Orientation in Objects

Einsatzmöglichkeiten der Eclipse RAP

Einsatzmöglichkeiten der RAP in einer Java-Enterprise-Architektur

) Schulung)

) Beratung)

) Entwicklung)

AUTOR

Steffen Schäfer
Orientation in Objects GmbH

Veröffentlicht am: 9.4.2008

EINSATZMÖGLICHKEITEN VON RAP IN EINER JAVA-ENTERPRISE-ARCHITEKTUR

) Artikel)

In den letzten Jahren hat sich die Rich Client Platform (RCP[1]) zu einem etablierten Application Framework entwickelt. Mit der Rich Ajax Platform (RAP[2]) sollen die Vorteile und Konzepte, die RCP auszeichnen nun auch bei der Web-Entwicklung genutzt werden können. RAP greift dabei auf den Extension Point Mechanismus zurück und bietet einige aus der RCP Entwicklung bekannte Extension Points.

Dieser Artikel behandelt die typischen Problemstellungen, welche einen Entwickler bei der Arbeit mit RAP beschäftigen. Es wird vorausgesetzt, dass sich der Leser bereits mit RCP und RAP beschäftigt hat. Zur Einarbeitung in RAP existieren einige Artikel [3].

Orientation in Objects GmbH

Weinheimer Str. 68
D-68309 Mannheim

Tel. +49 (0) 6 21 - 7 18 39 - 0
Fax +49 (0) 6 21 - 7 18 39 - 50

www.oio.de info@oio.de

EINLEITUNG

Stellen sie sich vor, dass sie eine erfolgreiche RCP-Anwendung entwickelt haben. Um den Bedürfnissen des Marktes gerecht zu werden, müssen sie nun den Sprung ins Web wagen. Das ist allerdings leichter gesagt, als getan. Die Einarbeitung in ein Web-Framework und die Migration einer Anwendung darauf kostet wertvolle Zeit. RAP verspricht nun, dass damit Web 2.0 Anwendungen mit den bekannten APIs der Rich Client Platform entwickelt werden können. Somit bietet sich der Einsatz der bestehenden RCP-Anwendung mit RAP an.

Doch eine 100%ige Kompatibilität zu RCP kann auch RAP nicht bieten. Es gibt dort bisher nicht umgesetzte Teile (im Vergleich zu RCP) und auch mit den Besonderheiten der Web-Entwicklung muss man sich zwangsläufig beschäftigen. Der Artikel soll deshalb die bei der Entwicklung mit RAP auftretenden Probleme und dazu passende Lösungsstrategien vorstellen.

Anhand einer Beispielanwendung zur Verwaltung von Bookmarks wurden die in diesem Artikel vorgestellten Konzepte getestet. Um typischen Einsatzszenarien möglichst nahe zu kommen, enthält die Anwendung einen Server zur Datenverwaltung, der mit Hilfe des Spring Frameworks vom Client verwendet wird. Abbildung 1 zeigt das Hauptfenster mit der Perspektive der Beispiel-Anwendung.

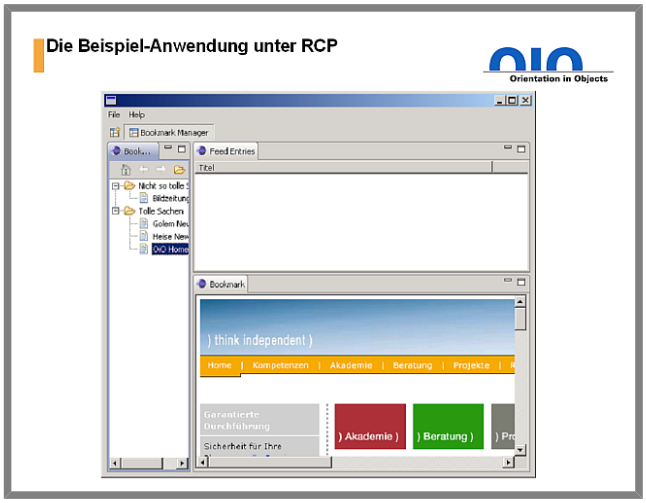

Abbildung 1: Die Beispiel-Anwendung unter RCP

GLIEDERUNG

- Allgemeine Probleme bei der Entwicklung mit RAP
- Portierung einer Anwendung von RCP auf RAP
- Entwicklung einer Anwendung für beide Plattformen
- Das langfristige Ziel

ALLGEMEINE PROBLEME BEI DER ENTWICKLUNG MIT RAP

Dieser Teil des Artikels behandelt allgemeine Probleme bei der Entwicklung mit RAP. Hierfür sollen auch passende Lösungen besprochen werden.

Der größte Unterschied zwischen Desktop- und Web-Entwicklung ist die Mehrbenutzerfähigkeit von Web-Anwendungen. Eine RCP-Anwendung läuft auf dem Rechner des Benutzers. Somit steht jedem Benutzer eine eigene Instanz der Anwendung zur Verfügung. Eine RAP-Anwendung ist jedoch eine Web-Anwendung und läuft auf einem Server. Mehrere Benutzer teilen sich also eine Instanz der Anwendung. Abbildung 2 zeigt diesen Unterschied schematisch am Beispiel einer Client-Server-Anwendung.

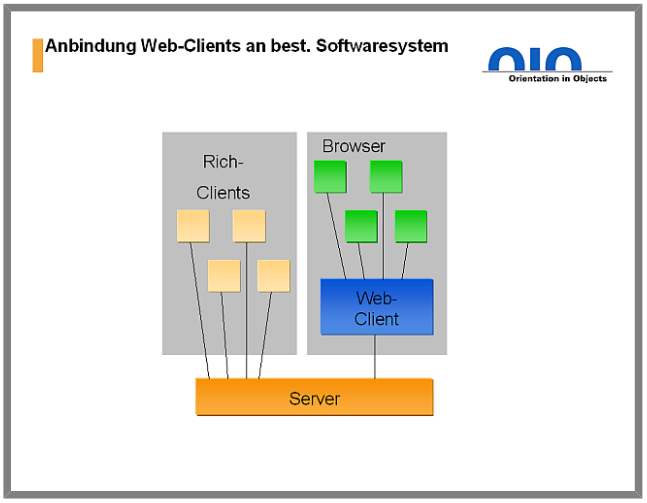

Abbildung 2: Anbindung einer Web-Anwendung an eine Java-Enterprise-Architektur

Aus diesem Grund muss man sich bei der Entwicklung einer Web-Anwendung Gedanken darüber machen, wie die benutzerspezifischen Daten der einzelnen Anwender voneinander getrennt aufbewahrt werden können. Hierfür wird eine Session verwendet, in welcher die benutzerspezifischen Daten hinterlegt werden können. Während der Bearbeitung von Requests eines Benutzers kann die Anwendung auf die Session des Benutzers zugreifen und mit den enthaltenen Daten arbeiten.

Für eine Umsetzung von Session-Zugriffen stellt RAP verschiedene Möglichkeiten bereit. Neben dem direkten Zugriff auf die Session über die Klasse "RWT", kann auch ein Singleton implementiert werden, der eine Instanz pro Session zur Verfügung stellt. Beispiel 1 zeigt die Implementierung einer solchen Klasse.

```
import org.eclipse.rwt.SessionSingletonBase;
public class MySessionSingleton extends SessionSingletonBase
{
 public static MySessionSingleton getInstance() {
  return
  (MySessionSingleton)getInstance(MySessionSingleton.class);
 }
 [...]
}
```

Beispiel 1: Implementierung eines SessionSingletons

Die Methode "getInstance()" gibt nun bei jedem Aufruf eine Instanz der Klasse zurück, welche nur für den aktuellen Benutzer gültig ist. Für jeden Benutzer wird außerdem automatisch eine neue Instanz der Klasse erstellt, sodass hierfür keine zusätzliche Programmierarbeit notwendig ist.

Ein weiteres grundlegendes Problem hat damit zu tun, dass RAP andere Plugins umfasst, als RCP. Aus diesem Grund funktioniert eine RCP-Anwendung nicht ohne Änderung auch mit RAP. Man muss deshalb bei der Entwicklung mit RAP zu den speziellen Plugins greifen, welche RAP mitbringt. Als Übersicht zeigt Tabelle 1 eine Gegenüberstellung einiger RCP Plugins mit den entsprechenden Plugins bei RAP.

RCP-Plugin	RAP-Plugin
org.eclipse.swt	org.eclipse.rap.rwt
org.eclipse.jface	org.eclipse.rap.jface
org.eclipse.workbench	org.eclipse.rap.workbench
org.eclipse.ui	org.eclipse.rap.ui

Tabelle 1: Gegenüberstellung der Plugins von RCP und RAP

Das Plugin "org.eclipse.ui" basiert auf den grundlegenden RCP-Plugins (SWT, JFace, Workbench) und exportiert diese wiederum. So stehen beim Import von "org.eclipse.ui" auch diese Plugins direkt zur Verfügung. Bei RAP nimmt das Plugin "org.eclipse.rap.ui" diese Rolle ein. Die Dependencies einer RAP-Anwendung könnten also wie in Abbildung 3 aussehen.

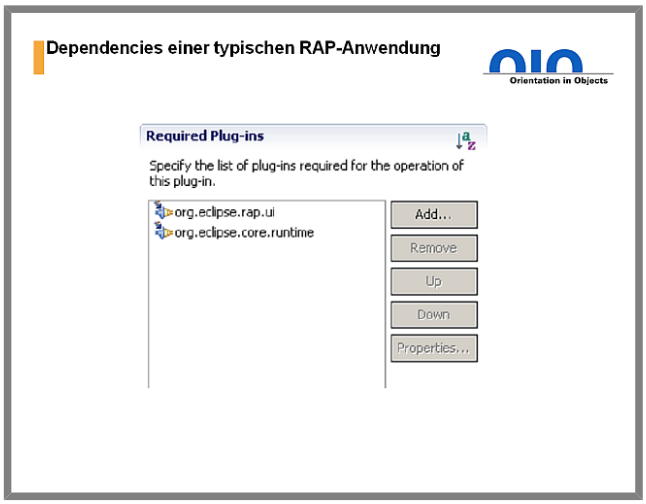

Abbildung 3: Dependencies einer typischen RAP-Anwendung

Bei dieser Gegenüberstellung von RCP und RAP entsteht der Eindruck, dass es für jedes Plugin eine entsprechende RAP-Version gibt. Das ist jedoch nur bei den Basisplugins der Fall. Sind darauf aufbauende Plugins nicht von sich aus schon kompatibel zu RAP, so gibt es in den seltensten Fällen ein kompatible Version. Die Folge ist also, dass diese Plugins nicht eingesetzt werden können, oder auf RAP portiert werden müssen. Dies betrifft alle Plugins, die nicht zum grundlegenden Umfang der Rich Client Platform gehören.

Zur Veranschaulichung soll an dieser Stelle ein prominentes Beispiel aus der Praxis dienen. Ein beliebtes Plugin in wirtschaftlichen Anwendungen ist BIRT, mit dessen Hilfe auf einfache Weise Reporting in der eigenen Anwendung umgesetzt werden kann. Es ist nicht möglich, den BIRT-Designer in einer RAP-Anwendung einzusetzen. Als Alternative für die Portierung kann man in Betracht ziehen, den Einsatz des Designers auf die RCP-Version der Anwendung zu beschränken. In der RAP-Version können bestehende Reports jedoch einfach in einem Browser-Widget angezeigt werden. Mit dieser Lösung steht allen Benutzern das Reporting zur Verfügung, neue Reports können aber ausschließlich in der Desktop-Anwendung erstellt werden.

Neben diesem speziellen Problem wollen wir uns nun wieder RAP selbst zuwenden. Es müssen nun noch weitere Unterschiede beachtet werden, die durch das Fehlen von APIs bei RAP zustande kommen. Auch für diese müssen teilweise Alternativlösungen gefunden werden. Um welche genauen Unterschiede es sich handelt und wie man diese angehen kann, soll folgende Auflistung zeigen:

- Das in Eclipse 3.3 enthaltene DateTime-Widget wird in RAP nicht unterstützt. Es gibt aber einige alternative Date-Picker-Widgets für SWT. Manche von diesen funktionieren auch unter RAP, sodass eine Alternative zur Verfügung steht. Stattdessen kann auch ein JavaScript Date-Picker in RAP als eigenes Widget eingebunden und benutzt werden.
- Das Editieren von Zellinhalten in Tabellen und Bäumen ist nicht möglich. Tabellen und Bäume, deren Inhalte editiert werden sollen, können mit einem Detail-Bereich versehen werden. Dieser stellt für die einzelnen Spalten Eingabefelder zur Verfügung, mit deren Hilfe die Inhalte bearbeitet werden können.
- Bei RAP fehlt die Unterstützung für drag&drop. In vielen Fällen lässt sich die Funktionalität, welche auf drag&drop basiert durch Kopieren und Einfügen der betreffenden Elemente lösen.
- RAP unterstützt nicht alle Event-Typen, die SWT bietet. Es fehlen z.B. Mouse- und KeyEvents. In einigen Fällen helfen SelectionEvents statt MouseEvents und ModifyEvents statt KeyEvents bei der Umsetzung entsprechender Use Cases.
- Native Dialoge wie Dialoge zur Datei-Verarbeitung sind in RAP nicht enthalten. Da RAP-Anwendungen auf einem Web-Server und nicht lokal laufen, hat die Anwendung ohnehin keinen Zugriff auf das lokale Dateisystem. Somit muss man generell andere Lösungen für die Dateiverarbeitung finden.
- Das StyledText-Wiget, welches die Grundlage von Text-Editoren in Eclipse bildet steht nicht zur Verfügung. Ist es für einen Fall nur wichtig, formatierten Text zu bearbeiten, so kann ein JavaScript-Widget hierfür verwendet werden. Ist die Anwendung jedoch auf die APIs des speziellen Widgets angewiesen, so steht momentan keine einfache Lösung zur Verfügung.

Wie man sieht, lassen sich manche Probleme sehr elegant lösen. Bei anderen muss man jedoch etwas tiefer in der Trickkiste graben, um eine passende Lösung zu finden, die die Anwender zufrieden stellt.

PORTIERUNG EINER ANWENDUNG VON RCP AUF RAP

Der folgende Teil des Artikels beschreibt die nötigen Änderungen, um eine Anwendung von RCP auf RAP zu portieren.

Da RAP andere Plugins umfasst, als RCP (siehe Tabelle 1), müssen im ersten Schritt die Abhängigkeiten geändert werden. Statt wie bisher z.B. das Plugin "org.eclipse.swt" zu verwenden, wird nun "org.eclipse.rap.rwt" verwendet. Die Abhängigkeiten der einzelnen Plugins sind in der Manifest-Datei definiert. Beispiel 2 zeigt einen Ausschnitt dieser Datei. Das Attribut "Require-Bundle" gibt hierbei die Plugins an, welche das vorliegende Plugin benötigt.

```
...
Bundle-SymbolicName: de.oio.bookmarks.ui
Bundle-Version: 1.0.0
Bundle-Vendor: Orientation in Objects GmbH
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime
```

Beispiel 2: Definition der Abhängigkeiten mit Require-Bundle

Um statt RCP nun RAP zu benutzen, wird "org.eclipse.ui" durch "org.eclipse.rap.ui" ersetzt, sodass das Attribut "Require-Bundle" anschließend wie in Beispiel 3 aussieht.

```
...
Require-Bundle: org.eclipse.rap.ui,
 org.eclipse.core.runtime
```

Beispiel 3: Die Abhängigkeiten unter RAP

Den Einstiegspunkt einer RCP-Anwendung bildet der Extension Point "org.eclipse.core.runtime.applications". Bei RAP wird stattdessen der Extension Point "org.eclipse.rap.ui.entrypoint" verwendet, der auf die Besonderheiten der Web-Entwicklung bei RAP eingeht. Beispiel 4 zeigt den Ausschnitt der plugin.xml, um einen Entry Point zu definieren.

```
  <extension
      point="org.eclipse.rap.ui.entrypoint">
    <entrypoint
        class="de.oio.bookmarks.ui.rap.EntryPoint"
        id="de.oio.bookmarks.ui.rap.entrypoint"
        parameter="bookmarks">
    </entrypoint>
  </extension>
```

Beispiel 4: Die Definition des Entry Points einer RAP-Anwendung

Der Extension Point verlangt die Implementierung des Interfaces "org.eclipse.rwt.lifecycle.IEntryPoint". In der Beispiel-Anwendung wurde dieses wie in Beispiel 5 gezeigt, implementiert.

```
public class EntryPoint implements IEntryPoint {
  public Display createUI() {
    Display result = PlatformUI.createDisplay();
    PlatformUI.createAndRunWorkbench( result, new
    BookmarksWorkbenchAdvisor() );
    return result;
  }
}
```

Beispiel 5: Die Implementierung des Entry Points

Die Implementierungen des WorkbenchAdvisor, WorkbenchWindowAdvisor und des ActionBarAdvisor können hierbei wiederverwendet werden.

Die bisher durchgeführten Änderungen müssen bei der Portierung jeder Anwendung vorgenommen werden. Da RAP jedoch nicht alle APIs unterstützt, müssen anschließend noch auftretende Probleme behoben werden. Fehlende APIs machen sich dabei in Eclipse durch Compiler-Fehler bemerkbar, sodass diese leicht erkannt werden können.

Welche typischen Probleme an dieser Stelle auftreten hat das vorherige Kapitel gezeigt. Ist man an diesem Punkt einer Portierung angelangt, so sollte den dort aufgeführten Lösungsansätze wieder einige Aufmerksamkeit zukommen.

Zur Bewertung des nötigen Aufwandes einer Portierung soll folgendes Beispiel aus der Praxis dienen: Es wurde eine Perspektive des RCP-Clients eines wirtschaftlichen Softwaresystems auf RAP portiert. Die Entwicklung dieser Perspektive dauerte ursprünglich ca. 1/4 Entwicklerjahr. In unter einer Woche konnte diese erfolgreich auf RAP portiert werden.

ENTWICKLUNG EINER ANWENDUNG FÜR BEIDE PLATTFORMEN

Der Artikel hat bisher gezeigt, dass eine RCP-Anwendung mit vertretbarem Aufwand auf RAP portiert werden kann. Dieses Kapitel beschreibt die Entwicklung einer Anwendung, die sowohl mit RCP, als auch mit RAP eingesetzt werden kann.

Bei der Entwicklung für beide Plattformen muss man sich mit den gleichen Problemen und Unterschieden beschäftigen, wie bei der Portierung. Hierbei sind jedoch teilweise andere Lösungen notwenig. Beispielsweise muss die RAP-Version der Anwendung auf die Session zugreifen, während eine solche unter RCP gar nicht zur Verfügung steht.

Aber der Reihe nach. Das erste und grundlegendste Problem, mit welchem man sich befassen muss, betrifft die einheitliche Nutzung beider Plattformen. Bei der Portierung wurden die Abhängigkeiten zu RCP-Plugins durch die entsprechenden RAP-Plugins ersetzt. Anschließend kann die Anwendung jedoch nicht mehr mit RCP verwendet werden. Während des Build-Prozesses könnte man aber die Dependencies z.B. durch ein Ant-Skript setzen lassen, sodass die Anwendung in der jeweiligen Distribution auf den korrekten Plugins basiert. Während der Entwicklung muss man bei dieser Methode zum Testen die Dependencies aber immer anpassen, was einen zusätzlichen Aufwand erzeugt.

Zur Formulierung der Abhängigkeiten [3] kann aber auch "Import-Package", statt "Require-Bundle" in der Manifest-Datei verwendet werden. So lassen sich die Abhängigkeiten ohne Bezug auf das implementierende Plugin darstellen. Quellcode 6 zeigt als Beispiel die importierten Packages (Ausschnitt aus der Manifest-Datei) eines Plugins, welches eine View mit SWT und JFace implementiert.

```
Import-Package: org.eclipse.swt,
 org.eclipse.swt.events,
 org.eclipse.swt.layout,
 org.eclipse.swt.widgets,
 org.eclipse.jface.dialogs,
 org.eclipse.jface.viewers,
 org.eclipse.jface.window,
 org.eclipse.ui,
 org.eclipse.ui.part
```

Beispiel 6: Dependencies mit "Import-Package"

Hierdurch erhält das Plugin Zugriff auf die Klassen von SWT und JFace, ohne sich festzulegen, welche Implementierung benutzt werden soll. Hierbei muss man wieder beachten, dass in RAP z.B. nicht alle Widgets enthält, die jedoch auch in diesen Packages zu finden sind. Um einen Einsatz auch mit RAP zu ermöglichen, muss man auf deren Nutzung verzichten.

Als Alternative für die Benutzung von "Import-Package" stehen weitere Lösungen zur Verfügung. Es besteht beispielsweise die Möglichkeit, beide Plattformen als optionale Abhängigkeit zu importieren. Den entsprechenden Ausschnitt der Manifest-Datei zeigt Beispiel 7.

```
Require-Bundle: org.eclipse.ui;resolution:=optional,
 org.eclipse.rap.ui;resolution:=optional,
 org.eclipse.core.runtime
```

Beispiel 7: Optionale Dependencies

Der Vorteil bei dieser Methode ist, dass die Manifest-Datei deutlich übersichtlicher ist. Auch der Wartungsaufwand ist geringer, da mit dem Import von "org.eclipse.ui" bzw. "org.eclipse.rap.ui" auch der Zugriff auf SWT, JFace und den Workbench ermöglicht wird.

Ein weiteres Problem, welches beachtet werden muss ist die unterschiedliche Implementierung von Details bei beiden Versionen der Anwendung. Die Einstiegspunkte beider Versionen werden in zwei unabhängigen Plugins implementiert, da z.B. das Interface IEntryPoint, welches für RAP benötigt wird, bei RCP nicht vorhanden ist. Es gibt jedoch auch häufig weitere Details in einer Anwendung, die unterschiedlich gelöst werden müssen. Es werden nun verschiedene Konzepte vorgestellt, wie man diese umsetzen kann, ohne für einen großen Teil der Anwendung zwei Codebasen zu erhalten.

Eine Möglichkeit der unterschiedlichen Implementierung kann durch zwei Klassen erreicht werden, die die selben Methoden enthalten. Diese Klassen müssen im gleichen Package liegen und den gleichen Namen besitzen. Je ein Plattform-spezifisches Plugin liefert eine Implementierung dieser Klasse und exportiert das entsprechende Package. Dieses Package kann von einem gemeinsamen Plugin importiert und die Klasse benutzt werden. Abbildung 4 zeigt eine schematische Darstellung dieses Prinzips.

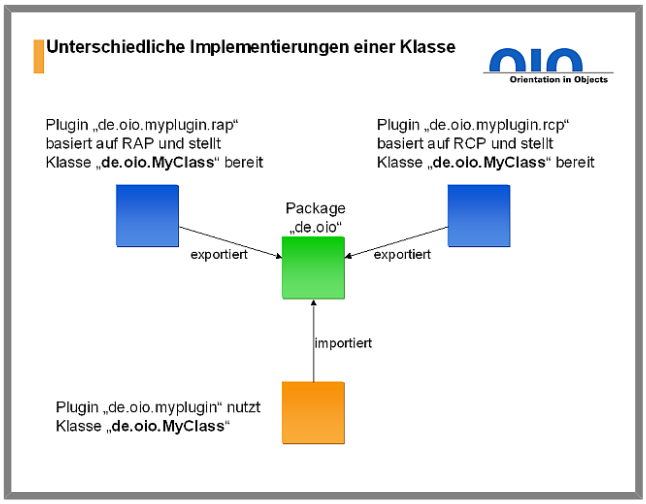

Abbildung 4: Unterschiedliche Implementierungen einer Klasse

Das Prinzip ist hierbei das gleiche, welches die Beispiel-Anwendung verwendet, um beide Plattformen zu importieren. Da man in diesem Fall kein Interface implementiert, muss der Entwickler selbst dafür sorgen, dass beide Klassen zueinander kompatibel sind, was auch den Schwachpunkt hierbei darstellt.

In der Beispiel-Anwendung kommt dieses Prinzip zum Einsatz, um Benutzer-spezifische Daten zu speichern. Die Klasse "DataStore" ist hierbei äußerlich ein Singleton. Im RCP-Plugin ist die Klasse auch als klassischer Singleton implementiert, während bei RAP ein SessionSingleton benutzt wird.

Sollen Details in beiden Versionen unterschiedlich implementiert werden, so lässt sich dies mit einem selbst definierten Extension Point elegant lösen. Die entsprechenden Extensions können auch hier wieder von unabhängigen Plugins geliefert werden, die nur unter einer Plattform genutzt werden.

In der Beispiel-Anwendung wurde der Login-Mechanismus auf diese Art an die Plattform-spezifischen Plugins delegiert. Abbildung 5 zeigt die Definition des Extension Points und dessen Nutzung in der RCP-Version der Anwendung.

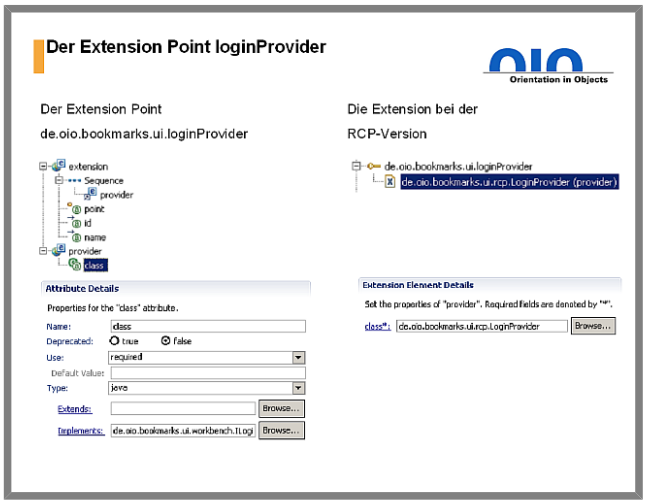

Abbildung 5: Definition eines eigenen Extension Points

Ein gemeinsames Plugin, welches den Login auslöst, benutzt hierfür den entsprechenden LoginProvider. Dieser wird über die Extension Registry ermittelt und instanziert.

Neben einem eigenen Extension Point kann eine unterschiedliche Implementierung auch mit Hilfe eines OSGi Services erreicht werden. Dieser Service wird im gemeinsamen Kern der Anwendung als Interface definiert. Ein Plattform-spezifisches Plugin kann dieses Interface implementieren und über die OSGi Service-Registry veröffentlichen, sodass dieser im Anwendungskern verwendet werden kann.

Das Ergebnis einer solchen Entwicklung ist eine Anwendung, die unter beiden Plattformen ausgeführt werden kann. Die gesamte grafische Oberfläche ist hierbei in gemeinsamen Plugins implementiert, während nur einige Details in spezielle Plugins für eine der beiden Versionen ausgelagert sind.

Abbildung 6 zeigt die Architektur der Beispiel-Anwendung, welche nun unter beiden Plattformen ausgeführt werden kann.

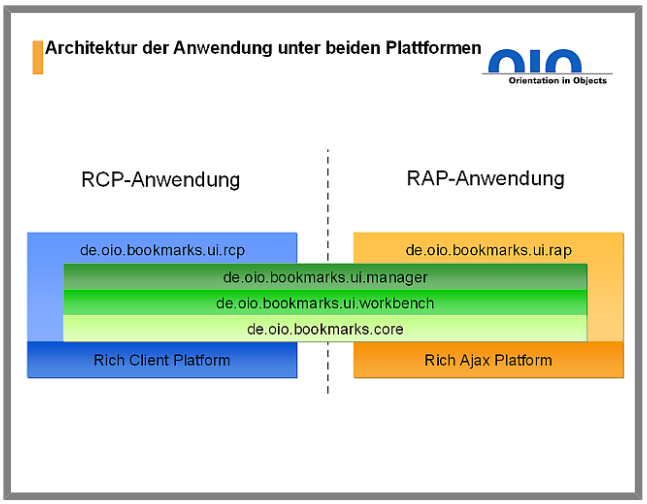

Abbildung 6: Die Architektur der Beispiel-Anwendung unter beiden Plattformen

Wie dieses Kapitel gezeigt hat, können mit etwas Arbeit die Besonderheiten beider Plattformen verborgen werden. Auch unterschiedliche Implementierungen von Teilen lassen sich in beiden Versionen transparent umsetzen.

Die Beispielanwendung können Sie hier als Quellcode herunterladen.

DAS LANGFRISTIGE ZIEL

Mit der Möglichkeit, eine Anwendung für beide Plattformen bereit zu stellen, kann auch das Konzept für die Portierung nochmals überdacht werden. Eine echte Portierung führt zu zwei getrennten Code-Basen, sodass ab diesem Punkt zwei Anwendungen gepflegt werden müssen. Diese enthalten teilweise auch die gleichen Fehler, sodass diese doppelt beseitigt werden müssen.

Die bessere Lösung ist also, die Anwendung so umzuschreiben, dass sie weiterhin in einer RCP-Umgebung eingesetzt werden kann, aber zusätzlich auch unter RAP funktioniert. Hierzu müssen natürlich einige Teile in spezifische Plugins ausgelagert werden, ein großer Teil der Anwendung baut jedoch auf eine gemeinsame Codebasis. Durch die Aufteilung der Anwendung in verschiedene Plugins lassen sich die gemeinsamen und unterschiedlichen Teile sauber trennen.

FAZIT UND AUSBLICK

Das RAP-Projekt hat in den letzten Monaten eine rasante Entwicklung durchlaufen. Nach sechs Milestones im Laufe des Jahres 2007 ist die finale Version 1.0 im Oktober erschienen, die sich zu Recht ein stabiles Release nennen darf. Die aktuelle Version ist 1.0.1.

Die Zielgruppe, welche sich am einfachsten von RAP überzeugen lässt, sind RCP-Entwickler, welche mit geringem Aufwand nun auch Web-Anwendungen entwickeln können. RAP bietet durch seine hohe Kompatibilität zu RCP viele Einsatzmöglichkeiten. Bestehende RCP-Anwendungen können portiert werden. Daneben kann man auch eine Anwendung für beide Zielplattformen entwickeln. Wie der Artikel gezeigt hat, bietet RAP diese Möglichkeiten nicht nur in der Theorie. Es wurde veranschaulicht, wie man mit den dabei auftretenden Problemen umgehen kann und welche Konzepte einem Entwickler zur Verfügung stehen.

Dass hier noch nicht das Optimum erreicht wurde, zeigen einige Aktivitäten bei der aktuellen Entwicklung an der RAP Version 1.1 [5], die zusammen mit Eclipse 3.4 im Juni 2008 erscheinen soll. Hier sollen noch einige existierende Unterschiede zu RCP weiter verringert werden. Z.B. ist geplant, mit diesem Release MouseEvents, zu unterstützen. Des Weiteren sollen die neuen APIs von Eclipse 3.4 ebenfalls unterstützt werden.

Zusammenfassend lässt sich die Entwicklung von RAP mit dem Motto "Evolution statt Revolution" charakterisieren.

REFERENZEN

- [1] Rich Client Platform
  http://www.eclipse.org/rcp/ (http://www.eclipse.org/rcp/)
- [2] Webseite des RAP-Projektes
  http://www.eclipse.org/rap/ (http://www.eclipse.org/rap/)
- [3] Übersicht von Veröffentlichungen über RAP
  http://www.eclipse.org/rap/buzz.php (http://www.eclipse.org/rap/buzz.php)
- [4] Definition von Dependencies
  http://help.eclipse.org/help33/
  (http://help.eclipse.org/help33/topic/org.eclipse.pde.doc.user/guide/tools/editors/manifest_editor/dependencies.htm)
- [5] RAP Projekt-Plan
  http://wiki.eclipse.org/RapPlan (http://wiki.eclipse.org/RapPlan)