HANSER

Leseprobe

Bernhard Steppan

Eclipse Rich Clients und Plug-ins

Modulare Desktop-Anwendungen mit Java entwickeln

ISBN (Buch): 978-3-446-43172-0

ISBN (E-Book): 978-3-446-43316-8

Weitere Informationen oder Bestellungen unter

http://www.hanser-fachbuch.de/978-3-446-43172-0

sowie im Buchhandel.

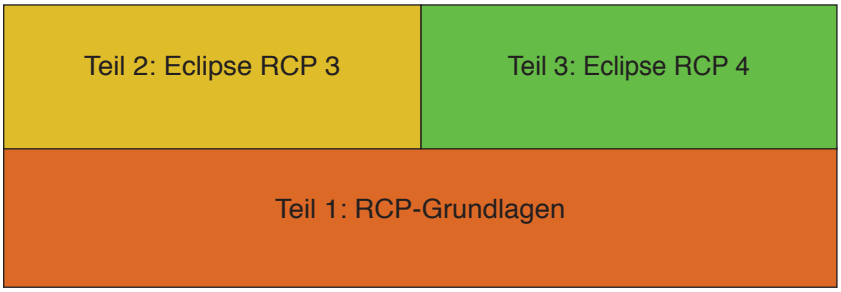
Inhalt

Vorwort

Eclipse – Finsternis, Verdunklung. Hinter diesem geheimnisvollen Namen verbirgt sich eine der leistungsfähigsten Softwareplattformen. Mit ihr lassen sich modulare Desktop-Programme für Windows, Unix und Mac OS X entwickeln. Dieses Buch stellt Ihnen die Eclipse Rich Client Platform (RCP) in drei Teilen vor.

Das Buch startet mit den »RCP-Grundlagen«. Dort geht es um die grundlegenden Eclipse-Konzepte sowie um die UI-Toolkits »SWT« und »JFace«. Im Buchteil »Eclipse RCP 3« dreht sich alles um den schrittweisen Aufbau einer Datenbankanwendung für Eclipse RCP 3. Hier werden die eingangs vorgestellten UI-Bausteine zu einer RCP-Anwendung zusammengesetzt.

Die neue Welt der Eclipse-Programmierung heißt »Eclipse RCP 4«. In diesem Teil steht die neue Eclipse-Plattform im Mittelpunkt. Hier wird gezeigt, wie die zuvor entwickelte RCP-Datenbankanwendung mit Eclipse RCP 4 aussieht. Der Anhang beendet das Buch mit einem Kapitel zur Fehlersuche sowie einem Glossar.



Das Buch deckt Grundlagen und beide Versionen der Eclipse-Softwareplattform ab.

Zielgruppe

Dieses Buch richtet sich an professionelle Softwareentwickler, die die Programmierung mit der Eclipse Rich Client Platform von Grund auf erlernen wollen. Da die Konzepte der Plattform sehr eng mit der Entwicklungsumgebung und Java verbunden sind, sollten Sie die Eclipse-Entwicklungsumgebung kennen und über gute Java-Kenntnisse verfügen.

Schriftkonventionen

Um verschiedene Textteile hervorzuheben, setzt dieses Buch folgende Schriftkonventionen ein:

Textteil	Darstellung
Variablen im Fließtext	*Object*
Variablen in Überschriften	»Object«
Befehle	*Menu → File → Open*
Dateien	*splash.bmp*
Verzeichnispfade	*e3_Application*
URL	*http://eclipse.org*
(...)	Hier fehlt ein Teil des Quellcodes
Verweise auf Literatur oder Internetquellen	[BS1]

Danksagung

Hiermit möchte ich mich bei allen bedanken, die mich beim Schreiben dieses Buchs unterstützt haben: Dem Carl Hanser Verlag und meiner Lektorin Brigitte Bauer-Schiewek für das Vertrauen in meine Arbeit und die große Geduld, der Herstellung des Verlags für die professionelle LaTeX-Vorlage und der Firma Adobe für die Leihstellung des Adobe Illustrators, mit dem sämtliche Grafiken dieses Buchs entstanden sind.

Meine Frau Christiane hat mich wie immer sehr bei diesem Projekt unterstützt. Herzlichen Dank für Deine Hilfe! Danken möchte ich zudem Angéline Feldkamp, die mir bei der Übersetzung der grafischen Oberfläche des RCP-Beispielprojekts »Vocat« ins Französische geholfen hat, und Jürgen Dubau, der das Manuskript sorgfältig Korrektur gelesen hat.

Nicht zuletzt geht mein besonderer Dank an die Entwickler der Eclipse Rich Client Platform. Viele davon arbeiten seit Jahren unentgeltlich an der Verbesserung und Erweiterung der Plattform.

Kontakt

Trotz größter Sorgfalt lässt sich nicht immer verhindern, dass der eine oder andere Fehler in einem Buch übersehen wird. Wenn Sie Fehler finden, Verbesserungsvorschläge oder Fragen haben, senden Sie mir einfach eine Mail an *eclipse_rcp@steppan.net*. Ich werde Ihre Fragen möglichst schnell beantworten und versuchen, Ihre Verbesserungsvorschläge in kommenden Auflagen zu berücksichtigen. Die jeweils aktuellsten Ergänzungen und weitere Informationen finden Sie unter *http://www.steppan.net*. Nun wünsche ich viel Spaß beim Lesen und Entwickeln Ihrer Eclipse-Programme und Plug-ins!

Bernhard Steppan Wiesbaden im Juni 2015

2 Eclipse Rich Client Platform

2.1 Einleitung

Die Eclipse Rich Client Platform (RCP) ist für die Entwicklung portabler Desktop-Anwendungen aufgrund ihrer einzigartigen Konzeption konkurrenzlos. Das folgende Kapitel gibt Ihnen einen Überblick über die Besonderheiten der Eclipse RCP 3 und RCP 4.

2.2 Software-Voraussetzungen

Bitte kontrollieren Sie, ob Sie alle für dieses Kapitel erforderliche Software installiert haben. Dieses Kapitel setzt folgende Software voraus:

- Java Runtime Environment ab Version 7.0
- Eclipse-IDE for RCP and RAP Developers ab Version 4.4
- Beispiele des Verzeichnisses *e2_Rich_Client_Platform*

Die Installationshinweise stehen in Kapitel 1, »Entwicklungsumgebung«, unter dem Abschnitt 1.4 und dem Abschnitt 1.8.

2.3 Historie

Der Name Eclipse steht für viele Dinge, zum Beispiel für die bekannte Java-Entwicklungsumgebung oder die Open-Source-Foundation. Eines der wichtigsten Projekte dieser Eclipse Foundation ist die Eclipse Rich Client Platform. Diese Plattform wurde nicht von Anfang der Eclipse-Geschichte geplant, sondern entwickelte sich aus der Eclipse-Entwicklungsumgebung (siehe Abbildung 2.1). Diese integrierte Entwicklungsumgebung (IDE) wurde 2001 von der IBM als Nachfolger der VisualAge-Produktfamilie vorgestellt. Die Eclipse-IDE sollte einige grundsätzliche Mängel von VisualAge beseitigen und wurde daher architektonisch offener ausgerichtet. VisualAge gehörte zur ersten Generation von integrierten Java-Entwicklungsumgebungen. Diese erste IDE-Generation war ein Meilenstein für die Java-Entwicklung. Sie löste die mühevolle Arbeit mit diversen Editoren und den vollkommen unzureichenden JDK-Werkzeugen wie dem JDK-Debugger ab.

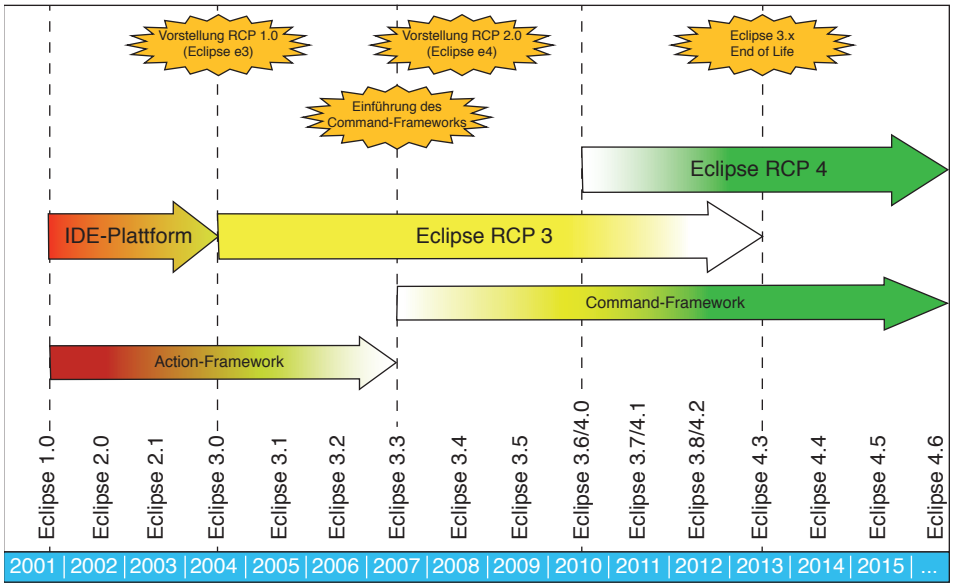


BILD 2.1 Die Geschichte der Rich Client Platform

2.3.1 Vorläufer VisualAge

IBM entwickelte mit VisualAge eine nahezu monolithische Umgebung mit fest eingebautem Editor, proprietärem Compiler, Debugger, Versionskontrolle und Team-Repository. Dieser ganzheitliche Ansatz hatte viele Vorteile. Er brachte aber vor allem bei dem Versuch, externe Werkzeuge zu integrieren, eine Fülle von Schwierigkeiten. Das führte dazu, dass VisualAge immer mehr Marktanteile an offen ausgelegte Entwicklungsumgebungen wie zum Beispiel Borlands JBuilder verlor.

2.3.2 IDE-Plattform

IBM reagierte auf diese Marktentwicklung mit einem Strategiewechsel und der Konzeption der Eclipse-Entwicklungsumgebung. Die neue IDE brach mit dem bisherigen, monolithischen Ansatz der VisualAge-Entwicklungsumgebung, übernahm aber deren positive Eigenschaften wie den speziellen Build-Prozess und die verschiedenen Sichten auf die Java-Sourcen.

Wie radikal die Abkehr von traditionellen Umgebungen war, zeigt Abbildung 2.2. Links ist ein herkömmliches Programm abgebildet, rechts ein RCP-Programm. Ein herkömmliches Programm wie VisualAge besteht aus einem großen, unveränderlichen Kern. Im Gegensatz dazu kann bei einem RCP-Programm wie der Eclipse-IDE fast alles ausgetauscht werden. Die Grenzen bestimmt nur das Plug-in-Design der RCP-Anwendung.

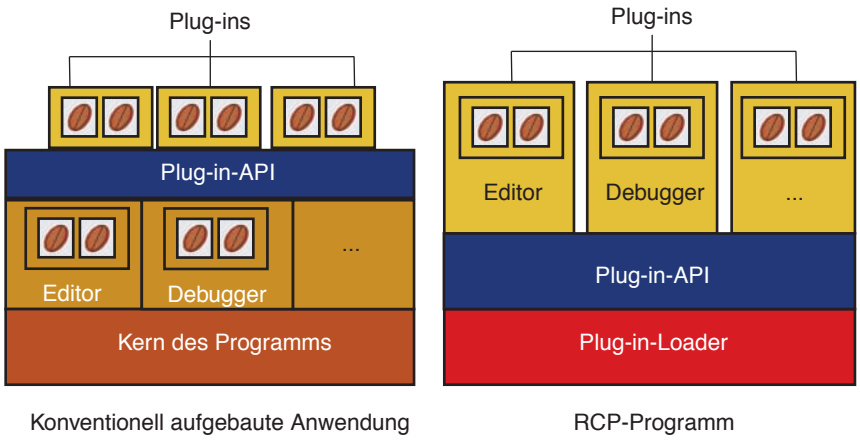


BILD 2.2 Vergleich zwischen einem konventionellen und einem RCP-Programm

2.3.3 Eclipse RCP 3

Nach der Vorstellung des VisualAge-Nachfolgers hat IBM die Eclipse-IDE als Open Source freigegeben und die Eclipse Foundation gegründet. Diese Community koordiniert die Eclipse-Entwicklung und verwaltet die Sourcen. Eclipse war in der Anfangszeit also lediglich eine Plattform für Entwicklungstools – die Eclipse Rich Client Platform lag noch in weiter Ferne.

Erst im Lauf der Zeit hat die Eclipse Foundation erkannt, dass die Eclipse-Frameworks und Eclipse-Konzepte viel zu schade sind, um sie ausschließlich Entwicklungswerkzeugen zu überlassen. Sie lassen sich – wenn man einiges an der damaligen Tool-Plattform Eclipse ändern würde – auch sehr gut in normalen Java-Desktop-Anwendungen verwenden. Man begann daher, die damalige Eclipse-Plattform von der Entwicklungsumgebung zu trennen. Diese gigantische Umstrukturierung des Projekts dauerte einige Jahre.

Mit der Eclipse-Entwicklungsumgebung 3.0 veröffentlichte die Eclipse Foundation dann das Ergebnis: die erste Version der Eclipse Rich Client Platform – kurz Eclipse RCP genannt. Diese 2004 vorgestellte RCP-Version bezeichnet man heute meistens als Eclipse RCP 3 oder einfach Eclipse 3.x. Eclipse RCP 3 bezieht sich darauf, dass diese Version auf der Eclipse 3.x-API aufbaut. Der Großteil der Eclipse-Plug-ins, die heute eingesetzt werden, sind nach diesem RCP-Programmiermodell entstanden. Es wird seit Eclipse 4.3 (Kepler) offiziell nicht mehr weiterentwickelt. Sie erfahren im Buchteil II »Eclipse RCP 3« alles Wesentliche über diese klassische Rich Client Platform.

2.3.4 Eclipse RCP 4

Auf der EclipseCon im Jahr 2008 kündigte die Eclipse-Community Eclipse RCP 4 als nächste Generation der Eclipse Rich Client Platform an. Unter Eclipse RCP 4 ist ein neues, vereinfachtes RCP-Programmiermodell zu verstehen, das auch weiterhin Eclipse RCP 3 unter-

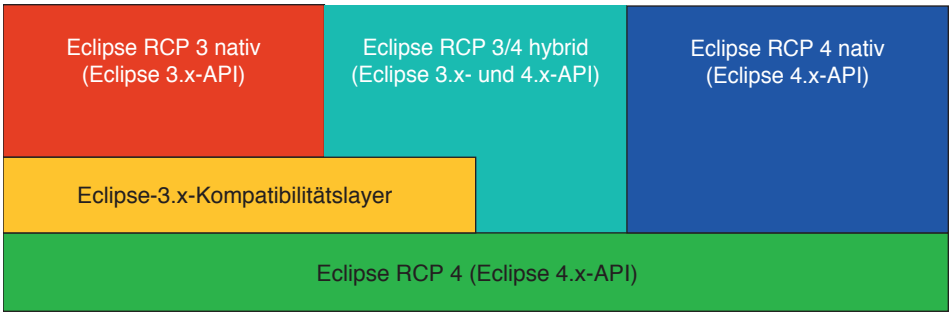


BILD 2.3 Es existieren mehrere Möglichkeiten, ein RCP-Programm zu entwickeln.

stützt (siehe Abbildung 2.3). Wie im Vorwort bereits erwähnt, nenne ich diese Version der Plattform zur besseren Unterscheidung Eclipse RCP 4 oder einfach Eclipse e4. Der Name wurde gewählt, weil das neue Programmiermodell auf der Eclipse 4.x-API aufbaut. Eclipse RCP 4 ist angetreten, die Attraktivität der Rich Client Platform durch folgende Verbesserungen zu steigern:

- UI-Modell: Die grafische Oberfläche wird in einem Applikationsmodell definiert.
- UI-Toolkit: RCP 4 ist unabhängig vom UI-Toolkit. Es lassen sich statt SWT/JFace auch andere UI-Toolkits verwenden.
- UI-Komponenten: Die im Modell enthaltenen UI-Komponenten werden nur deklariert.
- Style Sheets: Das Aussehen der Oberfläche kann, wie bei Webanwendungen, durch Style Sheets (CSS) bestimmt werden.
- JavaScript: RCP-Anwendungen sind in der Lage, JavaScript-Code auszuführen.
- Ressourcenmodell: Entwicklungstools können auf ein flexibleres Ressourcenmodell zurückgreifen.

Für viele RCP-Entwickler kam die neue RCP-Version vollkommen überraschend (siehe [BS01]). Sie fragten sich, welche Möglichkeiten der Entwicklung durch Eclipse RCP 4 bestehen. Es gibt drei (sinnvolle) Modelle, wie ein RCP-Programm heute aussehen kann: entweder als native Anwendung auf Basis der 3.x-API, als Hybridanwendung unter Verwendung der Eclipse 3.x-API und Eclipse 4.x-API oder als native Anwendung auf Basis der Eclipse 4.x-API. Wie eine Hybridanwendung und eine native Eclipse-4.x-Anwendung aussieht, erfahren Sie im Buchteil III, »Eclipse RCP 4«.

2.4 Programmiermodell

Ob Eclipse RCP 3 oder Eclipse RCP 4 – RCP-Anwendungen wie die Eclipse-IDE bestehen fast nur aus Plug-ins. Plug-in ist die Bezeichnung in der Eclipse-Community für das, was die Software-Plattform OSGi (siehe Abschnitt 2.5) als Bundle bezeichnet. Wie revolutionär der Ansatz war, eine Anwendung fast nur noch aus solchen Modulen aufzubauen, sehen Sie an der Architektur der Eclipse-IDE (siehe Abbildung 2.4).

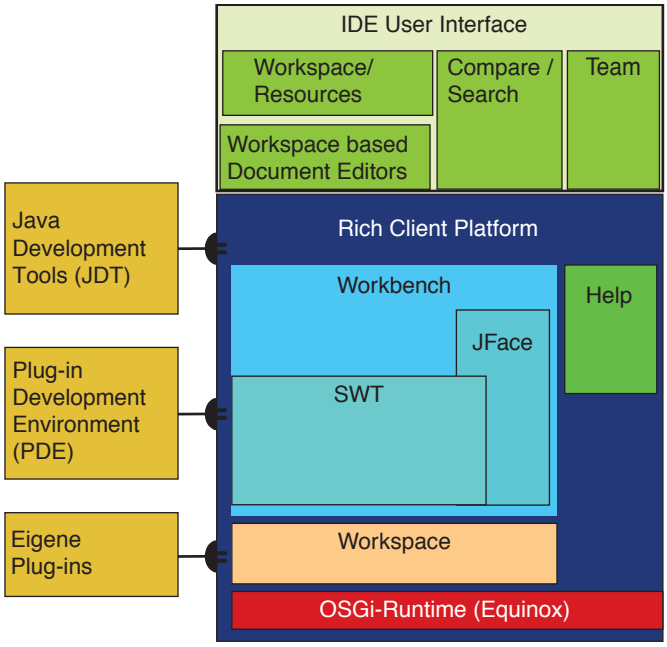


BILD 2.4 Die Architektur einer RCP-Anwendung am Beispiel der Eclipse-IDE

Die Eclipse-IDE ist nichts anderes als eine sehr große Anwendung auf Basis der Eclipse Rich Client Platform. Die Abbildung 2.4 zeigt am Beispiel der Eclipse-IDE für RCP- und RAP-Entwickler, dass eine solche Anwendung im Wesentlichen aus einem sehr kleinen RCP-Kern und diversen Plug-ins wie den Java Development Tools sowie der Plug-in Development Environment besteht. Eigene Plug-ins können Sie installieren, um das Programm nach Ihren Wünschen zu ergänzen. Um die Entwicklung von solchen komplexen Desktop-Anwendungen wie der Eclipse-IDE so einfach wie möglich zu halten, beinhaltet die Eclipse Rich Client Platform eine ganze Reihe von architektonischen Besonderheiten, allen voran das eingangs erwähnte Modulsystem.

2.5 Module

2.5.1 OSGi und Equinox

2.5.1.1 Java-Standard Jigsaw

Über ein dynamisches Modulsystem wird in Java-Kreisen seit Jahren kontrovers diskutiert. Vielleicht ist Ihnen der Begriff »Jigsaw« (engl.: Stichsäge) daher bekannt. Es ist der Codename des zukünftigen Java-Modulsystems. Dieser wichtige Modulmechanismus fehlt zum Zeitpunkt der Drucklegung dieses Buchs noch in Java. Er ist für Java 9 angekündigt (siehe [JIGS]). Warum ist ein solcher Mechanismus so wichtig, und was steckt hinter einem dynamischen Modulsystem?

Ein dynamischer Modulmechanismus ist dazu da, *unterschiedliche* Versionen eines Moduls (= Bundles, Plug-ins) in einem Programm zu verwalten, Module kontrolliert zur Laufzeit zu laden und auch wieder zu entfernen. Mit reinen Java-Sprachmitteln wie Packages ist so etwas nur über zusätzliche Programmierung möglich. Sie können durch das Package-Konzept zwar beispielsweise mehrere String-Klassen in einem Programm einsetzen wie zum Beispiel *java.util.String* und *java.lang.String*. Es kommt zu keinen Konflikten, weil sich die Klasse *String* in unterschiedlichen Packages befindet. Daher ist es nicht ohne Weiteres möglich, die Klasse *java.lang.String* in mehreren Versionen innerhalb einer Java-Anwendung zu verwenden.

2.5.1.2 Versionskonflikte

Gerade in größeren Enterprise-Anwendungen kommt es daher häufig zu Versionskonflikten, die durch Programmbibliotheken ausgelöst werden, die man verwenden muss, in denen sich aber gleichnamige Klassen mit unterschiedlichen Versionsständen befinden. Ein Beispiel: Gesetzt den Fall, Sie verwenden mehrere Bibliotheken in einer Anwendung, weil sie funktional notwendig sind. Mehrere dieser Bibliotheken benötigen eine Klasse mit exakt dem gleichen vollqualifizierten Namen – aber sie setzen unterschiedliche Versionen voraus.

Angenommen, wie in Abbildung 2.5 dargestellt, die Klasse *c1.class* in der Version 0.3.1 wird zuerst vom Java-Classloader geladen. Je nach Implementierung des Java-Classloaders steht ausschließlich *diese* Version der Klasse ab diesem Zeitpunkt *allen Verbrauchern* zur Verfügung, die ein Objekt dieser Klasse erzeugen möchten. Das gilt auch für den Fall, dass diese Verbraucher eine Klasse in der Version 1.0.1 erwarten, da es keine Möglichkeit gibt, vom Classloader genau diese Version anzufordern.

Wenn sich die Implementierung von c1-0.3.1 und c1-1.0.1 deutlich geändert hat, kommt es zu einem schwer verständlichen Fehlverhalten des Programms. Diese »Anomalien« kön-

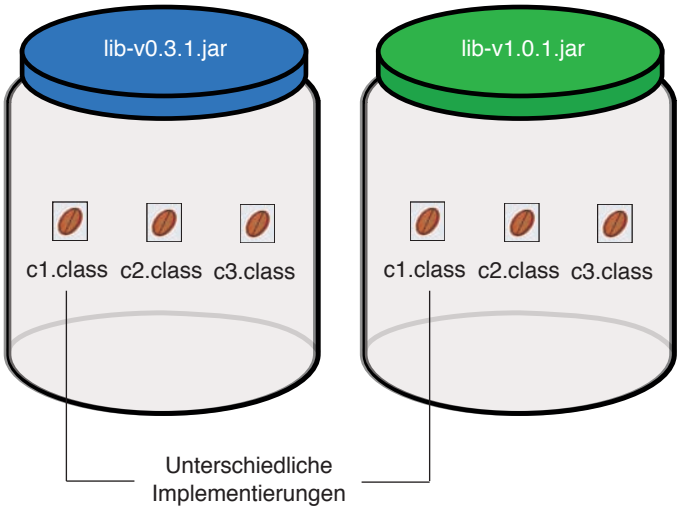


BILD 2.5 Versionskonflikte bei gleichen Klassen mit unterschiedlicher Implementierung

nen nur durch aufwendige Tests gefunden werden. Sie sind für jeden Tester ein Albtraum. Daher ist ein dynamischer Modulmechanismus mit einer Beherrschung von Versionskonflikten und dem dynamischen Laden und Entfernen von Bundles für komplexere Programme ein Muss. So sahen das auch die Entwickler der Eclipse-IDE.

2.5.1.3 Equinox

Als die Eclipse-IDE – und somit auch die Eclipse Rich Client Platform – entstand, konnte man nicht warten, bis die Java-Community sich auf eine Implementierung für einen dynamischen Modulmechanismus geeinigt hatte. Stattdessen gab es eine Spezifikation für eine solche Modulplattform namens OSGi. OSGi stand früher für »Open Services Gateway Initiative« und ist heute ein internationaler Standard für Modulplattformen. Es gibt eine Implementierung der Apache Group namens »Felix« und eine Implementierung der Eclipse Foundation namens »Equinox«. Die Eclipse Rich Client Platform setzt Equinox ein.

Das Laufzeitsystem auf Equinox-Basis lässt zu, dass Bundles (Plug-ins) in unterschiedlichen Versionen innerhalb eines Programms koexistieren, dynamisch geladen und entfernt werden können (Abbildung 2.6). Sie fordern einfach in ihrem Programm oder Plug-in eine bestimmte Version an oder legen fest, was die Mindestversion sein soll. Damit ist gewährleistet, dass das Programm exakt so funktioniert, wie es auch getestet wurde.

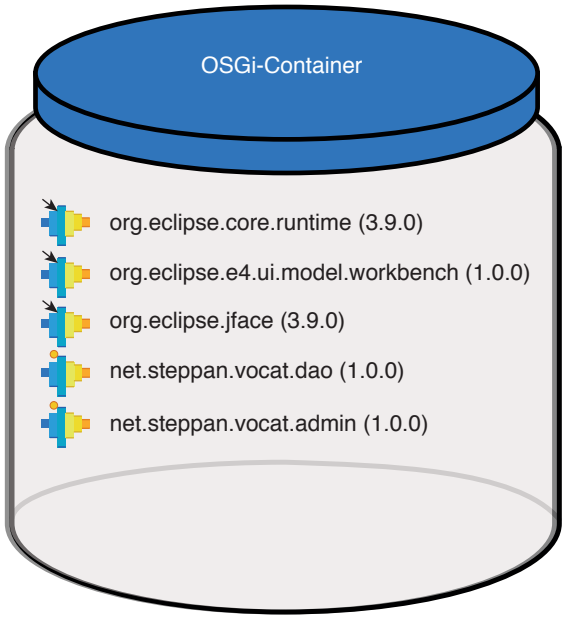


BILD 2.6 In einem OSGi-Container können Plug-ins unterschiedlicher Versionen koexistieren.

2.5.1.4 Service Registry

Damit ein OSGi-Framework Plug-ins unterschiedlicher Versionen verwalten kann, verfügt es über eine Service Registry. Das ist eine Art von Datenbank, wo Bundles mit ihrer Version eingetragen werden. Im Fall von Eclipse registriert sich eine RCP-Anwendung oder ein

Eclipse-Plug-in in einer solchen Datenbank. Sie befindet sich normalerweise in dem Verzeichnis des RCP-Programms wie zum Beispiel der Eclipse-IDE.

Ein Plug-in wird nicht einfach in einen Container geworfen und verharrt dort ein Leben lang ohne Änderungen. Es verfügt über einen Lebenszyklus mit einem definierten Beginn und Ende. Jedes RCP-Plug-in verfügt normalerweise über eine Klasse mit Methoden, um ein Modul zu starten und zu stoppen. Im Plug-in-Lebenszyklus gibt es folgende sechs Zustände: Installed, Resolved, Starting, Stopping, Active and Uninstalled (Abbildung 2.7).

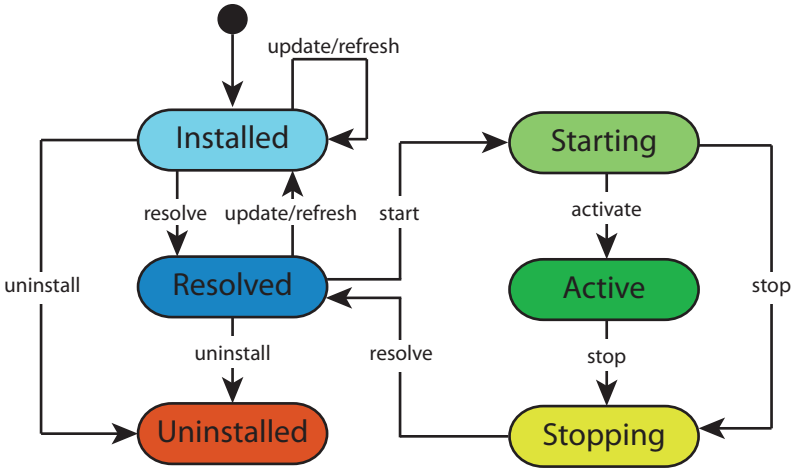


BILD 2.7 Plug-ins durchlaufen einen Life Cycle (Quelle: Eclipse Foundation/Virgo).

Installed bedeutet, dass das Plug-in bisher lediglich zum Container hinzugefügt wurde. Es ist weder im Resolved- noch im Starting-Zustand. Wenn sich der Zustand dieses Plug-ins nicht verändert, kann das bedeuten, dass Abhängigkeiten noch nicht aufgelöst wurden oder ein anderes Plug-in mit dem gleichen Namen bereits ausgewählt wurde. In diesem Zustand können keine Packages oder Extensions dieses Plug-ins genutzt werden.

Der Zustand *Resolved* bedeutet, dass alle Voraussetzungen für das Plug-in erfüllt wurden, es aber noch nicht gestartet wurde. Das Plug-in kann der Rich Client Platform Packages, Extensions und Extension Points zur Verfügung stellen. Wenn verlangt wurde, dass das Plug-in starten soll, bekommt es den Zustand *Starting*. Das passiert beim Programmstart. Ist es betriebsbereit und gestartet, nimmt es den Zustand *Active* ein.

Beim Beenden eines Programms wird das Plug-in gestoppt und bekommt den Zustand *Stopping*. Danach liegt es wieder als *Resolved* im Container. Wird es nicht benötigt, kann es deinstalliert werden und bekommt den Zustand *Uninstalled*. In diesem Zustand kann es nicht vom Programm aus verwendet werden.

2.5.2 Eclipse-Plug-ins

OSGi-Plug-ins wurden eingeführt, um Eclipse-Programme flexibel aufbauen und erweitern zu können. Solche Modulsysteme gab es in Anwendungen schon früher, aber selten so durchdacht. Während normale Anwendungen immer einen Grundanteil fest integrier-

ter Funktionen mitschleppen müssen, ist bei einem RCP-Programm fast alles austauschbar. Wie gut das funktioniert, hängt nur vom Design der Plug-ins ab.

Die Modularisierung geht so weit, dass RCP-Entwickler den Spruch geprägt haben: »Alles ist ein Plug-in«. Das stimmt fast. Der Plug-in-Loader ist zum Beispiel eine Ausnahme. Er ist der Minikern, der zuerst ausgeführt wird, wenn Sie eine Eclipse-Anwendung wie zum Beispiel den Vokabeltrainer dieses Buchs starten. Der Plug-in-Loader hat die Aufgabe, alle erforderlichen Plug-ins zu laden. Hierbei kann er erkennen, ob Plug-ins »zusammenpassen«. Das Modulsystem ist also in der Lage, Versionskonflikte zu erkennen.

Dieser durchdachte Mechanismus ist viel zu schade, um ihn ausschließlich für Entwicklungswerkzeuge zu nutzen. Daher kam die Eclipse-Community auf die Idee, die Eclipse-Plattform als Grundlage »normaler« (Geschäfts-) Anwendungen und nicht nur als Grundlage für Eclipse-Entwicklungswerkzeuge zu nutzen: Eine universelle Java-Plattform für Desktop-Anwendungen war entstanden, die im Laufe der Zeit immer weiter verbessert wurde. Mit der Eclipse-Entwicklungsumgebung 3.0 und der Vorstellung der ersten Version der Rich Client Platform haben sich viele Firmen entschlossen, ihre Rich Clients auf Basis von Eclipse 3.x zu entwickeln. Diese erste Version der Eclipse Rich Client Platform bezeichnet man daher als Eclipse RCP 3 bzw. Eclipse 3.x (siehe Teil II, »Eclipse RCP 3«).

Heute lassen sich mithilfe der Rich Client Platform Anwendungen entwickeln, denen nur Fachleute ansehen, dass sie diese Plattform als Grundlage verwendet haben. Den Kunstgriff erreichten die Eclipse-Entwickler dadurch, dass sie die Teile der Rich Client Platform, die für die Entwicklungsumgebung benötigt werden, und die allgemein verwendbaren Teile der Rich Client Platform im Laufe der Zeit konsequent getrennt haben. Zudem können viele UI-Komponenten der Plattform über CSS in Farbe und Form verändert werden.

2.6 Workspace

Wie speichert man Programmeinstellungen dauerhaft? Hier gibt es verschiedene Konzepte: Der Eclipse-Vorläufer VisualAge verwendete zum Beispiel ein zentrales Repository, eine Art Archivdatenbank für alle Projektbestandteile. Der Vorteil war, dass damit auch Teams zusammenarbeiten konnten, weil eine Versionierung des Quellcodes automatisch sichergestellt war. Man erkaufte sich damit aber den Nachteil, dass man nicht gleichzeitig mit anderen Werkzeugen auf Projektdateien zugreifen konnte, die in diesem datenbankähnlichen, monolithischen Speicher auf einem Server lagen. Man musste den Inhalt exportieren, bearbeiten und wieder importieren – ein umständliches Verfahren.

Ein flexibleres Konzept ist es, die Programmeinstellungen in einer oder mehreren Dateien auf dem Arbeitsplatzrechner zu speichern, wie es bei verschiedenen Entwicklungsumgebungen der Fall ist. Eine RCP-Anwendung wie die Eclipse-IDE organisiert ihre Programmeinstellungen in einem kompliziert strukturierten Verzeichnis auf der lokalen Festplatte, dem sogenannten »Workspace« (Abbildung 2.8).

Der Workspace eines RCP-Programms ist wie bei der VisualAge-IDE als Repository organisiert. Im Gegensatz zum Repository-Konzept von Visual Age hat dieses RCP-Repository jedoch den Vorteil, dass es offen für andere Werkzeuge ist. Es hat allerdings den Nachteil, dass es ohne weitere Ergänzungen nicht geeignet ist, mit mehreren Anwendern gleichzeitig

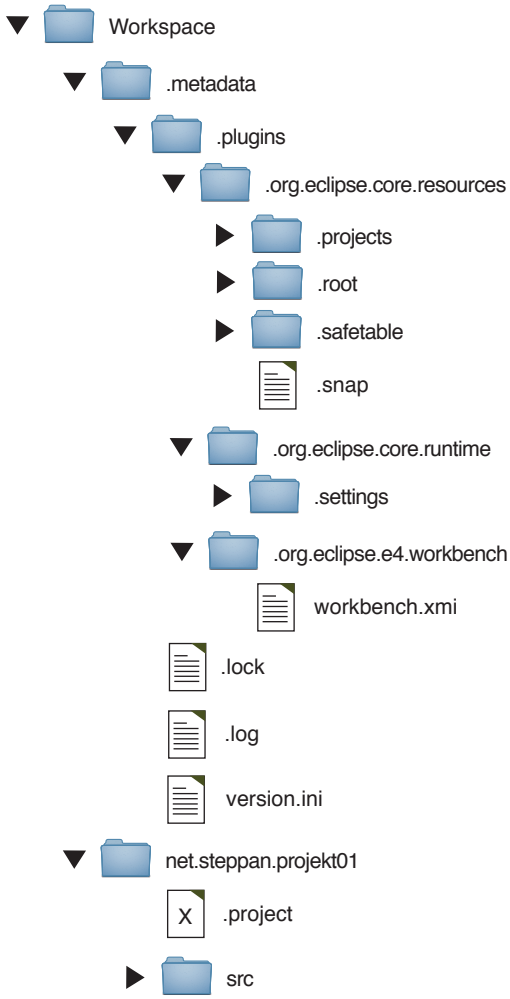


BILD 2.8 Der Aufbau des Eclipse-Arbeitsbereichs

an einem Projekt zu arbeiten. Um eine solche Teamfähigkeit zu erreichen, ist ein zusätzlicher Server für die Versionierung von Projektdateien notwendig, zum Beispiel CVS oder SVN. Mit ihm lassen sich Änderungen, die an einem Projekt lokal auf den einzelnen Workstations vorgenommen werden, mit anderen Teammitgliedern austauschen.

Der Workspace enthält unter anderem ein verstecktes Verzeichnis mit Metadaten. Der Inhalt dieses Verzeichnisses ist abhängig von der Konfiguration des Eclipse-Programms. In der Regel besteht das Verzeichnis aus den drei Dateien `.lock`, `.log` und `version.ini` sowie dem Verzeichnis `.plug-ins`. Die versteckte Lock-Datei innerhalb des Arbeitsbereichs zeigt der Eclipse-Anwendung an, ob sich der Workspace bereits im Zugriff eines anderen Eclipse-Programms befindet, während die Log-Datei die vom Programm gespeicherten Logging-Informationen bereithält.

LISTING 2.1 Ermittlung der Projekte innerhalb eines Workspaces

```
1  ArrayList<Project> projects = new ArrayList<Project>();
2  (...)
3  IWorkspace workspace = ResourcesPlugin.getWorkspace();
4  // Root ermitteln
5  IWorkspaceRoot root = workspace.getRoot();
6  System.out.println("Workspace '" +
7    Platform.getInstanceLocation().getURL() + "' wird eingelesen...");
8  // Alle Projekte finden:
9  IProject[] projects = root.getProjects();
10 // Das Modell befuellen:
11 for (IProject project : projects) {
12   try {
13     project.open(null); // Null: keine Fortschrittsanzeige
14     projects.add(new Project(project.getName(),
15                              project.getFullPath().toString(),
16                              project.getDescription().getComment()));
17     System.out.println("Projekt eingelesen: " +
18       project.getDescription().getName());
19   } catch (CoreException e) {
20     e.printStackTrace();
21   }
22 } //for
```

Der Inhalt des Plug-in-Verzeichnisses variiert sehr stark in Abhängigkeit der Konfiguration des RCP-Programms. Hier finden Sie die Einstellungen der einzelnen Eclipse-Module in Ordnern, deren Namen der Bezeichnung des jeweiligen Plug-ins entsprechen. Wenn Sie ein Plug-in installieren, wird es im Workspace ein neues Verzeichnis anlegen und seine Informationen speichern. Zum Beispiel speichert die RCP-Laufzeitumgebung ihre Informationen in dem Verzeichnis *org.eclipse.core.runtime*.

Die Eclipse Rich Client Platform stellt jedem RCP-Programm Java-Klassen zum Zugriff auf den Arbeitsbereich zur Verfügung, sodass die RCP-Anwendung die Struktur des Workspace nicht im Detail kennen muss. Das Listing 2.1 zeigt ein Programmfragment, das den Workspace mit seinen Projekten einliest und einem Array zuordnet. In Zeile 3 ermittelt das Programm den gesetzten Workspace. Er kann über den Parameter *-data* in der Ini-Datei des RCP-Programms übermittelt werden. Danach holt sich das Programm die Wurzel des Workspaces und von dort alle Projekte. In der nachfolgenden Schleife werden diese Projekte samt Projektnamen, Pfad und Kommentar ausgelesen.

Eine RCP-Anwendung ist in der Lage, mehrere Workspaces zu verwalten. Sie kann den Speicherort auf der Festplatte selbst bestimmen oder vom Anwender bestimmen lassen. Beim Wechsel des Workspaces ist bisher jedoch ein Neustart erforderlich. Aufgrund der Struktur des Arbeitsbereichs kann nicht jedes RCP-Programm immer den Workspace eines anderen Programms vollständig auslesen. Das funktioniert nur dann, wenn die Konfiguration des RCP-Programms, das ihn einlesen möchte, kompatibel zu dem Programm ist, das ihn geschrieben hat. Ist das nicht der Fall, können nicht alle Informationen interpretiert werden, und es kommt zu einem Fehler. Die Daten bleiben hierbei aber erhalten.

2.7 UI-Toolkits

Im Gegensatz zu traditionellen Java-Programmen verwendeten die Eclipse-Erfinder bei der Konzeption der Eclipse-Entwicklungsumgebung nicht die UI-Bibliotheken AWT und Swing der Java-Plattform für die grafischen Oberflächen von Eclipse-Programmen, sondern entwickelten zwei neue UI-Toolkits: das Standard Widget Toolkit (SWT) und das darauf aufbauende JFace. Die Gründe für diesen nicht unumstrittenen Schritt liegen in der Unzufriedenheit mit dem Zeitverhalten und dem Speicherkonsum der Swing-Bibliothek.

Während die Swing-Bibliothek alle Oberflächenbausteine selbst zeichnet, ist das Eclipse-Framework SWT eine sehr systemnahe GUI-Bibliothek: Es delegiert das Zeichnen der Widgets über das Java Native Interface (JNI) an das Betriebssystem. Dieser andere Ansatz führt natürlich zu einem wirklich nativen Erscheinungsbild – auch dann, wenn das Look and Feel des Betriebssystems sich durch eine neue Version ändert. Zudem wird in der Regel die Oberfläche einer Eclipse-Anwendung schneller aufgebaut, weil die Zeichenfunktionen des Betriebssystems denen der Swing-Bibliothek überlegen sind.

Wenn Sie RCP-3-Programme und RCP-3-Plug-ins entwickeln, müssen Sie die UI-Toolkits SWT und JFace einsetzen. Das gilt nicht für Eclipse RCP 4, wo Sie zum Beispiel auch ausschließlich JavaFX oder Swing verwenden können. Die nachfolgenden vier Kapitel »UI-Toolkits«, »Standard Widget Toolkit«, »JFace« und »Layoutmanager« geben Ihnen einen tiefen Einblick in die Einzelheiten der beiden klassischen RCP-Toolkits SWT und JFace.

2.8 Workbench

Wenn Sie mit der Eclipse-Entwicklungsumgebung oder einer anderen RCP-Anwendung arbeiten, ist Ihnen das spezielle Layout der Oberfläche aufgefallen, das sich von traditionellen Desktop-Anwendungen durch einige Neuerungen unterscheidet. Im Eclipse-Jargon spricht man von der Workbench, von Views (Sichten), Editoren (Eingabefenster) und Perspektiven. Die Workbench ist der zentrale Teil einer RCP-Anwendung (siehe Abbildung 2.9). Sie fasst die einzelnen Workbench-Bestandteile wie Fenster, Menüs, Symbolleisten und Statusleisten zusammen. Diese Workbench lässt sich dynamisch über Plug-ins erweitern.

2.8.1 Parts

Fast jede Anwendung benötigt Fenster, die den Inhalt lediglich anzeigen (Views) und Fenster, deren Inhalt man verändern kann (Editoren). Diese Fenster bezeichnet Eclipse RCP als Parts. Eclipse RCP 3 gibt vor, dass solche Views von der Basisklasse *ViewPart* und Editoren von der Basisklasse *EditorPart* abgeleitet werden müssen. Unter Eclipse RCP 4 sind Parts lediglich Basisklassen und können beide Rollen einnehmen.

Views und Editoren ist gemeinsam, dass man solche Fenster vom Hauptfenster abreißen und an anderer Stelle neu platzieren kann. RCP-Fenster lassen sich in einem Fensterbereich übereinander anordnen, sodass von ihnen nur die Register zu sehen sind. Solche Fenster können geschlossen oder auch so konfiguriert werden, dass man sie nicht schließen kann. Zudem lassen sie sich zu Perspektiven zusammenstellen.

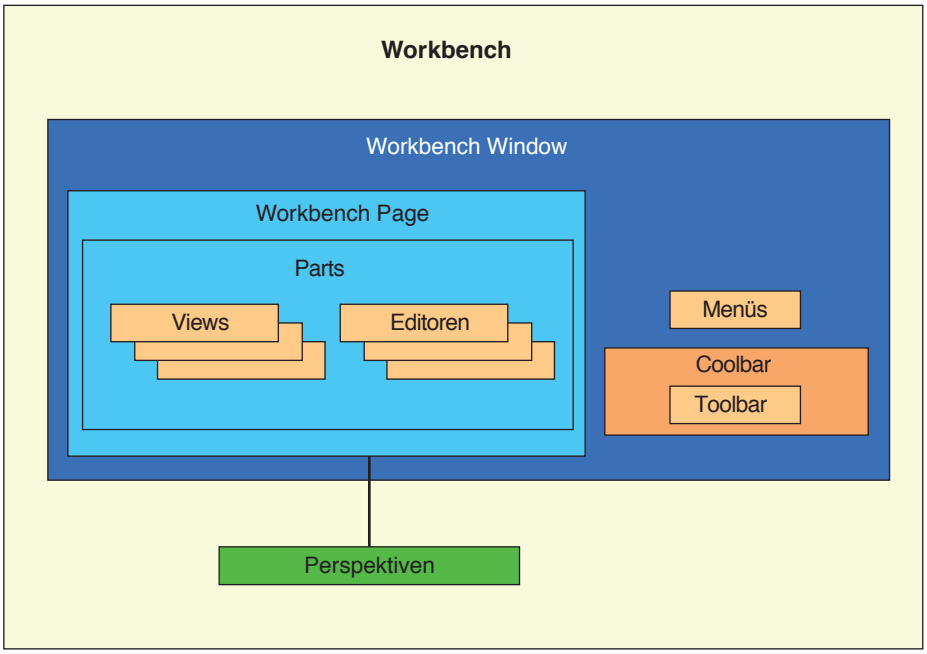


BILD 2.9 Die Workbench ist die Arbeitsoberfläche einer RCP-Anwendung.

2.8.2 Perspektiven

Sie kennen Perspektiven sicher bereits von der Eclipse-Entwicklungsumgebung. Eine Perspektive ist nichts anderes als ein festgelegtes Fensterlayout mit einer Auswahl bestimmter Parts. Eine RCP-Anwendung wie das Programmierbeispiel dieses Buchs kann verschiedene Perspektiven für unterschiedliche Arbeiten mit dem Programm definieren (siehe Abbildung 2.10). Das Programm speichert nicht nur, welche Parts verwendet werden. Es speichert auch die Position und Anordnung der Fenster in dem schon erwähnten Arbeitsbereich auf der Festplatte (siehe Abschnitt 2.6, »Workspace«).

Ein Eclipse-Programm wird mit mindestens einer Perspektive ausgeliefert. Sie sind der Vorschlag der Anwendungsentwickler, welche Parts (Views und Editoren) zusammenpassen und wie sie sinnvoll angeordnet werden. Von der Wahl des Layouts ist abhängig, ob Parts übereinander oder nebeneinander liegen und veränderbar in ihrer Größe sind. Viele RCP-Programme erlauben dem Anwender darüber hinaus, Perspektiven nach Belieben zu verändern, Parts zu schließen und andere Parts hinzuzufügen. Zudem lassen sich eigene Perspektiven erzeugen und unter einem Namen dauerhaft speichern.

Dadurch, dass sich die Änderungen am Fensterlayout dauerhaft auf der Festplatte speichern und auf Knopfdruck wieder aktivieren lassen, kann ein Anwender sein RCP-Programm sehr flexibel an seine bevorzugte Arbeitsweise anpassen. Da diese Einstellungen im Workspace gespeichert werden, sind sie normalerweise unabhängig von einer bestimmten Version eines RCP-Programms. Das heißt, selbst wenn der Anwender eine komplett neue Version seines RCP-Programms installiert, kann er in der Regel wieder sofort mit der vertrauten Anordnung seines Programms arbeiten, wenn er den gleichen Arbeitsbereich wählt.

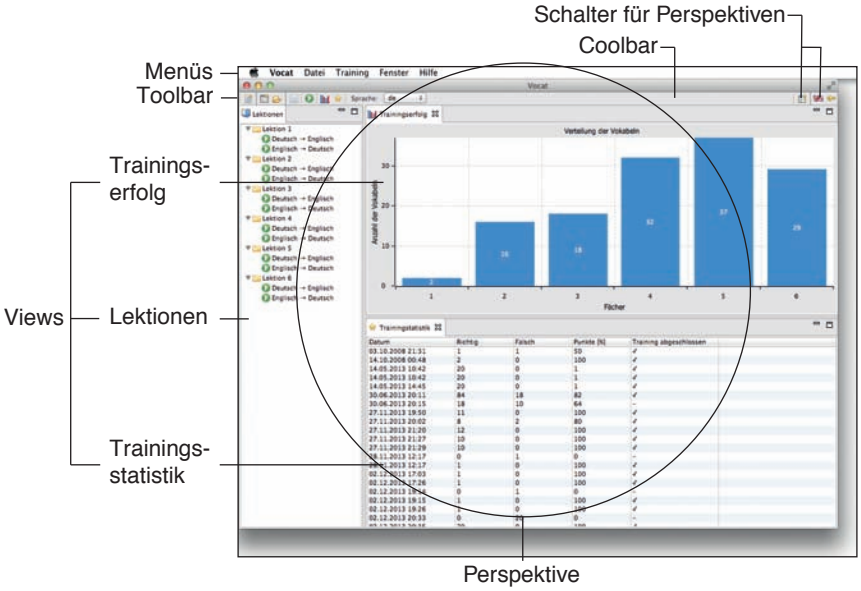


BILD 2.10 Vocat 2.0 – das Eclipse-RCP-4-Beispiel dieses Buchs

2.8.3 Features

Features könnte man wörtlich als fachliche Besonderheit oder Charakteristikum eines Programms übersetzen. Rein technisch versteht man darunter im Rahmen der RCP-Programmierung eine Gruppe von zusammengehörigen Plug-ins. Features helfen vor allem bei wirklich großen Projekten, die Fülle von Plug-ins sinnvoll zu strukturieren und später auf dem Computer des Endanwenders zu verteilen. Das ist nützlich, um sehr große Programmerweiterungen, zum Beispiel für die Eclipse-Entwicklungsumgebung, überschaubar zu halten. Über Features wird zudem die Softwareaktualisierung gesteuert.

2.9 Softwareaktualisierung

Um Eclipse-Anwendungen nach der Installation auf dem neuesten Stand zu halten, stellt die Rich Client Platform bzw. die OSGi-Implementierung Equinox einen Update-Mechanismus (auch kurz »p2« genannt) bereit. Die Bezeichnung p2 steht hierbei für Provisioning 2.x (engl. Provisioning: Versorgung, Beschaffung). Die Version 2.x kommt dadurch zustande, dass es schon einen Vorläufer gab. Durch diese Update-Funktion lassen sich neue Versionen von Plug-ins (Bundles) oder vollständig neue Plug-ins automatisch oder manuell nachladen und wieder entfernen. Das geschieht entweder über eine Update-Website (Cloud), ein Laufwerk oder über ein Archiv. Mehr dazu in Kapitel 19, »Verteilung«, für Eclipse RCP 3 und in Kapitel 31, »Verteilung«, für Eclipse RCP 4.

2.10 Hilfesystem

Ein weiterer wichtiger Pluspunkt der Eclipse Rich Client Platform ist das plattformübergreifende Hilfesystem. Eclipse-Anwendungen können mit einer globalen Hilfeunterstützung ausgeliefert werden, die wie eine eigenständige Webanwendung im Browser angezeigt wird. Zudem lässt sich jeder Dialog mit einer kontextbezogenen Hilfe ausstatten. Eine weitere Möglichkeit der Hilfeunterstützung sind Willkommensseiten mit Tutorien (sogenannte »Intro«), die man ebenfalls sehr einfach in eine RCP-Anwendung integrieren kann.

2.11 Remote Application Platform

2.11.1 Webprogrammierung

Als Abschluss dieses Kapitels möchte ich Ihnen noch einen Ausblick auf die Webprogrammierung nach dem RCP-Programmiermodell geben. RCP für das Web hört sich erst einmal völlig abwegig an. Alle Bibliotheken, die im RCP zum Einsatz kommen, sind auf die clientseitige Programmierung für Desktop-Anwendungen zugeschnitten. Mit serverseitiger Programmierung hat RCP also nichts zu tun.

Bei der Programmierung mit der Remote Application Platform geht es auch nicht darum, die Rich Client Platform eins zu eins auf das Web zu übertragen. Es geht vielmehr darum, das RCP-Programmiermodell für Webanwendungen zu nutzen und im Gegensatz zu Java-Server Faces eine sehr hohen Abstraktion von der üblichen Webprogrammierung zu erzielen. Das relativ neue Eclipse-Projekt nannten die Entwickler zunächst Rich Ajax Platform (RAP), da die verwendeten Widgets Ajax einsetzen. Später wurde die Plattform in Remote Application Platform umbenannt (siehe [BS02]).

2.11.2 RAP Widget Toolkit

Um die Rich Client Platform auf das Web zu übertragen, musste neben vielen anderen RCP-Funktionen das Standard Widget Toolkit (SWT) gegen ein Framework ausgetauscht werden, das die Low-Level-Kommunikation mit dem Servlet-Container übernimmt (Abbildung 2.11). Dieses Framework heißt RAP Widget Toolkit (RWT).

SWT und RWT sind sich sehr ähnlich, aber im Detail verschieden. Zum Beispiel verwenden RWT-Komponenten den Namespace *org.eclipse.rap.rwt* statt *org.eclipse.swt*. Da es bestimmte Stile der SWT-Widgets in einer Webanwendung nicht gibt, unterscheiden sich auch hier die beiden Komponentenbibliotheken. RAP verfügt ebenfalls über eine spezielle JFace-Version. Sie stellt Viewer, Webdialoge, Actions und eine Image Registry zur Verfügung.

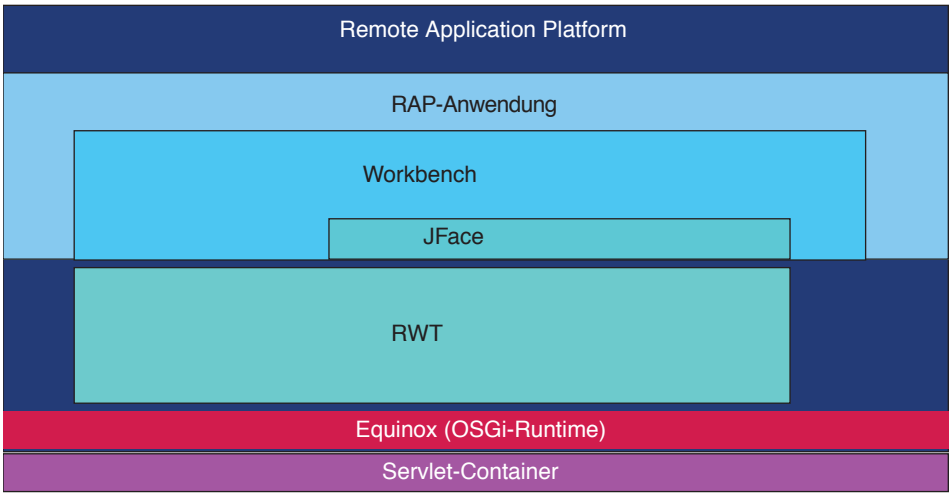


BILD 2.11 Die Architektur der Remote Application Platform

2.11.3 Modulare Webanwendungen

Teil der RAP-Architektur ist – wie bei RCP – eine OSGi-Runtime sowie die Workbench als Klammer der Web-UI. Der Aufbau eines RCP- und eines RAP-Programms ist sehr ähnlich (siehe Abbildung 2.11). Dort ist zu sehen, dass die OSGi-Runtime das Modulsystem der Webanwendung übernimmt und oberhalb des Servlet-Containers sitzt. Vorhandene Bibliotheken wie Eclipse BIRT können relativ leicht übernommen werden.

Auch die Werkzeugunterstützung ist verbessert worden. Während RAP in der Anfangszeit eine Sammlung von Plug-ins war, die man in seine Eclipse-IDE installieren musste, gibt es seit einiger Zeit eine gemeinsame Eclipse-IDE für RCP- und RAP-Entwickler. Unter den Wizards zur Erzeugung eines Eclipse-Plug-ins finden sich mehrere Vorlagen zur Generierung einer RAP-Anwendung. Sie können damit neue RAP-Anwendungen entwickeln und mit vergleichbar überschaubarem Aufwand vorhandene RCP-Programme in das Web übertragen.

2.12 Zusammenfassung

Die Eclipse Rich Client Platform ist eine sehr leistungsfähige Plattform für Java-Desktop-Anwendungen. Eclipse-RCP-Anwendungen sind modular aufgebaut und basieren auf dem Modulsystem OSGi. Sie können dank der OSGi-Infrastruktur ohne die Gefahr von Versionskonflikten erweitert werden. Durch die Workbench mit ihren Views, Editoren und Perspektiven lässt sich die Oberfläche eines RCP-Programms sehr leicht und flexibel konfigurieren.

2.13 Quellcode

Sie finden den Quellcode der in diesem Kapitel vorgestellten Beispiele im Verzeichnis *e2_Rich_Client_Platform* der Downloads.

Links & Literatur

[BS01] Steppan, B.: *Eclipse e4 – Architekturwandel löst Zweifel aus.* *http://www.heise.de/artikel-archiv/ix/2010/08/082_Zwischenwelt*

[BS02] Steppan, B.: *Ajax-Entwicklung erobert Eclipse.* *http://www.cowo.de/a/578598*

[JIGS] *Oracle streicht Project Jigsaw aus Java 8.* *http://www.cowo.de/a/2518150*

[MILE] *Key Milestones Over 10 Years of Eclipse.* *http://eclipse.dzone.com/articles/key-milestones-over-10-years*

[OSGI] *OSGi-Homepage.* *http://www.osgi.org*

[WP01] *Eclipse (IDE).* *https://de.wikipedia.org/wiki/Eclipse_%28IDE%29*

Stichwortverzeichnis