunft dann mehr WSRP Producer existieren, bietet es sich auch an, das Plugin um eine Schnittstelle zu erweitern, die eine UDDI Registrierungsstelle nach verfügbaren Diensten abfragt. Diese Implementierung wurde bis zu diesem Zeitpunkt bewusst weggelassen. Erstens aufgrund der wenigen Anbieter und zweitens da bis jetzt noch nicht unterschieden werden kann, ob die angebotenen Portlets WSRP-konform sind.

Abzuwarten ist auch welche Änderungen und Neuheiten die für Mitte 2005 angekündigte Version 2.0 der WSRP Spezifikation bringen wird. Eine oft geäußerte Kritik an dem bisherigen Standard ist, dass es keinen Support für eine Portlet-to-Portlet Kommunikation gibt. So wären Szenarien denkbar, dass z.B. ein Wetter-Portlet zusammen mit einem Landkarte-Portlet interagiert. Durch Selektieren eines Bereiches auf der Landkarte, wird in dem Wetter-Portlet zu der ausgewählten Region die Temperatur angezeigt.

Zusammenfassend kann also festgehalten werden, dass die Verknüpfung neuer Technologien, mit teilweise neuen Ideen, Konzepten und Standardisierungen, interessante neue Möglichkeiten darbieten. Die technischen Probleme, welche meist generell bei neuen Lösungswegen vorzufinden sind, werden in Zukunft durch weitere Standardisierung und einer größerer Menge an Erfahrungen bzgl. der neuen Technologien gelöst werden.

6 Literaturverzeichnis

[Abdelnur/Hepper 2003] Abdelnur, Alejandro; Hepper, Stefan (2003): Java Portlet Specification Version 1.0

Aus: http://jcp.org/aboutJava/communityprocess/final/jsr168/index.html

am 03.10.2005

[Allamaraju/ Brooks 2004] Allamaraju, Subbu; Brooks, Rex (2004): OASIS Web Services for Remote Portlets 1.0 Primer

Aus:

http://www.oasis-open.org/committees/download.php/10539/wsrp-primer-1.0.html

am 03.10.2005

[Brüssau/Widdler 2005] Brüssau, Kai; Widdler, Oliver (2005): Eclipse – Die Plattform

1. Auflage, Software & Support Verlag, Frankfurt 2005

[Burnette 2004] Burnette, Ed (2004): Rich Client Platform Tutorial

Aus: http://eclipse.org/rcp/

am: 03.10.2005

[Chappel/Jewell 2003] Chappel, A. David; Jewell, Tyler (2003): Java Web Services

1. Auflage (Deutsche Ausgabe), O'Reilly Verlag GmbH &Co. KG, Köln 2003

[Daum 2004] Daum, Berthold (2004): Java-Entwicklung mit Eclipse 3

2., überarbeitete und erweiterte Auflage, dpunkt verlag GmbH, Heidelberg 2004

[Graham 2002] Graham, Steve; Simeonov, Simeon; Boubez, Toufic; Davis, Doug; Daniels, Glen; Nakamura, Yuichi; Neyama, Ryo (2002): Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI

1. Auflage, SAMS, Indianapolis, Indiana 2002

[Hauser/Löwer 2004] Hauser, Tobias; Löwer, Ulrich M. (2004): Web Services - Die Standards

1. Auflage, Galileo Computing, Bonn 2004

[Holzner 2004] Holzner, Steve (2004): Eclipse

1. Edition, O'Reilly Media, Inc., Sebastopol 2004

[Hunter/Waller/Dovey/Tilton/Allen/Awre 2004] Hunter, Jon; Waller, Stewart; Dovey, Matthew; Tilton, Justin; Allen, Jonathan; Awre, Chris (2004): Contextual Resource Evaluation Environment - Investigation of JSR 168 and WSRP standards

Aus: http://www.hull.ac.uk/esig/cree/downloads/CREEtechnicalreportA.pdf

am 03.10.2005

[Jones/Nye 1995] Jones, Russ; Nye, Adrian (1995): HTML und das World Wide Web

1. Auflage, O'Reilly/International Thomson Verlag, Bonn (1995)

[Kirchhof/Gurzki/Hinderer/Vlachakis 2004] Kirchhof, Anja; Gurzki, Thorsten; Hinderer, Henning; Vlachakis, Joannis (2004): Was ist ein Portal (Whitepaper)

Aus:
http://www.ebi.iao.fraunhofer.de/Whitepaper%20Was%20ist%20ein%20Portal_mit%20Logo.pdf

am 03.10.2005

[Koschek 2004] Koschek, Holger (2004): Porta(be)l - Portlet-Spezifikation harmonisiert Portale. In: iX, Heise Zeitschriften Verlag GmbH & Co. KG, Ausgabe 9 (2003), S.108-S.111

[Kraenzel 2004] Kraenzel, Carl (2004): IBM Workplace Client Technology (Rich Client Edition) ISV Integration Guide

Aus: http://www.redbooks.ibm.com/redpapers/pdfs/redp3883.pdf

am 03.10.2005

[Kropp/Leue/Thompson 2003] Kropp, Alan; Leue, Carsten; Thompson, Richard (2003):

OASIS Standard: Web Services for Remote Portlets Specification Version 1.0

Aus:

http://www.oasis-open.org/committees/download.php/3343/oasis-200304-wsrp-specification-1.0.pdf

am 03.10.2005

[Schaeck 2002] Schaeck, Thomas (2002): Web Services for Remote Portlets (WSRP) Whitepaper

Aus: http://www.oasis-open.org/committees/wsrp/documents/wsrp_wp_09_22_2002.pdf

am 03.10.2005

[Thompson/Schaeck 2003] Thompson, Richard; Schaeck, Thomas (2003): Enabling Interactive, Presentation-Oriented Content Services through the WSRP Standard

Aus: http://www.idealliance.org/papers/dx_xml03/papers/04-06-05/04-06-05.html

am: 03.10.2005

[Zukowski 2000] Zukowski, John (2000): Definitive Guide to Swing für Java 2

2. Auflage, Apress, Berkeley 2000

Ohne Verfasser

[IBM 2005] IBM (2005): Software Online Katalog von IBM Deutschland (2005)

Aus:

http://www-306.ibm.com/software/info/ecatalog/de_DE/products/I286055S78920R10.html

am 03.10.2005

[Grama/Attenborough/Banks-Binici/Marsden/Kraenzel/Calow/Ramaswamy/Zurko 2004] Grama, Harish; Attenborough, Keith; Banks-Binici, John; Marsden, Jim; Kraenzel, Carl; Calow, Jeff; Ramaswamy, Shankar; Zurko, Mary Ellen: IBM Workplace Client Technology (Rich Client Edition) Technology Overview

Aus: http://www.redbooks.ibm.com/redpapers/pdfs/redp3884.pdf

am 03.10.2005

[OSGi 2003] OSGi (2000): Service Gateway Specification Release 1.0 (Dokumentation)

Aus: http://www.osgi.org/osgi_technology/

am 03.10.2005

[Wikipedia 2005] Wikipedia (2005): Application Programming Interface (API)

Aus: http://de.wikipedia.org/wiki/Application_Programming_Interface

am 03.10.2005

[WSRP4J 2003] WSRP4J (2003): WSRP4J Homepage

Aus: http://ws.apache.org/wsrp4j/

am 03.10.2005

[WSRP4J 2003 a] WSRP4J (2003): WSRP Consumer Architecture

Aus: http://ws.apache.org/wsrp4j/arch/consumer.html

am 03.10.2005

[WSRP4J 2003 b] WSRP4J (2003): WSRP Architecture

Aus: http://ws.apache.org/wsrp4j/arch/index.html

am 03.10.2005