Eclipse Rich Client Platform

Setrak Michaelis

setrak.michaelis@student.uni-siegen.de

Juni 2011

Inhaltsverzeichnis

Einleitung:

Was wäre der Heutige Programmierer ohne eine angemessene Programmierumgebung, die ihn unterstützt, Tipps gibt, korrigiert und noch vieles mehr. Ein wichtiges, wenn nicht schon fast das wichtigste Hilfsmittel des Programmierers ist die Programmierumgebung und deren Features, namentlich: Eclipse, BlueJ, Matlab und einige andere. In den folgenden Seiten, werde ich ihnen das Programm Eclipse etwas näher bringen, und werde etwas genauer auf das Plug-In RCP (Rich Client Platform) eingehen.

Hintergründe:

Java, eine der am häufigsten genutzten und gelehrten Programmiersprachen, unter anderem auch eine der Sprachen, die Eclipse unterstützt. Wie ist es dazu gekommen, hier ein paar geschichtliche Hintergrundinformationen zu Java und Eclipse.

Java

„*Green Projekt*“, so hieß die Projektgruppe die von Sun Microsystems 1991 ins Leben gerufen wurde. Zu Beginn war es Ziel des Projekts eine möglichst effiziente zugleich einfache Sprache zu entwickeln, die, wie man heute sagen würde, „Plattformunabhängig“ ist. Nach einigen Konzepten, die die Programmiersprache C++ als Grundlage nahmen, wurden den Entwicklern jedoch klar, dass die von ihnen entwickelte Sprache, in dem Fall OAK (Object Application Kernel), noch nicht das sei, dass ihnen zu Beginn vorschwebte. Und doch war OAK der erste Schritt in Richtung Java. Zu der Zeit begann das World Wide Web immer mehr an Bedeutung zu zunehmen, und die Entwickler vom „*Green Projekt*“ erkannten das Potenzial und fingen an ihre Arbeit den ersten grafischen Web-Browsern anzupassen. Dadurch, dass OAK die Vorteile besaß klein, objektorientiert, plattformunabhängig und robust zu sein, wurde es immer populärer. Nach einigen Modifikationen wurde OAK Java genannt, nach einem altenglischen Begriff für Kaffee. „*WebRunner*“, das war der Namen des ersten Browsers, das kleine Java-Programme aus dem Word Wide Web laden und innerhalb des Browsers ausführen konnte. Nach dem großen Erfolg von „*WebRunner*“ wurde dieser in „*HotJava*“ umbenannt, modifiziert und im Mai 1995 der Öffentlichkeit vorgestellt. Kurz darauf haben auch die führenden Web-Browser wie *Netscape Navigator* und *MS Internet Explorer* Java integriert. Nach ihrem crashkurshaften Erfolg mit dem Web-Browsern wurde im Jahr 1996 das erste JDK 1.0 (Java Development Kit) veröffentlicht. Darauf gründeten die Mitglieder des „*Green Projekt*“ JavaSoft, eine Tochterfirma von Sun Microsystems. Im Januar 2010 übernahm die Oracle Corporation Sun Microsystems. Es wurden immer wieder neue Versionen des JDK veröffentlicht, unter anderem die erste Betaversion des JDK 1.1 im Februar 1997, Anfang Dezember 1998 das JDK 1.2. Kurz darauf wurde das JDK in SDK (Software Development Kit) umbenannt. 2004 erschien die Java Version 5.0 die der Nachfolger der Version 1.4 war. Die nächste Version ist für Ende Juli geplant, die derweil schon bei Version 7 angekommen ist.

Eclipse

Der Wichtigste Aspekt von Eclipse ist, das es ein quelloffenes Programm zur Entwicklung von Software verschiedenster Art ist. Zu Beginn wurde Eclipse als Programmierumgebung ausschließlich für Java genutzt, da das auch die Programmiersprache sei, in der Eclipse geschrieben ist, aber mittlerweile wird Eclipse als Programmierumgebung für viele andere Entwicklungsaufgaben eingesetzt. Doch fangen wir am Anfang an. Eclipse ist der Nachfolger des von IBM entwickelten *Visual Age*, das für die meisten gängigen Sprachen wie C, C++, Smalltalk und Java zur Verfügung stand. Am 7. November 2001 gab IBM den Quellcode von Eclipse frei, und damit wurde das erste Eclipse-IT-Projekt von IBM begründet und durch ein Konsortium von führenden Softwareanbietern unterstützt. Hierzu gehörten Unternehmen wie Borland, IBM, Rational Software, Red Hat, SuSE und TogetherSoft.

Die Reorganisation zu einem echten Open-Source-Projekt erfolgte im Februar 2004, und wurde unter die Leitung der Eclipse Foundation gestellt, die seitdem für die Entwicklung von Eclipse verantwortlich ist. Eine anbieterunabhängige, offene und transparente Gemeinschaft sollte mit der Gründung der Eclipse Foundation geschafft werden, dies hat auch gut geklappt, da die Eclipse-Community heute aus unterschiedlichen Unternehmen und Einzelpersonen aus allen Bereichen der Softwareentwicklungsbranche besteht. Die von der Open-Source-Gemeinschaft entwickelte Technologie steht kostenfrei unter der Eclipse Public License weltweit zur Verfügung.

Eclipse unterstützt die Java-Version 5 seit Version 3.1 und Java 6 seit Version 3.2. Veröffentlichungstermine von neuen Versionen wurden angepasst, um Versionskonflikte zu vermeiden und Eclipse-Anwendern die Nutzung zu erleichtern

Grundlegendes

Zu Java

„Was ist Java eigentlich?“- fragen sich einige Leute bestimmt. Um genau zu sein ist Java sowohl eine objektorientierte Programmiersprache, als auch eine Plattform. Die Ziele bei der Entwicklung von Java waren, eine einfache, plattformunabhängige und sichere Programmiersprache zu erzeugen, wie vorhin schon erwähnt. Der Sicherheitsaspekt bezieht sich dabei auf die Sicherheit der Sprache, der Laufzeitumgebung und auf zusätzliche Sicherheitsmechanismen, wie z.B. Kryptographie. Dank der Java Virtual Machine (JVM) können Java-Dateien plattformunabhängiger kompiliert werden, wobei Bytecode generiert wird und von der JVM ausgeführt wird. Nicht nur für graphische Benutzeroberflächen / Programme wird Java verwendet, sonder auch auf Serverseiten. Besonders der Einsatz von Java Server Pages (JSP) bei der Programmierung von Webservern gewinnt immer mehr an Bedeutung. Nicht nur diese Aspekte machen Java zu einer sehr starken, benutzerfreundlichen und stabilen Programmiersprache, sondern auch eine Reihe von Konzepten:

Einfach:

Java wurde in Anlehnung an C++ entwickelt, somit hat es viele seiner Stärken, doch wurden so Sachen wir Pointer und Mehrfachvererbung bewusst wegen ihrer Fehlerträchtigkeit weggelassen, um die Sprachen möglichst einfach und klein zu halten. Zudem wurde das Speichermanagement von Java automatisiert, womit dem Programmierer Arbeit abgenommen wurde.

Objektorientiert:

Java ist objektorientiert. Die Grundidee der objektorientierte Programmierung (OOP) ist, Daten und Funktionen, die auf diese Daten angewandt werden können, möglichst eng in einem sogenannten Objekt zusammenzufassen und nach außen hin zu kapseln so, dass Methoden fremder Objekte diese Daten nicht versehentlich manipulieren können.

Verteilt:

Java bietet Funktionen an, die die Kommunikation von Anwendungen über das Netz mittels Standardprotokolle wie TCP/IP ermöglichen. Dies macht er möglich mittels verteilten Systemen und Netzen zu Programmieren. Mit Hilfe des RMI (Remote Method Invocation)-Protokolls ist es möglich Java-Code auf anderen Rechnern zu erstellen und diesen dort auszuführen. SQL Datenbankzugriffe werden auch mittels JDBC-API (Java Database Connectivity) ermöglicht. Außerdem ist es dank JDBC mögliche ganze Datenbank-Applikationen in Java zu schreiben.

Dynamisch:

Java wurde so entworfen, dass es sich einer veränderlichen Umgebung anpassen kann und ist somit eine dynamische Sprache. Ein Programm lädt beispielsweise seine Klassen erst, wenn sie benötigt werden.

Robust:

Java ist dafür bekannt, sehr zuverlässige und robuste Software zu liefern. Es ist zwar immer noch dem Programmierer überlassen, seine Programme möglichst „sinnvoll“ zu gestalten, doch Java selbst sorgt zum Teil für Korrekturen von einigen Programmierfehlern. Somit ist es mit Java wesentlich leichter zuverlässige Programme zu schreiben.

Sicher:

Mittels einer bestimmten Virtual Machine, der sogenannten Sandbox ist es möglich die Java Applets in einem abgegrenzten Bereich auszuführen. Java Applets sind Java Programme, die man mittels des Browsers über das World-Wide-Web empfangen kann. Diese Applets können bewusst oder versehentlich Schaden am eigenen Rechner anrichten. Aus dem Grund benutzt man die Sandbox.

Plattformunabhängig:

Portabilität oder Plattformunabhängigkeit ist die Fähigkeit, dass ein Programm auf unterschiedlichen Plattformen und unterschiedlichen Betriebssystemen läuft. Dies ist ein wesentlicher Vorteil von Java.

Effizient:

Eine sehr wichtige Eigenschaft von Java ist es, das hier das JIT (Just-in-Time)-Compieling angewandt wird, um die Effizienz der in Java erstellten Programme noch deutlich zu erhöhen.
Das Verfahren funktioniert wie folgt: im Gegensatz zu einem herkömmlichen Compiler übersetzt ein JIT-Compiler erst während der Laufzeit und bei Bedarf den vorliegenden Code in einen nativen Maschinen-Code. Anschließend wird der Maschinen-Code ausgeführt, was zu einer deutlichen Performancesteigerung gegenüber Interpretern führt. Hochentwickelte JIT-Compiler können speziell für dynamische Sprachen schnelleren Code als herkömmliche Compiler generieren

Multi-Threaded:

Das Multi-Threading ist auch ein wichtiger Bestandteil der Java Programmierung, da es mit Hilfe von Java möglich ist Programme zu entwickeln, die parallel arbeiten. Durch die Multi-Core PC‘s der heuten Zeit ist genau das sehr wichtig, um die Effizienz noch weiter zu steigern. Zudem wird die Eigenschaft des Multi-Threading von noch sehr wenigen Sprachen angeboten.

Zu Eclipse:

Gehen wir nun etwas näher auf die von IBM entwickelte Programmierumgebung ein und schauen uns die Stärken und Schwächen von Eclipse an. An sich ist der blanke Kern von Eclipse nichts weiter wie das Fenster, in dem sich die Views und Editoren befinden, denn auch diese sind in Eclipse eingebaute Plug-Ins. Im Folgenden will ich auf einige der wichtigen Inhalte von Eclipse eingehen und diese erläutern.

Das Workspace und die Workbench sind aufeinander aufbauende Komponente. Das Workspace steht über der Workbench und in der Workbench befinden sich, die für den Programmierer relevanten Features wie:

Menü und Tool Bars: Diese Leiste ist am obersten Teil der Workbench angebracht und wird quasi intuitiv benutzt. In ihr findet man Buttons die häufig benötigte Operationen ausführen, wie speichern oder öffnen.

Perspective: Container artige Oberfläche, näheres dazu später.

Views: Dienen der Visualisierung von Daten und Ordnern.

Editoren: Eine besondere Art der Views, die es erlaubt die vorhandenen Daten zu bearbeiten.

Ohne das SWT (Standard Widget Toolkit) würden wir unsere Eclipse Fenster nicht so sehen wie wir sie sehen, aus diesem Grund ist auch das SWT ein Bestandteil der Workbench. Das SWT stellt ein Java API bereit, um mit den nativen UI-Widgets des Betriebssystems zu arbeiten. Das Betriebssystem-Widget wird genutzt um auf allen unterstützten Plattformen ein natives „*Look&Feel*“ zu erreicht.

JFace

Weiterer Bestandteil der Workbench ist JFace. JFace ist nichts anderes als ein Hilfegerüst für das sehr umfangreiche und komplexe SWT. Dieses Toolkit baut auf SWT auf und erlaubt bequemes Programmieren, indem es die Ebenen von SWT abstrahiert und trotzdem den direkten Zugriff auf SWT nicht verhindert.

JFace ist ein Optionales Toolkit, da man auch ohne Probleme mit SWT GUI's programmieren kann, doch erleichtert JFace die Handhabung mit SWT und somit ist es nur zu empfehlen. Auch wenn nicht jeder Teil der GUI mithilfe von JFace dargestellt werden kann, bietet JFace dennoch viele Hilfen in der GUI-Programmierung. Unter anderem bietet es Viewer, vorgefertigte Wizards und Registries für Schriftarten und Bilder.

Team Support

Heute genauso wie früher ist es wichtig in einem Team arbeiten zu können. Und um das auch global schaffen zu können braucht man einen gewissen Cooperative Work Support, den uns Eclipse bietet. Durch einige Funktionalitäten von Eclipse ist es möglich Projekte Konfigurationen innerhalb eines gemeinsamen Team-Repositorys zu speichern. Durch das zur Verfügung gestellte Repository Provider API, ist die Einbindung verschiedener Team Repositories möglich. In den Team Repositories selbst können noch viele Spezifikationen eingestellt und geändert werden. Somit ist selbst das Arbeiten Global mit Mitarbeitern aus anderen Ländern kein großes Problem mehr.

Plug-Ins

Nicht jede Programmierumgebung unterstützt Plug-Ins, deswegen ist Eclipse auch so eine gute Wahl. Plug-Ins kann man als Erweiterung der schon vorhandenen Programmierumgebung sehen. Die meisten Plug-Ins sind mit dem Hintergedanken entwickelt worden, dem Programmierer das Leben zu erleichtern und gewissen Hilfestellungen zu leisten. So kann man sagen, dass Plug-Ins separat entwickelte Programme, welche spezielle Funktionen zur Verfügung stellen, sind. Plug-Ins für Eclipse sind in Java geschrieben und falls eine Funktionalität sehr komplex sein sollte kann man diese auch auf mehrere von einander abhängige Plug-Ins Verteilen. Die Bestandteile eines Plug-Ins befinden sich in der Regel im Stammverzeichnis des Plug-Ins, können aber auch auf mehrere Ressourcen verteilt werden. Der Inhalt und die Struktur jedes Plug-Ins wird in ihrer Manifest-Datei festgehalten, und mittels dieser Datei ist es Eclipse möglich, den Inhalt / die Funktionalität der jeweiligen Plug-Ins zu kennen, ohne diese aktivieren zu müssen. Über einen Classloader, den die Plug-Ins verfügen ist es ihnen möglich, benötigte Klassen bereit zu stellen.

Platform Runtime

Ohne die Platform Runtime würden wir keine Plug-Ins haben, da das gesamte Verwaltungsmodel in ihr steckt. Wie z.B. das bestimmen verfügbarer Plug-Ins und noch die Verwaltung der Plug-In Informationen in einer Datenbank. Zudem realisiert die Platform Runtime die „Kommunikation“ zwischen den Plug-Ins. Was auch eine sehr große Hilfe in der Aktualisierung der Eclipse-Plattform und der jeweiligen Plug-Ins ist, ist der Update-Manager, der automatisch nach verfügbaren Updates sucht.

Fat- Thin- oder Rich-Client?

Es gibt Thin Clients, Fat Clients und irgendwo dazwischen gibt es auch den Rich Client. Was genau unterscheidet diese Clients voneinander? Zählen wir erstmals die Gemeinsamkeiten auf. Bei all diesen Clients liegt eine Client-Server-Architektur vor. Doch haben die Clients recht hohe Unterschiede zueinander. Der Fat Client z.B. ist im Gegensatz zum Thin Client ein vollwertiges System, inklusive Applikationen. Das macht ihn sehr benutzerfreundlich und nicht abhängig vom Netzwerk. Der Nachteil des Fat Clients ist der hohe Aufwand der Aktualisierung der Anwenderprogramme. Im Gegensatz dazu sieht der Thin Client sehr mager aus. Denn hier werden weder Applikationen noch das Betriebssystem auf dem Computer gesichert, sonder all dies befindet sich auf den jeweiligen Servern. Ein Vorteil dabei ist, dass die Chance sich einen Virus einzufangen sehr gering ist. Der Nachteil bei den Thin Clients ist, dass es ein Netzwerk-Computer ist und ohne eine Verbindung zum Server ist man in den meisten Fällen aufgeschmissen. Kommen wir nun zum Rich Client, der eine Mischung aus Fat und Thin Client ist. Bevor man eine Software einen Rich Client nennen darf muss dieser erst bestimmte Kriterien erfüllen, wie z.B. eine Benutzerschnittstelle zu besitzen, die in einer XML Syntax basierenden Sprache geschrieben sein muss. Im Gegensatz zum Fat Client lässt der Rich Client auch die wichtigsten Verarbeitungsaufgaben vom Server ausführen, und anders als der Thin Client verarbeitet der Rich Client viele Daten vor Ort auf dem Client selbst.

Näheres zu Eclipse-RCP

Man könnte sagen, das Eclipse-RCP ein Plug-In basiertes Anwendungs-Framework ist. Und zwar zum einen ist es ein Plug-In in der Eclipse-Programmierumgebung, aber auch gleichzeitig eine von der Eclipse IDE mithilfe von der OSGi gekapselte Plattform. Das ist nötig, um eigene, unabhängig von der IDE laufende Programme entwickeln zu können. Und dadurch, dass Eclipse-RCP von Grund an aus Plug-Ins besteht, ist die Erweiterbarkeit und Integration durch andere Plug-Ins gegeben.

Vorteile von Eclipse-RCP

Wenn man von den Vorteilen der Programmiersprache Java Absieht auf der Eclipse basiert, wird einem auffallen, das Eclipse-RCP noch einige Vorteile von seiner Seite anzubieten hat, wie die Dynamischen Funktionalitäten. Folgend werde ich die wichtigsten Vorteile etwas näher erläutern.

- Arbeitsteilung

Verschiedene Teile eines Plug-Ins können als eigenes Plug-In entwickelt werden. Somit kann man ein Plug-In in Einzelteile zerlegen, diese in unterschiedlichen Teams entwickeln, und im letzten Schritt dann alle Teile des Plug-Ins zusammen fügen.

- Einheitlichkeit

Mit Hilfe einer Grundstruktur, die zur Verfügung gestellt wird, ist es möglich einheitlich zu Arbeiten und zu Entwickeln. Außerdem kann man Schnittstellen (Extension-Points) definieren, durch die zukünftige Plug-Ins das Programm erweitern können. Wie kann man sich diese Extension-Points vorstellen? Ganz einfach, in dem man die Extension-Points als einer Art Schlüsselloch sieht und Plug-Ins, die den passenden Schlüssel (Extensions) besitzen, können auf die Schnittstelle zugreifen. Diese Funktionalität macht das Entwickeln mit Eclipse-RCP sehr angenehm, da man mittels dieser Implementierung entwickelte Programme immer wieder ausbauen kann.

- Hilfreiche Funktionen

Durch Eigenschaften wie das Hot-Plugging, wodurch Plug-Ins im laufenden Betrieb des Programms dazu geladen werden können, muss man die Applikation nicht immer neustarten, wenn man Änderungen testen will.

-OSGi (Open Service Gateway Initiative)

Als Unterbau der Eclipse Plattform bietet OSGi eine Vielzahl an nützlichen Funktionalitäten, die es der Eclipse Plattform zur Verfügung stellt.

„On the fly“ Installation, Aktualisierung und Deinstallation von Komponenten, was so viel bedeutet wie ohne das System neustarten zu müssen, bzw. ohne Beeinflussung anderer Programme / Dienste.

Modularisierung von Anwendungen. Aufbau aus einzelnen „Teil“ Plug-Ins, und einiges mehr.

Gehen wir jetzt etwas auf die Entstehung von OSGi ein. OSGi wurde im Jahre 1999 durch einen Zusammenschluss von Sun Microsystems, Oracle und IBM gegründet um Java-Programme auf allen Hardware und Software Komponenten (eingebettete Systeme) Bedingungslos ausführen zu können. Im Jahr 2003 entschloss sich Eclipse OSGi als Unterbau zu verwenden.

Eclipse-RCP

Aufbau der Plattform

Nun da wir wissen was Java, Eclipse und ein Rich Client sind, wird es langsam Zeit uns dem Eclipse-RCP zu widmen. Auf den kommenden Seiten werde ich einige Komponente der Eclipse Rich Client Platform erläutern und näheres dazu sagen. Was man nicht vergessen sollte ist, dass es für unterschiedliche Programmieranlässe unterschiedliche Eclipse Entwicklungsumgebungen gibt. Wobei die eigentliche IDE gleich bleibt und nur durch einigen Plug-Ins erweitert wird, die zusätzliche Funktionalitäten, Perspektiven etc. mit sich bringen, wodurch gewisse Aufgaben leichter zu erledigen sind. Unter anderem die Entwicklung von Plug-Ins erleichtern können. Dadurch, dass Eclipse komplett auf der Programmiersprache Java basiert, ist zwangsläufig auch für den Betrieb der Plattform und das Ausführen von Java-Programmen die JRE (Java Runtime Environment) erforderlich. Über Eclipse-RCP entwickelte Plug-Ins können über RCP selbst ausgeführt werden, doch dies ist nicht zwangsläufig nötig. Plug-Ins die mittels RCP entwickelt wurden, können auch eigenständige Rich Clients sein, die auch ohne die Eclipse-IDE auskommen, doch sind und bleiben sie immer Eclipse Plug-Ins.

Eclipse Core Runtime

Das Starten und Betreiben wird von der Core Runtime von Eclipse übernommen, im Grunde werden alle Kernkomponenten bis auf die Benutzeroberfläche von ihr gesteuert und verwaltet. Außerdem werden noch Plug-In und Ressourcen-Management von der Core Runtime betrieben. Viele der Kernkomponenten von Eclipse wurden mittlerweile von der OSGi Implementierung Equinox übernommen.

Eclipse UI

Das User Interface ist ein sehr wichtiger Punkt in jeder Programmierumgebung, so nutzt Eclipse-RCP die schon in der Eclipse UI vorhandenen GUI Elemente für sich aus. Und zwar werden die Grundobjekte der UI übernommen, wie z.B. Views, Editoren, Wizards oder Menüstrukturen und die Plug-Ins sind dann dafür da, diese mit sinnvollen Inhalten zu versehen.

Window

Perspektiven, Views etc. klingt erst mal gut, aber als erstes braucht eine RCP-Applikation ein Workbench Window, in der dann die Perspektiven und Views ihren Platz finden. Die Workbench verwaltet dieses Fenster. Wird es geschlossen, so wird auch die Workbench geschlossen. Zudem befindet sich am untersten Teil des Windows eine Statusleiste, wo Statusinformationen der Applikation angezeigt werden.

Perspective

Unter Perspektiven kann man sich im Grunde Container vorstellen, die die einzelnen View-Editor-Elemente beinhalten. Die meisten Perspektiven beinhalten unterschiedliche View / Editoren, da es wenig Sinn machen würde, zwei oder mehr derselben Views offen zu haben, aber auch das ist möglich, da man in der Gestaltung und Anordnung der Views freie Hand hat.

View

Views finden in Perspektiven ihren Platz, und sind eigentlich nichts anderes wie eine Anzeige von Daten. Sie dienen der Visualisierung, und der Unterstützung der Editoren. In Eclipse ist es möglich so ziemlich alles in diesen Views darzustellen. In vielen Fällen werden sie mit JFace Komponenten wie der TableViewer, TreeViewer usw. eingebunden.

Editor

Der Editor in RCP unterscheidet sich nicht viel vom Editor im Standard Fester von Eclipse, er dient der Erstellung, Bearbeitung und Speicherung von Dokumenten. Zusätzlich besitzt RCP den Bereich *EditorArea*, wo Dokumente wie Views in Laschen dargestellt werden.

Optionale Komponenten

Unter Optionale Komponenten versteht man Implementierungen, die zum größten Teil von eigenen Implementierungen ersetzt werden können. Unter anderem zählen der Eclipse-Hilfesystem und der Update- Manager zu den Optionalen Komponenten.

Kleines RCP Beispiel

Nun wird der Aufbau einer kleinen Rich Client Anwendung erläutert. Zunächst erstellen wir ein neues Projekt mit Hilfe des „*Hello RCP-Template*“. Um das zu erreichen öffnet man die Eclipse-IDE und fügt dem Workspace ein neues Projekt hinzu. Im Anschluss wählt man beim Wizard unter *Plug-in Development* ein *Plug-in Project* aus. Jetzt muss man als *Target Platform* die aktuelle Eclipse Version auswählen und gibt dem Projekt einen Namen. Auf der nachfolgenden Seite muss der Haken bei „*This plug-in will make contributions to the UI*“ und das man eine Rich Client Anwendung erstellen will, ausgewählt sein. Nun kann man ein vorgefertigtes Template auswählen, in unserem Fall ist das das „*Hello RCP-Template*“. Entweder man beendet nun den Wizard, oder man kann auf der kommenden Seite noch nähere Angaben zu Applikation geben.

Wichtige Dateien

Bevor Eclipse auf die OSGi Service Plattform umgesetzt wurde, wurden die Informationen für die Ausführung des Plug-Ins in der plugin.xml Datei gespeichert. Somit ist die plugin.xml Datei neben der Manifest Datei eine der Wichtigsten Dateien einer Eclipse-RCP Anwendung / Plug-In gewesen. Aus dem Grund, dass heute als Unterbau der Eclipse Plattform OSGi genutzt wird, werden diese Informationen nun in der Manifest Datei des Plug-Ins gespeichert. *Extension* und *Extension-Points* werden jedoch weiterhin in der plugin.xml Datei gespeichert, die sich im Hauptverzeichnis befindet. Die Erweiterungspunkte und zugehörigen Erweiterungen, können mithilfe der *ExtensionRegistry*, die die plugin.xml Datei ausliest, abgerufen werden.

Das Ausführen

Eine neue Startkonfiguration wird benötigt um das Beispiel ausführen zu können. Dies erstellen wir unter dem Menü *Run* / *Run Configurations*... Jetzt wählt man *Eclipse Application* aus und gibt der Konfiguration einen sinnvollen Namen. Unter *Program to Run* bei der Lasche *Main* klickt man auf *Run an application* und wählt die erstellte Anwendung aus. Bei der Lasche *Plug-ins* wählt man *launch with: plug-ins selected below only* an und klickt rechts auf *Deselect All*. Damit kann man die Plug-Ins, die beim Starten der Anwendung im Framework automatisch installiert/ausgeführt werden, selbst bestimmen. Bei unserem Beispiel wird der Rich Client, sowie das erstellte Plug-In, ausgewählt und deren Auto-Start auf *true* gesetzt. Um das Ausführen zu vollenden muss nur noch ein Klick auf *Add Required Plug-ins* getätigt werden, womit noch alle zusätzlich benötigten Plug-Ins ausgewählt werden. Nun kann man einfach auf *Run* klicken um die Anwendung zu starten.

Fazit

Meiner Meinung nach ist Java, eine der am einfachsten zu erlernenden, aber wie viele andere auch, sehr schwer zu perfektionierende Programmiersprache. Und ich finde, dass Eclipse eine Programmierumgebung ist, die schon fast mehr macht, als nur eine etwas komfortablere Programmierung zu bieten. Als ob das nicht schon genug sei, kommt noch das Eclipse-RCP Plug-In dazu, das einem noch mehr und bessere Möglichkeiten bietet sich in der Welt der Programmierung auszutoben. Deshalb bin ich zu der Meinung gekommen, dass RCP ein sehr vielseitiges Plug-In ist um die unterschiedlichsten Plug-Ins / *stand-alone* Programme zu produzieren.

Quellen Verzeichnis

The Eclipse Foundation
http://www.eclipse.org

Jimmy Wales Gründer von Wikipedia
http://en.wikipedia.org

Geschäftsführer: Klaus Lipinski, Dipl.-Ing. (V.i.S.d.P.)
http://www.itwissen.info/

Lars Vogel
http://www.vogella.de/

Autor Ralf Ebert
http://www.ralfebert.de/eclipse_rcp/