Forschungszentrum Informatik
an der
Universität Karlsruhe
Forschungsbereich Softwaretechnik

Bewertung von Programmier- und Entwicklungsassistenten am Beispiel eines Eclipse Plug-Ins für den Entwurf verteilter Systeme mit der EventChannelNetwork-Kommunikationsinfrastruktur

Studienarbeit

cand. Inform. Mohammad Athar Januar

Verantw. Betreuer: Prof. Dr. Walter F. Tichy
Betr. Mitarbeiter: Dipl. Inform. Marc Schanne

Hermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt habe. Die verwendeten Quellen sind im Literaturverzeichnis vollständig aufgeführt.

Mohammad Athar Januar Karlsruhe, im November 2006

Kurzfassung

Eclipse ist eine integrierte Programmier-und Entwicklungsumgebung (Integrated Development Environment, IDE), die besonders als Java-IDE populär geworden ist.

Die komponentbasierte Struktur mit Plug-Ins ermöglicht die einfache Erweiterung von Eclipse und gilt als die Stärke von Eclipse im Vergleich zu anderen Entwicklungsumgebungen. Eclipse besteht standardmäßig aus einem Java-Plug-In (Java Development Toolkit, JDT) und einer Plug-In-Entwicklungsumgebung (Plug-in Development Environment, PDE). Java-Plug-Ins können als Programmierer- und Entwicklungsassistenten (Wizards) ganz unterschiedliche Funktionen besitzen. Sie bieten eine softwareprozessgestütze Entwurfsunterstützung, eine Projektverwaltung, oder, wie z.B. UML-Plug-Ins, eine graphische Darstellung der Entwicklung.

Ergebnis der Studienarbeit ist eine Bewertungsmetrik und eine Beschreibung, wie gute Programmier- und Entwicklungsassistenten als Plug-In-Komponente entworfen und implementiert werden können. Als Beispiel wird ein Entwicklungsassistent für verteilte Systeme im Rahmenwerk des Nachrichtenkanal-Netzwerks (EventChannelNetwork, ECN) implementiert. ECN verlangt die Entwicklung von Java-Klassen und abgestimmten XML-Konfigurationsdateien. Die Studienarbeit diskutiert dabei Kriterien, die zur Beurteilung von Entwicklungsassistenten geeignet sind, und verwendet den ermittelten Kriterienkatalog beim Entwurf der Benutzerschnittstelle und -führung im ECN-Plug-In.

Inhaltsverzeichnis

Kapitel 1

Motivation

Eine integrierte Programmierer- und Entwicklungsumgebung ("Integrated Development Environment", IDE) ist ein Computerprogramm zur Entwicklung von Software. Es bietet dem Entwickler "Programmierkomfort" und dieser benötigt keine zusätzlichen Werkzeuge, um z.B. den Quellcode zu debuggen oder zu übersetzen. Eine IDE integriert Werkzeuge für das effektive Programmieren: Editor mit Syntax-Highlighting, Übersetzer, Debugger, Projekt-Verwaltung, und Ähnliches.

Für Java existieren eine Reihe von IDEs, z.B. NetBeans [1], JBuilder [2], BlueJ [3], IntelliJ IDEA [4], und Eclipse [5]. Eclipse hat im Gegensatz zu anderen IDEs den großen Vorteil, der Erweiterbarkeit. Eclipse lässt sich durch sogenannte Plug-In-Komponente erweitern. Ganz gleich für welche Programmiersprache oder visuelle Programmierumgebung, durch Installation notwendiger Plug-In-Komponenten wird Eclipse eine Entwicklungsumgebung für beispielweise PHP-basierte Web-Anwendungen, oder graphische UML-Beschreibungen.

Dank der Plug-In-Komponentenarchitektur werden für das freie IDE-Projekt weitere Entwickler gewonnen. Schon jetzt sind über hundert Eclipse-Plug-Ins vorhanden und können kostenlos unter http://www.eclipseplugincentral.com/Web_Links.html heruntergeladen werden.

Das im FZI Karlsruhe entwickelte Nachrichtenkanal-Netzwerk (EventChannelNetwork, ECN) verwendet Eclipse als IDE und Java mit XML als Programmiersprache für die Entwicklung verteilter Systeme. In der vorliegenden Studienarbeit wird ein Eclipse-Plug-In für ECN entwickelt und daraus werden Kriterien für die allgemeine Bewertung von Programmier- und Entwicklungsassistent (PEA) ermittelt.

1.1 Inhaltsübersicht

Diese Studienarbeit gliedert sich in sechs Kapitel. Ausgehend von der Motivation der Studienarbeit werden in Kapitel 2 die notwendigen Grundlagen vorgestellt. Kapitel 3 beschäftigt sich mit dem Entwurf von dem ECN-Plug-In und gibt einen Überblick über die Anforderungen, die an einen PEA gestellt werden. In Kapitel 4 wird die Implementierung des ECN-Plug-Ins vorgestellt. Die Ergebnisse werden in Kapitel 5, und das Erreichte als "*Lessons learned*" in einem Kriterienkatalog für PEA-Erweiterungen zusammengefasst. Am Ende wird der Sinn solcher Erweiterungen in einem Fazit diskutiert.

Kapitel 2

Grundlagen

Dieses Kapitel gibt einen Überblick über Grundlagenkenntnisse, um das Eclipse-Plug-In für ECN zu entwerfen und zu entwickeln. Ausgehend von der Einführung in Entwicklungs-Assistenten werden die Grundlagen für Eclipse und die Eclipse-Benutzerschnittstelle vorgestellt. Die Umsetzung eines Assistenten innerhalb der Eclipse-IDE wird in diesem Abschnitt untersucht. Das in der vorliegenden Studienarbeit zu entwickelnde Plug-In soll die Arbeit bei der Programmierung von verteilten Systemen mit dem Nachrichtenkanal-Netzwerk ("EventChannelNetwork", ECN) unterstützen und daher wird eine notwendige Einführung in ECN gegeben. Das Kapitel wird durch einen kurzen Überblick verwandter Arbeiten abgerundet.

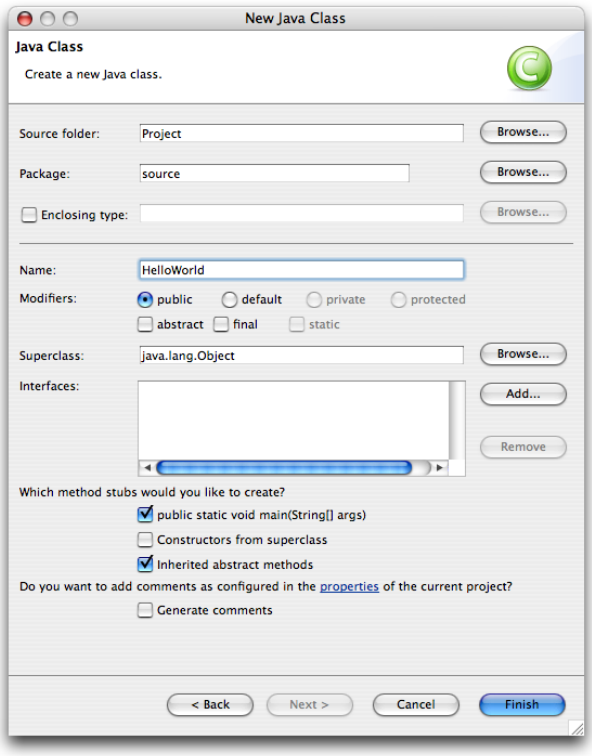
2.1 Entwicklungs-Assistent für Softwareentwicklungsprozess

Ein Assistent biete eine Oberfläche, mittels der ein Benutzer durch mehrere Dialoge für eine ergonomische Dateneingabe geführt wird. Zu Assistenten gehören z.B. Editoren, Ansichten, Ressourcenverwaltung, Aufgabenverwaltung, Problembehandlung, Hilfesystem, und sonstige Erweiterungen. Der Sprachgebrauch variiert stark von System zu System und der verwendeten Basis-IDE. Der Assistent ist eine gute Unterstützung für einen langen komplexen Prozess und bietet dem Benutzer eine erleichterte Vorgehensweise. Die geeigneten Prozesse für diesen Denkansatz sind typischerweise entweder verzweigt, oder lang und ermüdend. Sie bestehen aus einer Reihe von abzufragenden Informationen, die vom Benutzer einzugeben sind und für das Endergebniss wichtig sind.

Assistenten sind nach dem Teile und Herrsche-Algorithmus konzipiert. Die Idee ist, einen gesamten Prozess in mehrere Teilprozesse oder Schritte zu zerlegen. Dabei muss genau beachtet werden, dass Teilprozesse in eine vernünftige Abfolge angeordnet werden. Beim Entwerfen von Assistenten muss immer die Balance zwischen der Große eines Teilprozesses und der Anzahl der Teilprozesse beachtet werden. Es macht kaum Sinn, einen großen zwei-Schritte-Assistent, bzw. einen 15-Schritte-Assistent zu verwenden. Andererseits soll ein Teilprozess nicht "riesig" sein, sonst verliert man den Vorteil des Assistenten.

Jeder Teilprozess sollte als eine Seite verstanden werden, und damit besteht ein Assistent aus mehreren Seiten. In jeder Seite fragt der Assistent nach bestimmten Eingaben, die für die nächsten Seiten bzw. für das Endergebniss entscheidend sind. Dies erfolgt solange bis alle notwendigen Informationen gesammelt sind. Dann ist der Assistent zu Ende, er verarbeitet die erhaltenen Informationen und liefert das gewünschte Endergebnis.

Da ein Assistent aus mehreren Seiten besteht, muss eine Steuerung (Navigation) vorhanden sein, die dem Benutzer erlaubt, von einer Seite auf die nächste Seite zu gelangen und eine Navigation in die entgegengesetzte Richtung unterstützt. Bei Assistenten findet man diese Navigation oft als Vorwärts- und Rückwärts-Knöpfe (**Next**, **Back**), wie in der Abbildung 2.1 zu sehen ist.



Die Abbildung 2.1: Assistent in Eclipse

Der Assistent kann auch dem Benutzer die Möglichkeit bieten, den Vorgang abzubrechen. Dies geschieht durch den Abbruch-Knopf (**Cancel).** Das Ende bezeichnen Assistenten oft durch die Anzeige eines Ende-Knopfes (**Finish**).

Der Entwurf eines Assistenten hängt mit dem Konzept zur Verbesserung der Textverständlichkeit zusammen. [6] wiesen in umfangreichen wissenschaftlichen Experimenten nach, dass Texte besser verstanden werden, wenn einige Eigenschaften der Textverständlichkeit beachtet werden. Eine der vier Dimension der Textverständlichkeit ist dabei die Gliederung und Ordnung, der Rest dieser Arbeit wird hier nicht diskutiert, da es das Thema Assistenten nicht betrifft. Mit der Gliederungs- und Ordnungs-Dimension beurteilt man, wie geordnet und übersichtlicht der Text aufgebaut ist. Größere Bedeutung erhält diese Dimension erst bei längeren Texten. Dies gilt natürlich auch für lange und große Prozesse. Einen solchen Prozess kann man gliedern. Die Form eines Assistenten beschreibt durch seine Gliederung die Form von Teilprozessen. Dadurch gewinnt man mehr Übersichtlichtkeit, alles kommt schön “der Reihe nach”.

2.2 Eclipse

In der vorliegenden Studienarbeit wird ein Programmier- und Entwicklungsassistent (PEA) zur Unterstützung des Softwareentwicklungsprozesses bei der ECN-Programmierung (Kapitel 2.4) mit der Eclipse-IDE entwickelt. Dieses Unterkapitel gibt einen Überblick über verwendete Werkzeuge für Entwurf und Erstellung der Plug-In-Komponenten für Eclipse.

Eclipse ist eine IDE, die 2001 von IBM veröffentlicht wurde. Die laut IBM [28] 40 Million US$

teure IDE wurde der Open-Source-Gemeinde geschenkt, und damit kann jeder Eclipse kostenlos herunterladen. Die Nonprofit-Organisation Eclipse Foundation verwaltet die Weiterentwicklung von Eclipse, dabei spielt IBM weiterhin eine einflussreiche Rolle. Eclipse Foundation hat über 150 Mitglieder, zu denen große Firmen, wie z.B. Borland, IBM, Intel, und SAP AG gehören.

Mittlerweile ist Eclipse als Java-Entwicklungsumgebung etabiliert. Die ursprungliche Beschreibung, zitiert aus der Webseite von eclipse.org [8], lautet: *"Eclipse is an open source community whose projects are focused on providing a vendor-neutral open development platform and application frameworks for building software."*. Dabei wird überhaupt nicht erwähnt, dass die Eclipse-Plattform für einen speziellen Zweck gedacht ist. Eclipse ist eine Plattform für "alles Mögliche und nichts im Besonderen". Eclipse ist zwar populär als Java-Entwicklungsumgebung geworden, dies ist aber nicht alles, was man mit Eclipse machen kann. Der Anwendungsbereich umfasst nicht nur Java-Entwicklung, sondern geht über diese Anwendung hinaus. Dank des Erweiterungskonzeptes lässt sich das Ziel als universelle Plattform durch Plug-In-Komponenten realisieren. Das JDT-Plug-In verwandelt Eclipse in die Java-Entwicklungsumgebung und ist nur ein Beispiel für ein Eclipse-Plug-In. Durch das CDT-Plug-In ("C/C++ Development Toolkit") ist Eclipse als C/C++-Entwicklungsumgebung zu benutzen. Eclipse Foundation entwickelt auch mehrere Plug-In-Komponente, z.B. WTP ("Web Tools Platform") für die Entwicklung von Enterprise-Java, VE ("Visual Editor") für graphische Benutzerschnittstelle-Entwicklung ("Graphical User Interface", GUI), und anderes [9]. Die Abbildung 2.2 zeigt eine Eclipse-basierte Software der NASA namens Jet Propulsion Laboratory für die Erforschung des Mars [10] an.

Eclipse stellt außerdem mit seinen SWT- und JFace-Bibliotheken eine GUI-Alternative zu den Sun Java-Bibliotheken AWT und Swing zur Verfügung. Dies wird in Kapitel 2.2.2 vertieft. Mit Hilfe dieser Bibliotheken lassen sich Java-Anwendungen erstellen, die sich wie native Applikationen des Betriebssystems verhalten.

Eine Reihe von Bibliotheken stellt Eclipse für Java-Anwendungen zur Verfügung. Über die GUI-Bibliotheken hinaus stehen Klassen zur Entwicklung von eigenen Java-Anwendungen bereit. Dazu gehören die Klassen für Editoren, Ansichten, Ressourcenverwaltung, Problembehandlung, Hilfesystem, und verschiedene Assistenten. Sie werden selbst von Eclipse z.B. in der Java-IDE verwendet. Einige Klassen werden hier in Rahmen der Entwicklung der Plug-In-Komponente eingeführt.

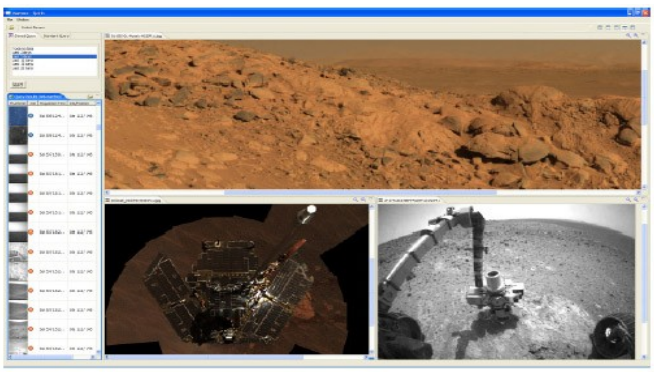


Abbildung 2.2 : NASAs Eclipse-basierte Software JPL

2.2.1 Überblick über Eclipse

Wie die Abbildung 2.3 zeigt, besteht Eclipse nur aus zwei Komponenten: einer Laufzeit-Plattform und einer Menge von Erweiterungskomponenten (Plug-Ins: **Workbench**, **Workspace**, **Help**, und **Version and Configuration Management**). Außerdem können neue Plug-In-Komponente in die Plattform eingebettet werden. Die Laufzeitplattform hat einen Mikrokernel und ist dafür zuständig, diese Komponente zu laden.

Der Arbeitsbereich (**Workspace**) hat die Aufgabe, die Resourcen zu verwalten. Er besteht aus einem oder mehreren Projekten, wobei jedes Projekt auf ein entsprechendes benutzerspefizisches Verzeichnis im Dateisystem abgebildet wird. Ein Projekt beinhaltet die Dateien, die vom Benutzer angelegt und verarbeitet werden. Alle Dateien innerhalb des Arbeitsbereichs können direkt auf die gewöhnlichen Programme und Werkzeuge des zugrunde liegenden Betriebssystems zugreifen. Die in der Plattform integrierten Werkzeuge sind mit Programmierschnittstelle ("Application Programming Interface", API) verfügbar, um die Ressourcen des Arbeitsbereichs zu nutzen. Ressourcen können in Form von Projekten, Dateien, und Verzeichnissen existieren.

Die Struktur von Projekten erlaubt dem Werkzeug, ein Projekt zu markieren, um ihm eine besondere Eigenschaft zu geben. Mit Java, beispielsweise, wird ein Projekt markiert, das Java-Quelldateien enthält.

Der Arbeitsbereich verfügt über einen Rücknahme-Mechanismus, der jede Änderung von Dateien beobachtet, um das Risiko von verlorenen Dateien zu reduzieren. Der Arbeitsbereich stellt einen Markier-Mechanismus zur Verfügung. Die Markierungen können als Fehlermeldungen, *To-Do*-Liste, Lesezeichen, Such-Treffer und Debugger-Haltepunkte erscheinen.

Die Eclipse-Plattform-Benutzerschnittstelle ist um eine Benutzerschnittstelle herum aufgebaut, die eine allgemeine Struktur bietet. Die Workbench-API und die Implementierung basieren auf zwei Klassenbibliotheken, SWT und JFace. SWT ist eine Gruppe von Benutzerschnittstelle-Komponenten und graphische Bibliothek. Sie ist in das native Fenstersystem integriert, aber mit einer betriebssystem-unabhängigen API. Dadurch sind die in Eclipse geschriebenen Java-Anwendungen portierbar und verhalten sich wie native Anwendungen des entsprechenden Betriebssystems. JFace, andererseits, ist ein Benutzerschnittstellen-Werkzeug, das für die Implementierung SWT verwendet, um allgemeine Benutzerschnittstelle-Programmieraufgaben zu vereinfachen. JFace setzt auf SWT auf, und die Workbench verwendet SWT und/oder JFace.

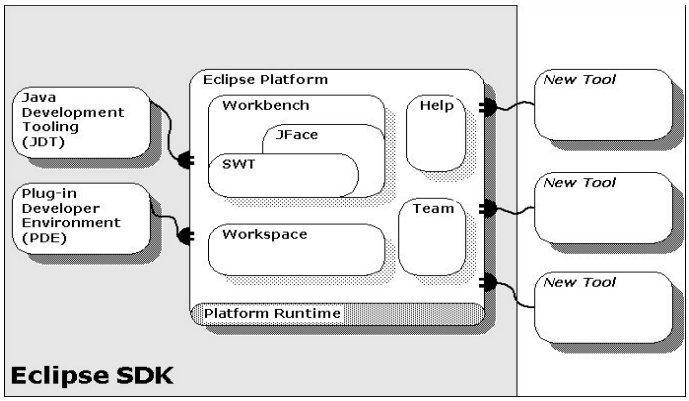


Abbildung 2.3: Die Architektur von Eclipse

Eine Team-Komponente (“Version and Configuration Management”, VCM) unterstützt Gruppenarbeit. VCM erlaubt die Versionierung von Projekten im Arbeitsbereich und die Verwaltung zugehöriger Team-Ablage. Die Eclipse-Plattform ist mit der Unterstützung für CVS-Ablagen eingerichtet. Der Zugriff auf die CVS-Ablage erfolgt durch das `pserver` oder `ssh`-Protokoll. Interessiert man sich für die Eclipse-Quellcode, kann direkt darauf zugreifen:

- CVS-Dienstgeber : `dev.eclipse.org`
- Ablage-Pfad : `/cvsroot/eclipse`
- Benutzername : `anonymous`
- Passwort : leer lassen
- Verbindungstyp : `pserver`

Für die Hilfe wird eine **Help**-Plug-In-Komponente verwendet. Sie ist eigentlich nur ein erweiterbares Dokumentationssystem. Typischerweise bietet das Hilfesystem die API-Dokumentation. Der grobe Inhalt wird als HTML-Dateien zur Verfügung gestellt. Die zentrale Hilfeeinrichtung der Eclipse-Plattform findet man unter der Webadresse http://help.eclipse.org.

2.2.2 Workbench und UI-Werkzeuge

Die Eclipse-Plattform brachte bei der ersten Vorstellung eine Überraschung, was die Benutzerschnittstelle (“User Interface”, UI) betrifft. Eclipse war zwar in Java geschrieben, aber nicht mit gewöhnlichen Java-UI-Bibliotheken, sondern mit einer eigenen, die mit Eclipe entwickelt wurde. Die neuen Bibliotheken nennen sich SWT (“Standard Widget Toolkit”) und JFace. SWT-Bibliothek bietet eine übliche betriebssystemunabhängige Programmierschnittstelle für UI-Komponente und Grafiken, die durch eine enge Anbindung an das unterliegenden nativen Fenstersystem implementiert sind. JFace beschreibt die höheren Ebene der Eclipse-Benutzerschnittstelle.

Der Verzicht auf die üblichen GUI-Bibliotheken war zwischen Java-Entwicklern sehr umstritten [11], denn die Sprache Java selbst vereinigte alle Eigenschaften der Plattformneutralität in sich. Für eine IDE, die selbst in Java geschrieben war, musste dies ebenfalls gelten. Im Laufe der Zeit folgten aber andere Töne, und immer öfter wurde das Resultat dieses “Frevels” gelobt.

AWT

AWT (“Abstract Window Toolkit”) war als eine Benutzerschnittstelle für Applets[1] konzipiert worden. Aber es fehlten viele Funktionen und die gewünschte Flexibilität [12]. Sun Microsystem als Hersteller von Java mochte damit eine “*write once, run everywhere*”-Umgebung schaffen [13]. Die Java-Anwendung sollte in einer Plattform (z.B. Microsft Windows) entwickelt werden. Und der übersetzte Bytecode sollte in unterschiedlichen Plattformen (z.B. Mac, Linux) in der gleichen Art und Weise ausgeführt werden. Das hat AWT leider nicht völlig geschafft. Da AWT von der Dienstgeber-GUI-Steuerung abhängt, um die GUI-Komponente zu implementieren, verhalten sich die AWT-Anwendungen deswegen im unterschiedlichen Betriebssystem oft unterschiedlich. Das führt zu einem “*write once, test everywhere*”-Zustand.

1 Java-Anwendung, die in HTML-Seiten eingebettet sind

Swing

Dieses Problem löste Sun durch die AWT-Erweitertung namens Swing. Aus theoritischer Sicht wurde Swing als MVC-Architektur (**Model View Controller**) konzipiert [11]. Hierbei werden die GUI-Daten sowohl von den Graphik-Komponenten als auch von den Bearbeitungs-Klassen getrennt. Swing löste aber ein neues Problem aus [13], denn Swing ist nicht in der Lage, Hardware-GUI-Beschleuniger richtig auszunutzen. Die Swing-Anwendungen sind deshalb langsamer, aber besitzen mehrere Komplexe-UI-Elemente.

SWT

Die oben genannten Gründen führten die Entwickler von Eclipse zu einer neuen GUI-Entwicklung. Sie setzten sich zum Ziel, die IDE zu allen marktüblichen Betriebssystemen kompatibel zu gestalten. Das Ergebnis sollte mindestens mit einem Visual-Studio und den Microsoft-Foundation-GUI-Klassen mithalten. Im Gegensatz zu Swing verwendet SWT die nativen graphischen Elemente des Betriebssystems, und damit weist SWT eine Optik, vergleichbar mit nativen Programmen, auf. Im Unterschied zu AWT, versuchen die SWT-Entwickler, native Widgets durch möglichst dünne Wrapper einzubinden und man bezeichnet die Steuerelemente in UI als **Widgets,** z.B. `Label`, `Button`, `Text`, usw.

Eine Schwäche von SWT tritt bei der Entwicklung von sich auf, denn hierfür muss der Entwickler weitere Einstellungen durchführen. Dies liegt daran, weil JDK[2] nicht zusammen mit SWT ausgeliefert wird. JDK aus dem Haus Sun Microsystem, und SWT von der Konkurrenz IBM. Man muss sich die SWT-Bibliothek zusätzlich herunterladen bzw. sie, zum gehörigen SWT-Projekt abbilden [14].

Um die SWT-Widgets verwenden zu können, müssen zwei wichtigen Paketen, `org.eclipse.swt` und `org.eclipse.swt.widgets` importiert werden. Danach sind zwei Grundkomponente zu initialisieren, Objekte der Klasse `Display` und `Shell`.

```
Display display = new Display();

Shell shell = new Shell(display, SWT.NONE);
```

Das Objekt von `Display` enthält alle GUI-Komponenten. Normalerweise wird für eine Anwendung nur ein einziges `Display`-Objekt erzeugt. Ein `Shell`-Objekt ist ein Fenster innerhalb der Anwendung.

Möchte man ein Label auf das Shell einfügen, so erfolgt dies z.B. mit:

```
Label label = new Label(shell, SWT.NONE);
```

SWT-Komponente werden immer verwendet, um die Benutzerschnittstelle darzustellen. Die Implementierung findet typischeweise in der `createControl()`-Methode statt.

JFace

JFace ist eine, auf SWT aufgebaute, Bibliothek für Benutzerschnittstelle. Sie implementiert mittels SWT-Widgets den GUI-Code, dadurch wird weniger Code notwendig als mit SWT. Um beispielsweise eine Nachrichtendialog zu erstellen, kann man entweder die SWT-Widgets verwenden, oder die von JFace bereitgestellten Klassen. Mit SWT muss das `Shell`-Objekt für den Dialog erstellt werden, seine Große und die Position der entsprechenden Widgets festgelegt

2 Java Development Kit

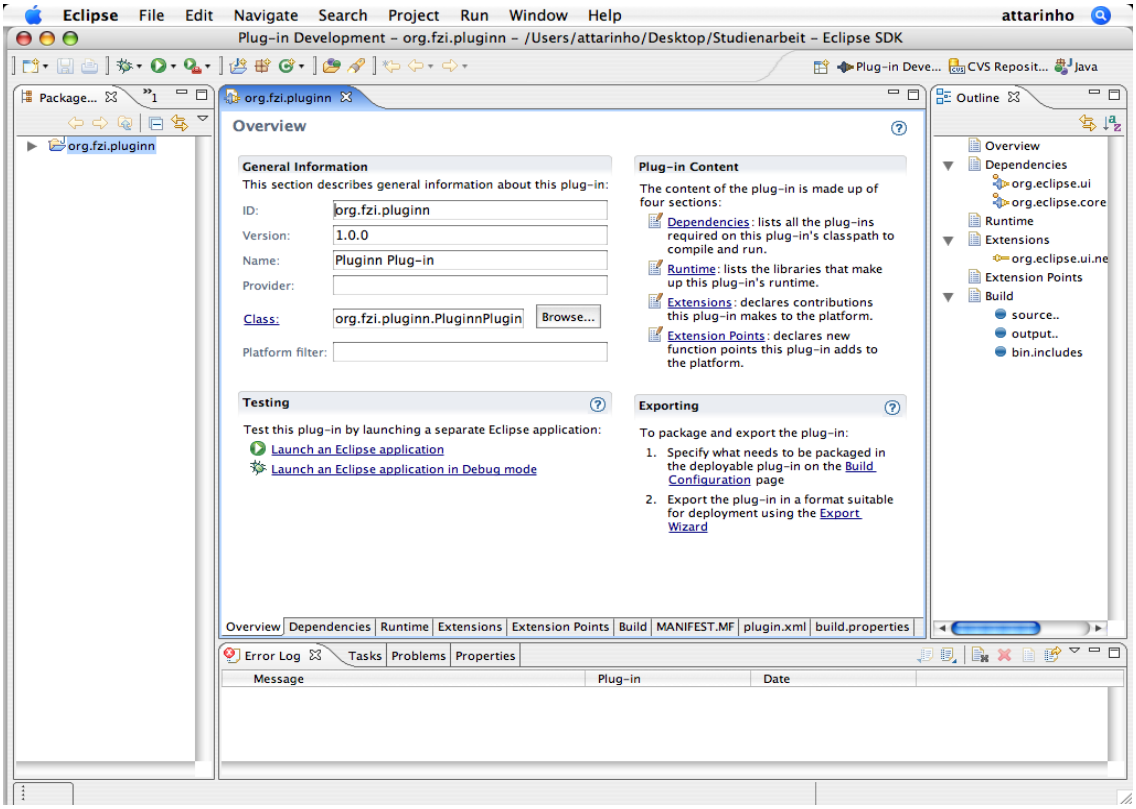


Abbildung 2.4 : Die Verwendung von Eclipse-Forms innerhalb PDE

werden, während mit JFace nur die Methode zur Erstellung des Dialogs mit den entsprechenden Argumenten aufgerufen werden muss. Der identische Nachrichtendialog wird erzeugt. Beide Verfahren bietet Vor- und Nachteile. Es ist aber kaum eine Plug-In-Komponente zu finden, die ohne die Verwendung von JFace entwickelt wurde. Die wichtigen Paketen von JFace finden sich im Plug-In-Paket `org.eclipse.jface`.

Eclipse Forms

Mit der Einführung von Eclipse 3.0 wurde die Eclipse-Forms-API neu vorgestellt. Ein Beispiel der Verwendung von Eclipse-Forms kann man bei der Plug-In-Entwicklungsumgebung von PDE (siehe Abbildung 2.4) finden. PDE wird ausführlich im nächsten Abschnitt besprochen.

Mit Eclipse-Form werden Plug-In-Komponenten, die eine Gruppe von spezifischen Widgets ermöglichen, angeboten [15]. Durch die Erweiterung der SWT-Bibliothek stellt die Eclipse-Forms-Bibliothek ein Werkzeug für eine “Web-ähnlich”-UI-Entwicklung zur Verfügung. Dies umfasst:

- Ein Konzept eines Formulars, das für die Einbeziehung innerhalb des Inhaltbereichs geeignet ist (Editor oder Sicht).
- Eine Werkzeug-Verwaltung für Farben, Hyperlink-Gruppen, und andere wichtigen Elemente eines Formulars.
- Eine Darstellung ähnlich zu HTML-Tabellen mit neuer Layout-Verwaltung.
- Spezifische Steuerung, die für Formular geeignet ist, z.B. Hyperlink, scrollbare

Kompositionen, Abschnitte.

- Mehrseitiger Editor wie PDE.

Diese API befindet sich im Plug-In-Paket `org.eclipse.ui.forms`. Damit hat der Entwickler ein mächtiges Mittel für die Gestaltung formularorientierter Sichten und Editoren zur Verfügung.

Im Wesentlichen verwenden Formulare die SWT-GUI-Elemente [7]. Diese werden jedoch etwas anders konfiguriert und zum Teil auch mit einigen neuen Elementen ergänzt. Da die richtige Konfiguration entscheidend für einen konsistenten Formularaufbau ist, sollte man darauf verzichten, innerhalb eines Formulars SWT-GUI-Elemente auf die gewohnte Weise mit Hilfe von Konstruktoren zu erzeugen. Stattdessen stellt die Eclipse-Forms-Bibliothek mit Hilfe seiner FormToolkit-Klasse verschiedene Fabrikmethoden zum Erzeugen dieser Elemente zur Verfügung. Folgender Code erstellt ein Formular, wie die Abbildung 2.5 illustriert.

```
public void createFormControl(IManagedForm managedForm){

      ScrolledForm form = managedForm.getForm();

      form.setText("This is an Eclipse Forms");

      form.getBody().setLayout(new GridLayout());

}
```

Mittels des Formulars und mehrseitiger Editoren lassen sich Änderungen am Quellcode vornehmen, ohne ihn per Hand zu ändern. Das wird in der PDE verwendet, um die Änderung einer XML-Datei durch ein Formular durchzuführen.

2.2.3 Assistenten in Eclipse

Die Eclipse-Plattform bietet den Benutzern die Möglickeit, den Aufwand des Softwareentwicklungsprozesses zu minimieren. Ein triviales Beispiel ist die Entwicklung von Java-Anwendungen. Der Arbeitsbereich verwaltet alle Ressourcen, die in einem Projekt-Verzeichnis gelagert sind. Möchte man eine neue Klasse erstellen, so wird ein Assistent angeboten. Er fragt vom Benutzer wichtige Infomationen ab. Diese sind z.B. der Name der Klasse, die erweiterte Oberklasse, implementierte Schnittstellen, oder ob die zu erstellende Klasse eine eigene `main()`-Methode besitzen soll. Nachdem die erforderlichen Infomationen

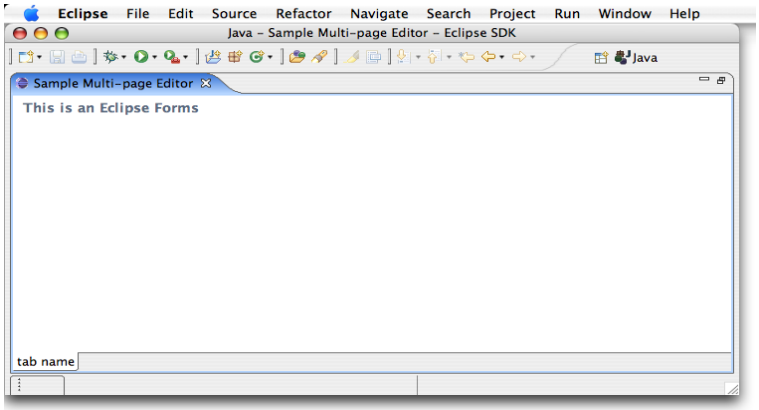


Abbildung 2.5 : Ein formularorientierter Editor mit Eclipse-Forms-Bibliothek

richtig eingegeben sind, generiert der Assistent automatisch die gewünschte Java-Klasse als Rahmen und speichert sie im zugehörigen Projekt. Diese Art eines Assistenten wird in Eclipse als **Wizard** bezeichnet. Bei der Java-Programmierung verfügt der Assistent über den geeigneten Editor. Der Editor ermöglicht Syntax-Markierungen, Fehlermeldungen, Warnungen, und vieles mehr.

Die Wizard-Bibliothek kommt zum Einsatz [16], wenn irgendeine Aufgabe in einer Reihenfolge zu erledigen ist. Der Java-Projekt-Wizard ist ein Beispiel, der aus zwei Seiten besteht. Die erste fordert die Angabe über den Projektnamen, während die zweite optional ist. Dabei handelt es sich um die Angabe über importierte Java-Archive, andere Java-Projekte, usw.

Eine `WizardDialog`-Klasse (`org.eclipse.jface.wizard.WizardDialog`) ist eine spezialisierte Unterklasse der Dialog-Klasse [17]. Sie definiert die gewöhnlichen Wizard-Knöpfe und verwaltet eine Gruppe von Wizard-Seiten. Ein `WizardDialog`-Objekt liefert die Möglichkeit des An- und Abschaltens von Vorwärts-, Rückwärts- und Ende-Knöpfen.

Die Wizard-Klasse steuert das gesamte Verhalten und Aussehen des Wizards, wie die Titelleiste, den Text, das Bild und das Vorhandensein des Hilfe-Knopfs. Die Hauptaufgabe des Assistenten besteht darin, die Wizardseiten zu erstellen und in Wizard einzufügen. Außerdem muss in dieser Klasse das Verhalten beim Drücken des Finish-Knopfs implementiert sein. Der folgende Code zeigt, wie man zwei Wizardseiten instanziieren und in den Wizard einfügen kann:

```
private ECNNewProjectWizardPageOne page1;

private ECNNewProjectWizardPageTwo page2;

public void addPages(){

      page1 = new ECNNewProjectWizardPageOne();

      addPage(page1);

      page2 = new ECNNewProjectWizardPageTwo();

      addPage(page2);

}
```

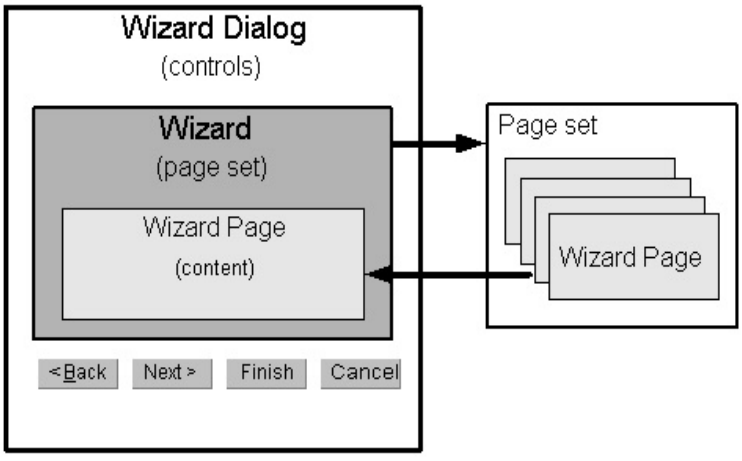


Abbildung 2.6: Die Struktur von Wizard in Eclipse

Die `WizardPage`-Klasse bestimmt mit Hilfe der `createControl()`-Methode das Aussehen des Inhalts der Wizardseite. Sie reagiert auf die Ereignisse in diesem Inhaltsbereich und entscheidet, wann die Seite fertig ist. Man erweitert normalerweise die `org.eclipse.jface.wizard.WizardPage`-Klasse, um die Wizardseiten zu erstellen. Innerhalb dieser Klasse kann man das Design von der Seite implementieren.

2.2.4 Einführung in Plug-In und PDE

Nach [18] ist die Definition einer Plug-In-Komponente ein Hilfsprogramm, das die Fähigkeiten einer größeren Anwendung erweitert. Ein Beispiel ist die Plug-In-Komponente der *Acrobat Reader*-Anwendung, das die Anzeige von PDF-Dokumenten direkt im Browser-Fenster ermöglicht. Das Wort in Deutsch bedeutet *Einschub*.

In Eclipse ist eine Plug-In-Komponente die kleinste Einheit der Eclipse-Plattform [19]. Typischerweise besteht eine Plug-In-Komponente aus Java-Quellcode in einer `jar`-Bibliothek, einiger *Read-Only*-Dateien zur Konfiguration, und anderen Ressourcen wie Bilder, HTML-Dateien, usw.

Jede Plug-In-Komponente besitzt eine Manifest-Datei, über die die Abhängigkeiten mit anderen Plug-In-Komponenten festgelegt wurden. Eine komplexe Plug-In-Komponente wird normalerweise in mehrere Plug-In-Komponenten zerlegt. Die Abhängigkeit erfolgt durch die Angabe von Erweiterungspunkten (*extension points*) und Erweiterungen (*extensions*) zu einem oder anderen Erweiterungspunkten in anderen Plug-In-Komponenten.

Ein Plug-In-Erweiterungspunkt ist von anderen Plug-In-Komponenten erweiterbar, so dass irgendeine Plug-In-Komponente über weitere Funktionalität von vorhandenden Plug-In-Komponenten verfügen kann.

Eclipse-Plattform bietet den Benutzern mittels der Plug-In-Entwicklungsumgebung ("*Plug-in Development Environment"*, PDE) das Werkzeug, mit dem man Plug-In-Komponenten erstellen, entwickeln, testen, debuggen, bauen und verteilen kann.

Plug-In-Komponenten sind üblicherweise im Verzeichnis `/eclipse/plugins/` zu finden. Beispielsweise ist die Plug-In-Komponente `org.eclipse.jdt.source_3.1.2`. Dabei entspricht der Ordnername dem Names der Plug-In-Komponente, auf den ein Unterstrich und dann die Versionnummer folgen.

Innerhalb eines Plug-In-Verzeichnises findet man die Dateien von folgenden Typen:

- `*.jar` ist der Java-Code der Plug-In-Komponente.
- `about.html` wird dann erschienen, wenn der Benutzer nach Infomationen zur Plug-In-Komponente fragt.
- `plugin.properties` enthält die Zeichenkette, die von `plugin.xml` genutzt werden.
- `plugin.xml` ist das Plug-In-Manifest, das die Plug-In-Komponente für Eclipse beschreibt.
- `lib/` ist ein Verzeichnis für zusätzliche `.jar`-Dateien.
- `icons/` enthält das von der Plug-In-Komponente verwendete Symbol.

Das allerwichtigste Element einer Plug-In-Komponente ist die `plugin.xml`-Datei, die dazu dient, Eclipse über die Plug-In-Komponente zu informieren. Die folgende einfachste `plugin.xml` informatiert Eclipse über Identifikator (ID), Namen, Versionnummer und Anbieter der Erweiterung.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
     id="org.fzi.ecn"
     name="ECN Plug-in"
     version="1.0.0"
     provider-name="FZI">
</plugin>
```

PDE stellt die Vorlagen bereit, die man gleich ausführen und bearbeiten kann. Unter anderem gibt es Vorlagen für Editoren, mehrfachseitige Editoren, Ansichten, usw. Fast alle möglichen Plug-In-Modelle kann man von der Vorlage gestalten.

Möchte man innerhalb von Eclipse eine eigene Plug-In-Komponente entwickeln, erstellt man ein Plug-In-Projekt. Dies öffnet den Plug-In-Manifest-Editor. Dieser zeigt die Karteikarte Übersicht (*Overview*) der Plug-In-Komponente. Außerdem gibt es weitere Karteikarten wie (siehe Abbildung 2.4):

- Abhängigkeit (*Dependencies*) gibt die Plug-In-Komponenten an, die für diese Plug-In-Komponente erforderlich sind.
- Laufzeit (*Runtime*) gibt die Bibliotheken an, die für die Ausführung diese Plug-In-Komponente benötigt sind.
- Erweiterungen (*Extensions*) geben die Erweiterungspunkte an, die diese Plug-In-Komponente verwendet.
- Erweiterungspunkte (*Extension Points*) geben die Erweiterungspunkte an, die von dieser Plug-In-Komponente definiert werden.
- `plugin.xml` öffnet einen XML-Editor, in dem man den Quellcode bearbeiten können.
- `MANIFEST.MF` definiert die Laufzeitansicht der Plug-In-Komponente und spezifisiert den Namen, die Versionnummer, den Klasse-Pfad der Plug-In-Komponente, usw.
- Übersicht (*Overview*) beschreibt die Inhaltsübersicht vom Plug-In-Manifest. Hier findet man die allgemeine Information über ID, Versionnummer, Namen, Klasse, und Anbieter. Man kann die Information hier ändern, oder aber in `MANIFEST.MF`, das ist gleich.

Die *Extensions*-Karteikarte zeigt an, wie diese Plug-In-Komponente die im System von anderen Plug-In-Komponenten bereitsgestellten Funktionalitäten erweitert, und entspricht dem Codestück in `plugin.xml`. Angenommen erzeugt man einen Erweiterungspunkt vom `org.eclipse.ui.newWizards`. Automatisch wird die XML-Datei um die neue Information vom neuen eingefügten Erweiterungspunkt geändert.

```
<extension point="org.eclipse.ui.newWizards">
...
```

```
</extesion>
```

Die häufig verwendeten Erweiterungspunkte sind u.a.:

- `org.eclipse.ui.acionSets` für Aktionen
- `org.eclipse.ui.views` für Sicht
- `org.eclipse.ui.editors` für Editoren
- `org.eclipse.ui.perspectives` für Perspektive

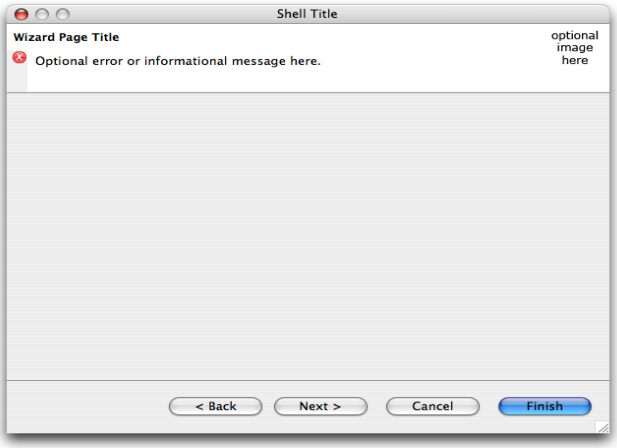
Die Erweiterungspunkte richtet die Definition der neuen Erweiterungspunkte ein, damit die anderen Plug-In-Komponenten ihre Funktionalitäten erweitern können.

Da die Erweiterungspunkte eine zentrale Rolle bei der Entwicklung der Plug-In-Komponente spielen, soll man sich gut damit auskennen.

2.3 Designgrundlagen GUI

Wie schon in Kapitel 2.2.1 erwähnt, ist Eclipse eine erweiterbare universelle Plattform. Dieser Vorteil kann jedoch als ein ernsthafter Nachteil angesehen werden, denn innerhalb der Eclipse-Plattform ist es nicht möglich, die Konsistenz der Benutzerschnittstelle zu gewährleisten. Dieser Abschnitt gibt deshalb einen kurzen Überblick über die Eclipse-Benutzerschnittstelle-Richtlinien [16], die für die allgemeine Bewertung von Programmier- und Entwicklungsassistenten (PEA) geeignet ist.

Das Standard-Design eines Assistenten ist in Abbildung 2.7 zu sehen. Der Assistent hat vier Bestandteile. Ganz oben befindet sich der Titel-Bereich. Er hat einen Fenster-Titel, einen Assistentseite-Titel, eine Nachricht, und ein Symbol. Dann kommt der Inhalt und nur hier kann man die Benutzerschnittstelle des Assistenten gestalten. Ganz unten sind die Navigations-Knöpfe zu finden. Nach Richtlinien 5.2 hat der Assistent in jeder Seite vier Knöpfe, **Back**, **Next**, **Finish**, und **Cancel**. Obwohl es die Möglichkeit besteht, im Assistent über eine Übersichtskarte zu verfügen, verwenden die Richtlinien sie aber nicht. Zwischen dem Knopfbereich und dem Inhalt ist noch ein Bereich für den Fortgangbereich und die Nachrichtenbereich.



Die Abbildung 2.7 : Ein Standard-Assistent

Neben den oben genannten Richtlinien ist eine Zertifizierung durch IBM namens RFRS (*Ready for Rational Software*) möglich. Ein Abschnitt von RFRS beschreibt die UI-Anforderungen [17]:

- ***Wizard look and feel (RFRS 3.5.2)*** sagt :

  *"Each wizard must contain a header with a banner graphic and a text area for user feedback. It must also contain **Back, Next, Finsih**, and **Cancel** buttons in the footer. A one-page wizard does not need to have the **Back** and **Next** buttons."*

- ***Open new file in editor (RFRS 3.5.6)*** sagt:

  "If a new file is created, open the file in an editor. If a group of files is created, open the most important, or central file, in an editor."

- ***New project switches perspective (RFRS 3.5.7)*** sagt:

  "If a new project is created, change the active perspective to suit the project type."

- ***Show new object (RFRS 3.5.8)*** sagt:

  "If a single new object is created, select and reveal the new object in the appropriate view. In case where the creation of a resource results in the creation of project or folder resources, the wizard should propose reasonable default locations."

- ***One-page wizard buttons (RFRS 5.3.5.13)*** sagt:

  *"A one-page wizard must contain the **Finish** and **Cancel** buttons, and should also contain grayed-out **Back** and **Next** buttons."*

Ausgehend von den Benutzerschnittstelle-Richtlinien und RFRS-Benutzerschnittstelle-Anforderungen soll eine Plug-In-Komponente entwickelt werden. Der in dieser Studienarbeit entwickelte PEA folgt ebenfalls diesen Richtlinien.

2.4 EventChannelNetwork (ECN)

ECN ist eine Programmbibliothek für asynchrones Nachrichtenkommunikation über sogenannte Nachrichtenkanäle in objektorientierten, verteilten Systemen. Um die Struktur verteilter Anwendungen mit ECN zu verstehen, bietet dieser Abschnitt einen kurzen Überblick. Weitere Informationen sind z.B in [20] zu finden.

ECN bietet eine Nachrichtenkommunikation nach dem Publiziere/Abonniere-Muster, mit Dienstnehmer- und Dienstgeber-Komponenten. Ein Kanal repräsentiert das Publiziere/Abonniere-Muster. Ein Dienstnehmer-Knoten A abonniert Nachrichten von Dienstgeber B. Nach dem Empfang einer Nachricht führt A eine bestimmte Aktion aus. Für Systeme mit Echtzeitbedingungen gelten hier zeitliche Anforderungen.

Ein Beispiel ist ein System mit Steueraktion, wie in [20] beschrieben. Innerhalb des Systems befinden sich vier Knoten (drei Dienstgeber und ein Dienstnehmer). Funktionsüberprüfung- , Lebenssignal- , und Temperatur-Knoten verhalten sich als Anbieter (Dienstgeber), und ein Controller-Knoten empfängt Nachrichten und führt Aktionen aus. Es werden drei Kanäle erzeugt (ein Funktionsüberprüfungs-, ein Lebenssignal- und ein Temperatursensor-Kanal), die

unterschiedliche Eigenschaften besitzen. Der Temperaturknoten schickt z.B. periodisch die aktuelle Temperatur an den Controller und dieser muss die Aktion *heizen* bzw. k*ühlen* ausgeführen.

Jeder Knoten wird in einer XML-Datei beschrieben, die alle wichtigen Informationen speichert. Die XML-Datei unterscheidet zwischen Dienstgebern und Dienstnehmern. Außer der die XML-Datei wird die Logik jedes Knoten auch mit einer Java-Klasse beschrieben.

2.5 Verwandte Arbeiten

Die Entwicklung von PEAs ist zahlreicher geworden [21], denn sie haben einen großen Vorteil, er spart viel Schreibarbeit, und vor allem die sich wiederholende Arbeit.

Eine verwandte Arbeit [22] beschreibt die Entwicklung eines Assistenten für Web-Entwicklung mit dem Titel “Design und Implementierung eines Projekt-Wizards”. [22] konzipiert und implementiert einen Assistenten, der Informationen sammelt, die die Ausgestaltung des Projekts regeln. Außerdem ist die Frage zu erklären, wie er sich verhalten soll. Das Ziel der Studienarbeit ist es, beim Anlegen eines neuen Projekts fallen eine Reihe von mechanischen Tätigkeiten an. Das geschieht, indem die Beschreibung des Projekts auf dem Assistenten abgebildet werden soll. Anhand der in der Projektbeschreibung enthaltenen Informationen sollen benötigte Werkzeuge durch den Assistenten automatisch eingerichtet werden. Die Infomationen, die in der Projektbeschreibung abgefragt werden sollen, sowie die durch den Assistenten einzurichtenden Werkzeuge, sollen ohne Eingriffe in den Programmcode des Assistenten frei konfiguriert werden können. Der Assistenten speichert alle Daten zu einem Projekt in deren Projektdatei in Form einer XML-Datei.

In [27] wurde ein PEA als Plug-In-Komponente entwickelt. Die Arbeit beschreibt die Entwicklung des *Eclipse Praktomat Plug-Ins.* Sie stellt den gesamten Entwicklungsprozess vom Entwurf bis hin zur Implementierung dar und dient für die Weiterentwicklung der Plug-In-Komponente. Der PEA sollte für das Datei-Hochladen, direkte Verbindungen als auch Verbindungen über einen Proxy-Dienstgeber unterstützen. Außerdem stellt die Arbeit über die Schwierigkeit bei der Entwicklung einer Plug-In-Komponente und allgemeine Tipps und Hinweise dazu dar.

Kapitel 3

Entwicklung der Plug-In-Komponente am Beispiel von ECN

Dieses Kapitel beschäftigt sich mit der Entwicklung des ECN-Plug-Ins. Beginnend mit den Schwierigkeiten bei der Programmierung von verteilter Anwendung mit ECN und den Möglichkeiten, die Eclipse bietet, werden die Benutzerführung bzw. -unterstützung des ECN-Plug-Ins diskutiert. Anschließend wird ein Überblick einer Bewertungsmetrik für Programmier- und Entwicklungsassistenten (PEA) gegeben, die später in dem nächsten Kapitel näher untersucht wird

3.1 Möglichkeiten

Zu Beginn des Entwurfs soll ein Überblick darüber gegeben werden, wie die XML-Dateien und Java-Klassen für eine ECN-Applikation aussehen. Dann wird darüber diskutiert, wie der Aufwand beim Programmieren reduziert bzw. vereinfacht werden kann.

Auftretende Schwierigkeiten

Bei ECN-Applikationen handelt es sich immer um Knoten, die sich entweder als Dienstgeber oder Dienstnehmer verhalten und innerhalb eines Kommunikationskanals interagieren. Die erste Schwierigkeit liegt darin, dass man die XML-Dateien für die Knoten anlegen muss. Hat man zehn Knoten, so gibt es auch zehn XML-Dateien zu erzeugen. Die einfachste Möglichkeit, um die nötigen XML-Dateien von jedem Knoten zu erstellen, ist sicher die *Copy-Paste*-Methode. Dieses Verfahren funktioniert sehr schnell, führt aber zu der unangenehmen Situation, dass man die Werte bzw. Attributen an die entsprechenden Knoten anpassen muss. Dienstnehmer benötigen ein Extra-Element `<proxy>`, das die Kopie eines Dienstgeber ist, aber mehr Schreibarbeit bewirkt und mehr Fehler erlaubt. Das ECN-Plug-In soll dem Programmierer die Möglichkeit bieten, die Erstellung von XML-Dateien zu vereinfachen und entsprechende Werte leichter zu modifizieren.

Jede XML-Datei besitzt ein Wurzelelement `<node>`. Das erste Kindelement `<rtsj>` beschreibt ein Profil, das für die verwendete Version der Echtzeit-Java-Implementierung ausgewählt wird:

```
<node name=”S” id=”1111”>
   <rtsj profile=”FSRTJ”>
      <factory>de.fzi.ecn.frt.adminchannel.AdminChannelEventChannelFactory</factory>
   </rtsj>
</node>
```

Als Profil-Attribut sind z.B. “HRTJ” für harte Echtzeitanforderungen, oder “FSRTJ” für flexible weiche Echtzeitanforderung möglich [26]. Die entsprechende Java-Klasse für eine Fabrikklasse der Nachrichtenkanäle wird hier angegeben.

Ein weiteres Kindelement heißt `<socket>` und bezeichnet die physikalische Netzwerkverbindung des Knotens. Es beinhalten die Angaben über die Sockel-Klasse, die Port-Nummer, Buffer, und verwendete Wrapperklassen. Für FSRTJ-Systeme sind mehrere Wrapper möglich, während für HRTJ nur ein einziger unterstützt wird. Innerhalb des Wrapper-Elements sind die entsprechende Wrapper-Klasse und die Referenz auf den Kanal enthalten. Da diese Daten voneinander abhängen, müssen sie konsistent geändert werden. Folgendes Beispiel gibt die Angaben über einen Knoten mit dem HRTJ-Profil, der an einen Kanal beteiligt ist:

```
<socket name="NW">
   <class>de.fzi.ecn.frt.UDPMultiCastSocket</class>
   <attributes>
      <value>14474</value>
   </attributes>
   <buffer>4096</buffer>
   <wrapper name="TemperatureSocket" id="1">
      <class>de.fzi.ecn.hrt.StaticIntEventWrapper</class>
      <channelref
         ref="TemperaturSensor.Server.TemperatureChannel" id="1" />
   </wrapper>
</socket>
```

Im Folgenden unterscheidet sich der XML-Code durch die Rolle des Knotens. Ein Dienstgeber enthält das Kindelement `<server>`, Dienstnehmer-Komponente beschreibt `<client>`. Der Code fängt mit dem Namen des Dienstgebers oder Dienstnehmers an, gefolgt mit den Angaben über den Kanal, woran er teilnimmt. Außerdem sind die anderen Informationen hier zu finden.

Bei Dienstnehmer sieht der Code etwas länger aus. Nach der Angabe über den Namen des Dienstnehmers kommt das Kindelement `<handler>`, das den Aktion-Klassenamen enthält. Der Code ist durch die Angabe über die Kanäle beendet, die der Dienstnehmer läuscht.

Neben der XML-Datei müssen Java-Klassen für die Kommunikationslogik erstellt werden. Sowohl Dienstgeber als auch Dienstnehmer sind einer oder mehreren Java-Klassen zugeordnet, die typischerweise aus einem Konstruktor, einer `run()`- und einer `main()`-Methode bestehen. Aktionklassen, die nur der Dienstnehmer besitzt, enthält normalerweise einen Konstuktor, eine `run()`- und eine `getAction()`-Fabrikmethode. Jede Klasse muss eine Reihe von vorgefertigten Klassen importieren, wie u.a. `de.fzi.ecn.EventChannelNetwork`.

Möglichkeiten

ECN-Applikationen zu entwickeln, kostet viel Schreibarbeit und kann bei den Änderungen (*Copy-Paste*-Methode) leicht zu fehlerbehaftem Code führen. Diese Tatsache soll mit Hilfe des

ECN-Plug-Ins verbessert werden. Außerdem soll die Modifizierung der XML-Beschreibung vereinfacht werden.

Von den oben genannten Herausforderungen ausgehend wird nun der Entwurf des ECN-Plug-Ins diskutiert, Ausgangspunkt sind die Möglichkeiten, die die Eclipse-Plattform bietet.

Innerhalb von Eclipse werden die neuen Dateien vom Assistenten mithilfe der Wizard-Bibliothek (siehe Kapitel 2.2.3) erstellt. Der Assistent sammelt Infomationen vom Benutzer, unter anderem:

- Anzahl der zu erzeugenden Knoten
- Die Rolle von jedem Knoten, ob Dienstgeber oder Dienstnehmer
- Die Beschreibung von jedem Dienstgeber und Dienstnehmer

Die Infomationen werden bei der Erzeugung von Java-Klassen und XML-Beschreibung verwaltet. Dabei unterscheidet man zwischen optionalen und erforderlichen Informationen. Diese können, beispielsweise, die Angabe darüber, wo die Dateien gespeichert sind, wie viele Knoten, und die Namen der Knoten. Zum anderen, optionale Angaben, die sich später im Editor eingeben oder ändern lassen. Diese sind z.B. die Angabe über die Send- und Empfangszeit von Nachrichten eines Kanals.

Die Dialog-Bibliotheken eignen sich für die Angabe über die jeweiligen Knoten, Kanälen oder Sockeln. Die Dialoge sind nicht anderes als ein Assistent, der nur eine Seite enthält. Sie fragen vom Benutzer die Eingabe über z.B. Knoten, Kanäle, Sockeln usw. ab.

Die Eclipse-Forms-Bibliotheken (siehe Kapitel 2.2.2 Eclipse Forms) können gut als Editoren für XML-Beschreibung verwendet werden. Dies hat das Ziel, die Modifizierung von XML zu vereinfachen, statt direkt im Quellcode die Änderungen vorzunehmen.

Die Erstellung eines ECN-Projekts durch das ECN-Plug-In geht , wie in folgender Abbilung beschrieben, vor:

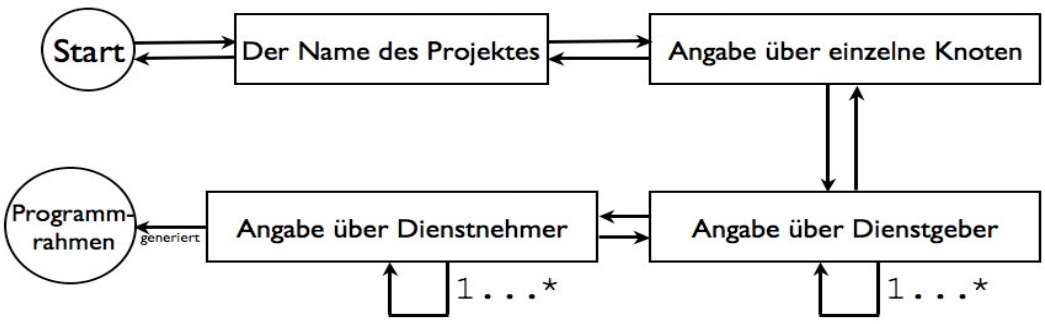


Abbildung 3.1 : Der Verlauf des Assistenten

Die obige Abbildung lässt sich in vier Klasse transformieren (der Reieh nach):

- `ECNNewProjectWizardPageOne`
- `ECNNewProjectWizardPageTwo`

- `ECNNewProjectWizardPageServer`
- `ECNNewProjectWizardPageClient`

3.2 Benutzurführungen/ -unterstützung

Anhand eines Beispiels [20] werden die Benutzerführungen und -unterstützung des Eclipse-Plug-Ins beschrieben. In diesem Beispiel sind vier Knoten mit dem harten Echtzeitprofil ("HRTJ") beteiligt, die aus drei Dienstgebern und einem Dienstnehmer besteht (vgl. Kapitel 2.4). Für jeden Knoten generiert der Assistent die entsprechende XML-Datei. Außerdem müssen drei logischen Kanäle mit den entsprechenden Eigenschaften definiert sein. Also :

- Vier Knoten : Funktionsprüfung, Lebenssignal, Temperatur als Dienstgeber und Controller als Dienstnehmer.
- Drei Kanäle : Funktionsprüfung-, Lebenssignal- und Temperatursensor-Kanal.

Das Eclipse-Plug-In verwendet folgende API-Bibliotheken:

1. Wizard und Dialog

   Dieser Assistent hat die Aufgabe, das Rahmenwerk der ECN-Applikation anzulegen. Das heißt, nach dem erfolgreichen Vorgang bekommt der Programmierer die XML-Dateien von jedem Knoten und auch die entsprechenden Java-Klassen.

2. Eclipse-Forms

   Die Aufgabe dieser Assistenten ist, dem Programmierer den Quellcode anzuzeigen und die Änderungen zu vereinfachen.

1. Wizard und Dialog

Die Verwendung dieser Bibliotheken hilft dem Programmierer bei der Generierung des ECN-Rahmenwerks. Folgendes ist die Beschreibung des Vorgangs (nach Abbildung 3.1):

- **Erste Seite des Assistenten**

  In der ersten Seite gibt der Programmierer die Angabe über den Namen für das Projekt und das Echtzeitanforderungsprofil ein.

- **Zweite Seite des Assistenten**

  Die nächste Seite beschäftigt sich mit der Erstellung der Knoten. In der Abbildung 3.2 sind ein Behälter und mehrere Knöpfe zu sehen.

  Wird der *Add Node*-Knopf gedrückt, taucht ein Dialog auf, der die Eingabe vom Programmierer verlangt. Dementsprechend gilt es fürs Anlegen der Sockeln. Alle erzeugten Knoten zeigt der Behälter an. Wie im Beispiel, hier erzeugt der Programmierer vier Knoten, und jeweiliger Knoten besitzt einen Sockel namens "NW", der schließlich durch einen Wrapper gewickelt ist.

  In der Abbildung 3.2 sind die vier gestellten Knoten mit Sockeln und Wrappern zu sehen.

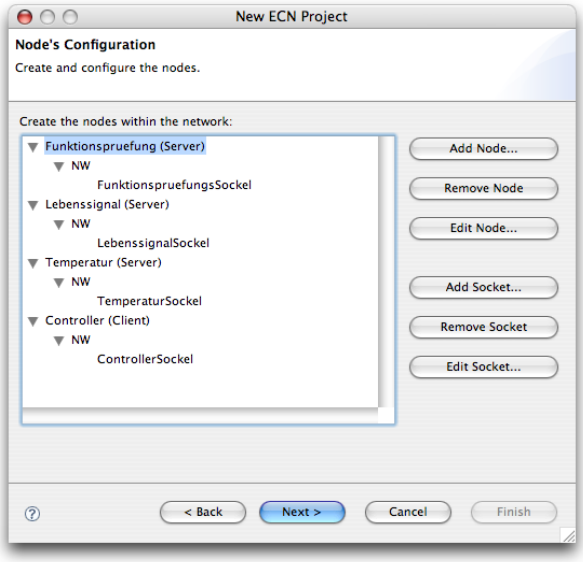


Abbildung 3.2: Die zweite Seite des Assistenten

- **Dienstgeberseite des Assistenten**

  Diese Seite stellt einen jeweiligen Dienstgeber dar und wiederholt sich drei mal, da drei Dienstgeber. Sie erlaubt dem Programmierer, den Namen des Dienstgebers zu vergeben und die Kanäle anzulegen. Wie die Abbildung 3.? veranschaulicht, besitzt der Funktionsprüfung-Knoten einen Kanal namens “Funktionsprüfungskanal”. Die Werte lassen sich mittels eines Dialogs setzen, der durchs Drücken des *Add Channel*-Knopf erscheint.

- **Dienstnehmerseite des Assistenten**

  Ähnlich wie die Dienstgeberseite funktioniert die Dienstnehmerseite. Hier kann der Programmierer die Handler für jeden Dienstnehmer erzeugen (im Beispiel nur einen einzigen). Er hat drei Handlers, die für jeden Dienstgeber zuständig sind.

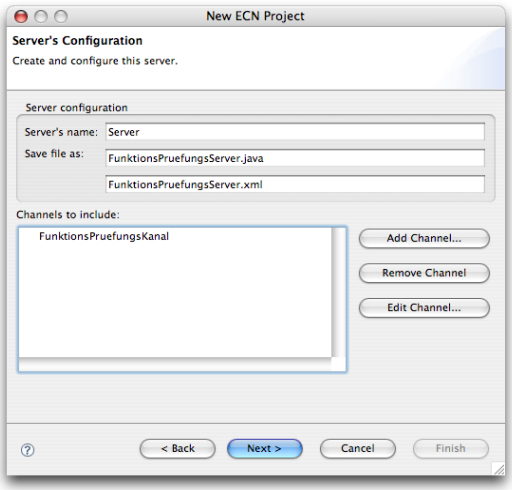


Abbildung 3.3: Der Funktionsprüfung-Knoten wird konfiguriert

Nun ist der Assistent fertig und der Programmierer kann auf den *Finish*-Knopf drücken. Der Assistent stellt dann das gewünschte Rahmenwerk der ECN-Applikation.

2. Eclipse-Forms

Durch diese Bibliothek wird der Editor implementiert. Er ermöglicht die Änderung des vom Wizard erzeugten Quellcodes. Bisher bekommt der Programmierer einen reinen Text-Editor für XML-Dateien und einen Java-Editor für Java-Klassen. Bezüglich der letzten genannten ist die Verwendung vom Java-Editor schon geeignet. Ein reiner Text-Editor für XML-Datei scheint nicht so hilfreich zu sein, da es an Funktionalitäten mangelt. Zum Beispiel, es gibt keine Ansicht von bestimmten Werten, die man eventuell umsetzen möchte. Die PDE verwendet deshalb den mehrfachseitigen Editor, der ein Formular und dazu einen reinen Text-Editor hat.

Das Formular benutzt den *Master/Details*-Entwurf [15]. Es besitzt eine Liste oder einen Baum (Master), und eine Gruppe von Eigenschaften (Details). Die Idee des Entwurfs ist, die Eigenschaften eines selektierten Listelements anzuzeigen und ihre Änderung möglich zu machen. Dadurch lässt sich der Code einfacher und angenehmer modifizieren. Außerdem werden dadurch die möglichen Schreibfehler minimiert, indem das Formular die geeigneten Benutzerschnittstelle-Komponenten verwendet, wie z.B. `Combo`, `Checkboxen`, usw.

Die Abbildung 3.4 zeichnet den Editor einer XML-Datei von einem Knoten mit dem *Master/Details*-Entwurf aus.

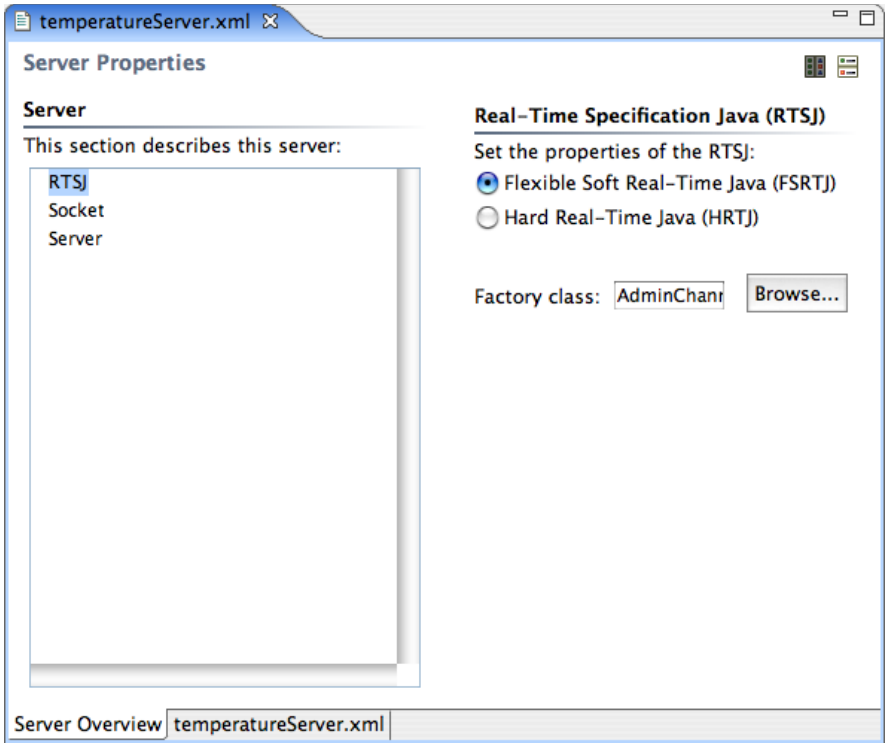


Abbildung 3.4: Der formularorientierte Editor für Modifizierung der XML-Datei

3.3 Anforderungen Bewertungen des Plans

Der Programmierer- und Entwicklungsassistent (PEA) als ECN-Plug-In hat das Ziel gesetzt, zum einen den Programmierer beim Programmieren zu untestützen. D.h. Das ECN-Plug-In erstellt ein Programm-Rahmenwerk der ECN-Applikation. Falls Änderungen vorzunehmen sind, bietet das ECN-Plug-In eine angenehme Arbeitsweise dank Eclipse-Forms.

Zum anderen, das ECN-Plug-In soll nach der Eclipse-Benutzerschnittstelle-Richtlinien gestaltet werden, so dass das Endprodukt sich wie eine native Applikation verhält und sich an das "*look and feel"* der entsprechenden Betriebssystemen anpassen.

Die folgenden zwei Hinweisen sind bei der Entwicklung hilfreich und sollten berücksichtigt werden [11]:

- Man achte darauf, dass keine neue Konzepte für die Bedienerführung erfunden werden. Die bestehenden Konzepte decken in der Praxis fast immer 100% der Bedürfnisse ab.
- Man achte beim Aufbau der Plug-In-Komponente, dass die strengen Regeln der API-Bibliotheken nicht verletzt sind. Das Konzept von Eclipse verlangt nach einer maximalen Wiederverwendung der bestehende Bibliotheken.

Von den oben genannten Anforderungen muss der PEA die entsprechende Leistung erbringen. Das Kapitel 5 diskutiert die Ergebnisse der Plug-In-Komponente und weitere Bewertungsmetrik.

Kapitel 4

Implementierung des ECN-Plug-Ins

Dieses Kapitel stellt die Implementierung des ECN-Plug-Ins als Prototyp. Ausgehend von der im vorherigen Abschnitt (Kapitel 2.2.3) beschriebenen Benutzerschnittstelle wird hier implementiert.

4.1 Eclipse Wizard und Dialog

Die PDE (Kapitel 2.2.4) bietet schon die Vorlage namens **New File Wizard** mit dem Erweiterungspunkt `org.eclipse.ui.newWizards`. In `plugin.xml` sieht der Code folgendes aus:

```
<extension point="org.eclipse.ui.newWizards">
```

Die Wizard-Klasse bestimmt das Verhalten und das Aussehen vom gesamten Assistenten, wie den Titel oder das Bild. Sie ist eine abstrakte Klasse, die die Schnittstelle `Iwizard` implementiert, und befindet sich im Verzeichnis `org.eclipse.jface.wizard.Wizard`.

Die Unterklasse kann folgende Methoden aufrufen bzw. überschreiben:

- `addPage`
- `setHelpAvailable`
- `setDefaultPageImageDescriptor`
- `setDialogSettings`
- `setNeedsProgressMonitor`
- `setTitleBarColor`
- `setWindowTitle`
- `createPageControl`
- `performCancel`
- `addPages`
- `performFinish`
- `dispose`

Jede Seite des Assistenten muss die Oberklasse `org.eclipse.jface.wizard.WizardPage` erweitern, um das Verhalten einzelner Seiten festzulegen. Außerdem wird die Schnittstelle Listener aus dem Paket `org.eclipse.swt.widgets` verwendet, um die Ereignisse abzufangen.

Folgende Methoden lassen sich von der Unterklasse aufrufen bzw. überschreiben:

- `setDescription`
- `setErrorMessage`
- `setImageDescriptor`
- `setMessage`
- `setPageComplete`
- `setPreviousPage`
- `setTitle`
- `canFlipToNextPage`
- `isPageComplete`
- `addPages`
- `dispose`

Der ECN-Wizard-Assistent besteht aus vier Seiten (vgl. Abbildung 3.?) :

- `ECNNewProjectWizardPageOne`

- `ECNNewProjectWizardPageTwo`
- `ECNNewProjectWizardPageServer`
- `ECNNewProjectWizardPageClient`

und einem Behälter namens `ECNNewProjectWizard`.

In der folgenden Abbildung ist das UML-Diagramm vom Assistenten zu sehen.

`ECNNewProjectWizard.java`

- Von jeder Seite muss eine Instanz erzeugt werden. Sie werden nach der Reihenfolge in die `addPages()`-Methode eingefügt.
- Bevor Eclipse irgendwelche Informationen an den Assistenten übergibt, initialisiert der Konstruktor ihn. Das gilt z.B. für dessen Titel oder Symbol.

```
super();
setWindowTitle("Create an ECN project");
setDefaultPageImageDescriptor(
      AbstractUIPlugin.imageDescriptorFromPlugin(
            "ECN", "icons/ecnnewproject.gif"));
```

- Die `performFinish()`-Methode wird dann aufgerufen, wenn der Zustand vom Assistent fehlerfrei ist. Sonst `return false`.

`ECNNewProjectWizardPageOne.java`

- Nun fängt die erste Seite mit der Abfrage nach dem Projektnamen und dem Echtzeit-Profil an. Im Konstruktor lassen sich der Titel und die Beschreibung benennen.

```
setTitle("ECN Project");
setDescription("Create an ECN project.");
```

- Die `createControl()`-Methode ist für die Darstellung der Seite zuständig.

```
public void createControl(Composite parent) {
      Composite composite = new Composite(parent, SWT.NULL);
      GridLayout gridLayout = new GridLayout();
      gridLayout.numColumns = 3;
      composite.setLayout(gridLayout);

      createProject(composite);

      Group group = new Group(composite, SWT.NULL);
```

```
        group.setText("RTSJ Property");
        GridData gridData = new GridData(GridData.FILL_HORIZONTAL);
        gridData.horizontalSpan = 3;
        group.setLayoutData(gridData);
        createRTSJGroup(group);


        setControl(composite);
    }
```

- Die Klasse implementiert die `Listener`-Schnittstelle, damit die Ereignisse behandelt werden können. Dafür ist die `handleEvent()`-Methode verantwortlich.
- Falls die Angabe nicht korrekt eingebeben ist, taucht eine entsprechende Fehlermeldung auf.

ECNNewProjectWizardPageTwo.java

- In dieser Seite werden die Knoten mittles des *Add*-Button erzeugt. Um die Knoten darzustellen, wird die Klasse org.eclipse.`jface.viewers.TreeViewer` verwendet.
- Ein Dialog kommt vor, wenn man den *Add*-Knopf drückt. Das Dialog ist eine eigene Klasse, die die Klasse `org.eclipse.jface.dialogs.Dialog` erweitert.
- Die überschriebene Methode ist `createDialogArea()`. Die Methode stellt die Benutzerschnittstelle-Elemente vom Dialog dar.

ECNNewProjectWizardPageServer.java

- Jetzt kommt die Seite vom Dienstgeber, die sich so oft wiederholt, wie die Anzahl von ihm.
- Die Seite kümmert sich vor allem um die Erzeugung eines Kanals, der einem Dienstgeber gehört.

ECNNewProjectWizardPageClient.java

- Ähnlich wie vorherige Seite ist diese Seite für die Einstellung des Dienstnehmer zuständig. Sie wiederholt soviel wie die Anzahl der Dienstnehmer.
- Hier kann man die Aktivität vom Dienstnehmer festlegen, durch die Erzeugung vom `Handler`.

4.2 Eclipse Forms

Wie schon besprochen im Kapitel 3.2 hat der Assistent zwei Ansichten, eine ist für das Formular und die andere für den normalen Editor. Die Verwendung vom Formular hat das Ziel gesetzt, dem Programmierer die Möglichkeit zu bieten, die Änderung vom XML-Werte zu vereinfachen und die möglichen Schreibfehler zu minimieren.

Bevor man anfängt zu implementieren, muss eine neue Abhängigkeit *(Depedencies)* `org.eclipse.ui.forms` eingefügt werden. Die Formulare-Programmierschnittstelle verfügt über folgende Pakete:

- `org.eclipse.ui.forms` : Das Hauptpaket
- `org.eclipse.ui.forms.editor` : Die Klassen, um mehrfachseitige Editoren wie in PDE zu gestalten
- `org.eclipse.ui.forms.events` : Neue Ereignisse, die die formular-orientierten UI-Komponente untestützen
- `org.eclipse.ui.forms.widgets` : Eine Gruppe von neuen UI-Komponenten, spezifisch für Formulare

Der Assistent nutzt den *Master/Details*-Entwurf (vgl. Kapitel 3.2) und dafür einen mehrfachseitigen Editor. Der Editor besteht aus zwei Karteikarten, für die formularorientierte Benutzeroberfläche und den Text-Editor.

Um einen mehrfachseitigen Editor zu implementieren, muss die Klasse `FormEditor` erweitert werden. Sie ist eine abstrakte Klasse, die die Oberklasse `MultiPageEditor` veerbt, und bildet die Basis auf den mehrfachseitigen Editor. Die Unterklasse implementiert die `addPages()`-Methode, um die Seite mittels der `addPage()`-Methode einzufügen.

Andere wichtigete Methode, die zu implemetieren sind:

- `doSave`
- `doSaveAs`
- `isSaveAllowed`

`ECNFormEditor.java`

- Die Klasse erweitert ebenfalls die Klasse `FormEditor` für die Basis auf mehrfachseitegen Editor.
- Die `addPages()`-Methode legt zwei weitere Editoren an.

```
protected void addPages() {
      try{
            addPage(new XMLFormPage(this));
            addPage(new XMLPage(this));
      }
      catch (PartInitException e) {
```

```
        }

    }
```

XMLFormPage.java

Die Klasse ist nach dem Master/Details-Entwurf konzipiert und muss folgendes tun:

- Den Hauptteil erstellen

```
protected void createMasterPart(final ManagedForm, Composite parent){

      final ScrolledForm form = managedForm.getForm();

      FormToolkit toolkit = managedForm.getToolkit();
      Section section = toolkit.createSection(parent,
                              Section.DESCRIPTION);
      section.setText("Model Objects");

      TableViewer viewer = new TableViewer(t);
      // ...

      viewer.setContentProvider(new MasterContentProvider());
      viewer.setLabelProvider(new MasterLabelProvider());
      viewer.setInput(page.getEditor().getEditorInput());
}
```

- Die Aktion für Form Tool Bar zur Verfügung stellen

```
protected void createToolBarActions(IManagedForm managedForm) {

      final ScrolledForm form = managedForm.getForm();
      Action haction = new Action("hor", Action.AS_RADIO_BUTTON) {
            public void run() {
                  sashForm.setOrientation(SWT.HORIZONTAL);
                  form.reflow(true);
            }
      };
      Action vaction = new Action("ver", Action.AS_RADIO_BUTTON) {
            public void run() {
                  sashForm.setOrientation(SWT.VERTICAL);
                  form.reflow(true);
            }
      };
      vaction.setToolTipText("Vertical orientation");
      vaction.setImageDescriptor(ExamplesPlugin.getDefault()
                  .getImageRegistry().getDescriptor(
                        ExamplesPlugin.IMG_VERTICAL));
      form.getToolBarManager().add(haction);
      form.getToolBarManager().add(vaction);
}
```

- Die detailierten Ansichten von jedem Listelement regiestrieren.

```
protected void registerPages(DetailsPart detailsPart) {

      detailsPart.registerPage(
            TypeOne.class, new TypeOneDetailsPage());
      detailsPart.registerPage(
            TypeTwo.class, new TypeTwoDetailsPage());
}
```

XMLPage.java

- Die Klasse ist für die Darstellung des reinen Text-Editor zuständig.

Kapitel 5

Ergebnisse

Dieses Kapitel beschreibt die Ergebnisse der Studienarbeit. Eine Beschreibung guter Programmier- und Entwicklungsassistenten (PEA) wird hier eingeführt. Ergebnis wie Eclipse-Plug-In-Komponenten entworfen und implementiert werden sollen. Am Ende beschreibt es "*lessons learned*" und die nötigen Arbeitsschritte, um Eclipse mit der Plug-In-Komponente zu erweitern.

5.1 Gute und schlechte Eclipse-Plug-Ins

Die Plug-In-Komponente in Eclipse ist, wie bereits im Kapitel 1 erwähnt, ein großer Vorteil, damit Eclipse sehr flexibel und einfach zu erweitern ist. Sie kann aber eine Schwachstelle sein, wenn bei der Erweiterung einige Dinge nicht beachtet werden, und darüber wird in diesem Abschnitt besprochen.

Die Entwicklung einer Plug-In-Komponente ist nicht anderes als die Entwicklung einer Software. Praktisch läßt sich die Entwicklung nach dem Softwareentwicklungsprozess [23] vorgehen. Eclipse mit PDE stellt aber einige Plug-In-Vorlagen zur Verfügung, die bereits weiterzuentwickeln sind. Man muss sich aber schon Vorstellungen machen, wie die Plug-In-Komponente aussehen und funktionieren muss. Je nach Größe und Komplexität sind solche Voruntersuchungen notwendig, sowie Modellierungsverfahren (z.B. UML).

Eine gute Planung verspricht ein gutes Ergebnis und ist deshalb ein entscheidender Punkt. Ohne solide Planung beschäftigt man sich mit unsicherer Gestaltung der Plug-In-Komponente, was dementsprechend Zeit kosten kann.

Die wichtigste Anforderung an gute Plug-In-Komponente ist natürlich, dass sie richtig funktionieren muss. Außerdem berührt die Anforderung darauf, wie einheitlich sie mit Eclipse integriert, zudem sie muss sich auf das native "*look and feel*" des Betriebssystems auswirken. Eine Plug-In-Komponente, die nicht nach der Eclipse-Benutzerschnittstelle-Richtlinie [16] gestaltet sind, gehört nicht zu einer guten Plug-In-Komponente. Der Einsatz der Richtlinien führt zu einer großartigen Konsistenz der Eclipse-Plattform. [16] deckt alle möglichen Benutzerschnittstellerichtlinien ab. Außerdem stellt IBM eine Qualitätsnorm in *Ready for Rational Software (RFRS)*-Zertifizierung zur Verfügung. Es wird damit erwartet, dass die neuen Plug-In-Komponenten sie festhalten [17].

Die Plug-In-Komponente soll nach dem Konzept der Trennung zwischen Model und Darstellung entwickelt werden. Die Code-Trennung macht eine saubere Implementierung, um eine spätere Änderung oder Erweiterung einfach zu halten und die Wiederverwendbarkeit der Komponenten zu ermöglichen.

Zusammenfassend sind die folgenden Punkte für die Bewertung von PEA als Eclipse-Plug-In-Komponente identifiziert worden:

- Die Wiederverwendbarkeit der API-Bibliotheken.

- Eine saubere Implementierung durch Trennung zwischen Model und Darstellung.
- Die Benutzerschnittstelle der Plug-In-Komponente muss unter Berücksichtigung der Eclipse-Benutzerschnittstelle-Richtlinien entwickelt werden. Um ein hohes Standard zu erreichen, dient die RFRS-Zertifizierung von IBM.
- Als eine härtere Anforderung soll der PEA die Namen der Paketen bzw. Klassen nach der Konvention [27] geben.

5.2 "Lessons learned"

Nachdem der PEA als eine Eclipse-Plug-In-Komponente entwickelt wurde, lassen sich einige wichtige Erfahrungen nennen.

Bevor man mit der Entwicklung startet, müssen ein Paar Fragen beantwortet werden:

1. Wie soll die neue Plug-In-Komponente den Programmierungsaufwand verringern?
2. Handelt es sich um Ansichten, Editoren, Perspektiven, allgemeine Prozessunterstützungs-Assistenten, usw. handelt?
3. Welche Erweiterungspunkte in Eclipse sollen verwendet werden?

Für Eclipse-Plug-In-Komponente stellt die Eclipse-Plattform eine eigene Umgebung namens PDE zur Verfügung. Durch die von PDE bereitsgestellten Vorlagen kann man mit der Entwicklung eigener Plug-In-Komponente beginnen. Im Prinzip besteht der Vorgang aus drei Arbeitsschritten [25]:

1. Eine neue Erweiterung deklarieren.
2. Die Deklaration in `plugin.xml` mit spezifischen XML-Elementen erweitern.
3. Die Java-Klassen schreiben.

Die erste zwei Schritte sind einfältig und erfolgen sowohl in Karteikarte `Extensions` als auch in `plugin.xml`. Aber die Schwierigkeit ist, relevante Erweiterungspunkte zu finden. Ein grundlegendes Verständnis der Konzepte hinter Eclipse ist unbedingt eine Voraussetzung. Deshalb ist eine Einarbeitung in Eclipse ein richtiger Anfang.

Der letzte Schritt fordert einen guten Entwurf von Java-Klassen an. Wie in 5.1 angesprochen, soll die Implementierung der Trennung zwischen Model und Darstellung folgen. Gute Einarbeitung in Eclipse vereinfacht die Implementierung, vor allem bei der Verwendung der bereitsgestellten API-Bibliotheken von Eclipse.

Um das Design der neuen Plug-In-Komponente völlig integriert zu sein, muss sie an die Benutzerschnittstelle-Richtlinien gestaltet werden. Dies umfasst vor allem Kleinigkeiten, die aber sehr wesentlich für die Darstellung und gute Integration in Eclipse sind. Dazu gehören beispielsweise die Großschreibung, das Symbol in der Direktstartleiste, die verwendete Farbe, usw. Schließlich, um ein hohes Standard zu erreichen, soll die Plug-In-Komponente die RFRS-Zertifizierung berücksichtigen.

Kapitel 6

Fazit

Eclipse IDE als eine universelle Plattform lässt sich durch Plug-In-Komponenten erweitern. Für die Entwicklung wird PDE zur Verfügung gestellt. Die in dieser Studienarbeit entwickelte Plug-In-Komponente soll als PEA (Programmier- und Entwicklungsassistent) die Arbeit bei dem Entwurf verteilter Systeme mit ECN-Kommunikationsinfrastruktur vereinfachen. Der Assistent generiert ein Rahmenwerk für die Programmierung von klassen mit ECN und bietet einen Editor für die Quellen.

Die Erweiterungspunkte (*extension points*) erlauben die Verwendung von Eclipse-Bibliotheken, wie z.B. `org.eclipse.ui.newWizards`. Da Eclipse aus dem MVC-Entwurfsmuster aufgebaut ist, wird die Wiederverwendbarkeit der API-Bibliotheken erleichtert. Die SWT- und JFace-Bibliotheken stellen die Benutzerschnittstelle-Klassen bereit und unterstützen die Programmierung der neuen Plug-In-Komponenten, z.B. durch ihre Oberklassen, abstrakte Klassen, so dass das Rad nicht neu erfunden werden muss.

Die Ergebnisse der Arbeit (vgl. Kapitel 5) haben konkrete Auswirkungen auf die Entwicklung und Bewertung von PEA im Allgemeinen und speziell bei der Nutzung von Eclipse.

Die Erweiterung von Eclipse durch die Plug-In-Komponente darf aber nicht die Konsistenz der Eclipse-Benutzerschnittstelle verletzen. In [16] wird die Eclipse-Benutzerschnittstelle-Richtlinien ausführlich beschrieben. Außerdem stellt IBM in *Ready for Rational Software* (RFRS) eine Zertifizierung, damit eine Plug-In-Komponente hohem Standard entspricht.

Ein guter PEA muss diese Anforderungen erfüllen und ist dafür verantwortlich, den Softwareentwicklungsprozess zu unterstützen und zu vereinfachen. Eine nahtlose Integration in Eclipse muss die Plug-In-Komponente aber auch anbieten.

Literatur

[1] NetBeans IDE. Webseite. http://www.netbeans.org, 2006.

[2] Borland IDE: JBuilder – Tools for Java and J2EE Application Development. Webseite. http://www.borland/jbuilder, 2006.

[3] BlueJ – The interactive Java environment. Webseite. http://www.bluej.org, 2006.

[4] IntelliJ IDEA The Most Intelligent Java IDE. Webseite. http://www.jetbrains.com/idea, 2006.

[5] Eclipse – an open development platform. Webseite. http://www.eclipse.org, 2006.

[6] Langer, I., Schulz v. Thun, Tausch, R. *Verständlichkeit in Schule, Verwaltung, Politik und Wissenschaft. Wissenpsychologie*. München, 1988.

[7] Daum, Berthold. Java Entwicklung mit Eclipse 3.1 : Anwendungen, Plugins und Rich Clients. Dpunkt-Verlag, 2005.

[8] What is Eclipse. Webseite. http://www.eclipse.org/org, 2006.

[9] Eclipse Projectss. Webseite. http://www.eclipse.org/projects, 2006.

[10] Eclipse. NASA Uses Eclipse for Interplanetary Operations. Webseite. http://www.eclipse.org/community/casestudies/NASAfinal.pdf, 2006.

[11] Assisi, Ramin. Eclipse 3 – Einführung und Referenz: Java Entwicklung mit der Open Source Plattform. Hanser, 2005.

[12] Department of Computer Science, Cornell University. Graphical User Interfaces (GUIs) Webseite. http://www.cs.cornell.edu/courses/cs211/2001fa/handouts/gui.pdf, 2006.

[13] Feigenbaum, Barry. SWT, Swing or AWT: Which is right for you? IBM Webseite. http://www-128.ibm.com/developerworks/library/os-swingswt/index.html, 2006.

[14] Batty, C., Scuse. Installing Eclipse. Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada, http://www.cs.umanitoba.ca/~eclipse/ 2006.

[15] Glozic, Dejan. Eclipse Forms: Rich UI for the Rich Client. A Eclipse Corner Article. http://www.eclipse.org/articles/Article-Forms/article.html, 2005.

[16] Edgar, N., Haaland, K., Li, J., Peter, K. Eclipse User Interface Guidelines Version 2.1.. A Eclipse Corner Article. http://www.eclipse.org/articles/Article-UI-Guidelines/Contents.html, 2004.

[17] Clayberg, Eric. Eclipse: building commercial-quality plug-ins. Addison-Wesley, 2006.

[18] Der Brockhaus. Computer und Informationstechnologie Fachlexikon. Brockhaus, 2003.

[19] Eclipse Platform Technical Overview. Eclipse Corner Article Webseite. www.eclipse.org/whitepapers/eclipse-overview.pdf, 2006.

[20] HIJA EU Project-Partner. D7.4 Evaluation of HIJA-Technology and Methodology Handbook, to appear December 2006.

[21] Eclipse IDE – Plugins. Wikipedia, 2006

[22] Leuchsenring, Stefan. Implementierung eines Projekt-Wizards. Studienarbeit im Fach Informatik. Friedrich-Alexander-Universität Erlangen-Nürnberg, 2004.

[23] Balzert, Helmut. Lehrbuch der Software-Technik: Software-Entwicklung. Spektrum, 2001.

[24] Jenifer, Tidwell. Designing Interfaces. O'Reilly, 2006.

[25] Proulx, Emmanuel. Eclipse Plugins Expose, Part 2: Simple GUI Elements. O'Reilly ONJava.com. http://www.onjava.com/pub/a/onjava/2005/03/30/eclipse.html, 2006.

[26] Schanne, Marc. Methodik für den Einsatz asynchroner Kommunikation beim Entwurf von verteilten, objektorientierten Systemen mit Echtzeitanforderungen. EventChannelNetwork-Webseite. http://www.eventchannelnetwork.org, 2006.

[27] Schneider, Christian. Entwicklung des Eclipse Praktomat Plug-Ins.Uni-Passau Webseite. http://www.infosun.fmi.uni-passau.de/st/staff/stoerzer/projects/Praktomat/PiE/entwicklung.pdf

[28] IBM Press Room Webseite. http://www-03.ibm.com/press/us/en/pressrelease/1025.wss