Diplomarbeit

Vladislav Neumüller

Analyse der Möglichkeiten zur Erstellung einer Entwicklungsumgebung für eine proprietäre Sprache

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Vladislav Neumüller

Analyse der Möglichkeiten zur Erstellung einer Entwicklungsumgebung für eine proprietäre Sprache

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Gunter Klemke
Zweitgutachter : Prof. Dr. rer. nat. Michael Schäfers

Abgegeben am 14. Mai 2008

Vladislav Neumüller

Thema der Bachelorarbeit

Analyse der Möglichkeiten zur Erstellung einer Entwicklungsumgebung für eine proprietäre Sprache

Stichworte

Mobile Geräte, J2ME, integrierte Entwicklungsumgebung, Rich Client Platform

Kurzzusammenfassung

Seit der Version 3.0 wurde von Eclipse die Rich Client Platform eingeführt. Heute ist diese Technologie zum einen neuen Standard geworden. Die Rich Client Platform soll die Entwicklung von komplexen Anwendungen ermöglichen. Welche Funktionalitäten die neue Plattform bietet, welche Alternativen existieren und wie effizient eine eigene Entwicklungsumgebung erstellen lässt, mit diesen Fragen beschäftigt sich diese Arbeit.

Vladislav Neumüller

Title of the paper

Analysis of the possibilities to create a development environment for a proprietary language

Keywords

Mobile devices, J2ME, integrated development environment, Rich Client Platform

Abstract

Since version 3.0 was Eclipse established Rich Client Platform. Today is this technology a new standard. The Rich Client Platform should make possible the development of complex applications. What features the new platform offers, what alternatives exist and how effectively a separate development environment can create, with these issues is busy this work.

1 EINLEITUNG 3

1.1 Problemstellung 4

1.2 Ablauf der Arbeit 4

2 SMOB-PLATTFORMARCHITEKTUR 6

3 GRUNDLAGEN 8

3.1 Java Platform, Micro Edition 8

3.1.1 Konfigurationen 9

3.1.2 Profile 11

3.2 .NET Compact Framework 15

3.3 J2ME Polish 18

3.4 SMOB 20

3.5 IDE 21

3.6 Java-Technologien 21

3.6.1 OSGi 21

3.6.2 Rich Client Platform 22

4 ANFORDERUNGSANALYSE 23

4.1 IDE 23

4.2 Texteditor 26

4.3 Debugger 29

4.4 Grafischer Editor 31

4.5 Projektverwaltung 33

4.6 Versionsverwaltung 34

4.7 CSS-Editor 35

5 ANALYSE 37

5.1 Plattformenvergleich 37

5.1.1 Installation 37

5.1.2 Architektur 39

5.1.3 Rich-Client-Platform 42

5.2 Konzept für die Entwicklung einer SMOB-IDE 46

5.3 RCP-Entwicklung 46

5.3.1 Texteditor 47

5.4 Schlussfolgerung 63

6 AUSBLICK 65

GLOSSAR 66

LITERATURVERZEICHNIS 69

ANHANG 1 71

ANHANG 2 78

ANHANG 3 82

ANHANG 4 95

ANHANG 5 98

ANHANG 6 100

ANHANG 7 102

1 Einleitung

Die mobilen Geräte sind heutzutage aus dem Leben der Menschen nicht wegzudenken. Sie sind zu den alltäglichen Begleitern geworden. Die Funktionsvielfalt reicht von reinem Telefonieren über die Benutzung der mobilen Geräte als multimediale Plattformen bis hin zur Nutzung als Internetbrowser und Navigationssystemen.

Mit der steigenden Leistungsfähigkeit der Geräte wachsen auch die Ansprüche der Benutzer. Sie erwarten mittlerweile von den mobilen Geräten die gleiche Funktionalität, die sie von den PCs gewohnt sind. Das Senden von Emails, das Verwalten von Adressen und Terminen, das Abspielen von Liedern und Videos, diese und viele weitere Funktionen sollen die mobilen Geräte ihren Besitzern bieten können. Um diese Bedürfnisse der Nutzer zu befriedigen, werden immer mehr Anwendungen speziell für die mobile Geräten entwickelt

Dieser Trend wurde auch von der Firma StarFinanz erkannt. Sie ist einer der führenden Anbieter im Bereich des Online-Banking. Als Produkte werden von der Firma für Privatkunden *StarMoney* und für die Geschäftskunden *StarMoney Business* angeboten. Für die mobilen Geräte wie Handys, Smartphones und PDAs wird der Kunde mit dem Produkt *StarMoney Mobile* beworben. Damit wird dem Kunden auch unterwegs die Möglichkeit gegeben, seine Bankkonten zu verwalten.

Die erste erschiene Version des Programms war StarMobile 1.0. Diese Anwendung basierte auf der WAP[1]-Technologie und ermöglichte dem Benutzer seine Bankgeschäfte unterwegs zu erledigen. Die aktuelle Version, *StarMoney Mobile 2.0*, ist ein Offline-Client und bietet dem Benutzer mehr Funktionen und Bedienungskomfort.

[1] WAP- Wireless Application Protocol

1.1 Problemstellung

Nach den gesammelten Erfahrungen im Bereich der Softwareerstellung für mobile Geräte wurde in der Firma nach neuen Geschäftsbereichen gesucht. Es entstand die Idee, eine Plattform zu entwickeln, mit der die Benutzer eigene Anwendungen für mobile Geräte jeglicher Art herstellen und nutzen könnten. Diese Plattform sollte Technologien, die auf dem Markt für mobile Geräte aktuell sind, unterstützen. Zurzeit sind das, Java Micro Edition von der Firma Sun Microsystems für die Handys und .Net Compact Network von der Microsoft Corporation für die PDAs. Ein wichtiger Teil dieser Plattform sollte eine selbstentwickelte Sprache sein. Diese Sprache sollte übersichtlich und einfach zu erlernen sein. Sie sollte alle nötigen Funktionen bieten, um vielseitige Anwendungen für mobile Geräte produzieren zu können. Diese Sprache wurde als *StarMoney Mobile Basic*, kurz SMOB, bezeichnet. Es wurde entschieden, dass zur Entwicklung der Anwendungen in dieser Sprache eine auf die SMOB-Sprache spezialisierte Entwicklungsumgebung von Vorteil ist. Diese Entwicklungsumgebung soll dem Prinzip der Rich Client Platform[2] Anwendungen entsprechen.

Gegenstand dieser Arbeit ist es, zu untersuchen mit welchen Technologien die Erstellung einer solchen Entwicklungsumgebung möglich ist. Dies schließt die Implementierung einer Prototyp-Anwendung, die den Anforderungen[3] entspricht, ein. Als Ergebnis soll präsentiert werden, mit welchen der existierenden Technologien, die erforderliche Entwicklungsumgebung sich am effizientesten erstellen lässt.

1.2 Ablauf der Arbeit

Die im Kapitel 2 beschriebene Plattform-Funktionalität umfasst mehrere Einsatzgebiete, Softwareentwicklung, Einsatz mobiler Geräte, Bereitstellung der Internetdienste[4]. In der Anfangsphase der Entstehung der vorgestellten Plattform gilt das Hauptaugenmerk der Sprache SMOB und der Entwicklungsumgebung, mit deren Hilfe Anwendungen programmiert werden sollen. Mit der Entwicklungsumgebung sollen die Programme erstellt werden, die in mobilen Geräten eingesetzt werden. Damit spielen mobile Geräte in dem gesamten Konzept eine zentrale Rolle. Die Technologien, die es möglich

---

[2] Vgl. Kapitel „Grundlagen“

[3] Vgl. Kapitel „Anforderungsanalyse“

[4] Webservices

machen, Programme für mobile Geräte zu entwickeln, was diese Technologien leisten, wie sie aufgebaut sind und welche Gerätarten dabei unterstützt werden, mit diesen Themen beschäftigt sich das Kapitel 3 dieser Arbeit. Zuerst wird die Basis-Technologie detailliert dargestellt, dann werden die in späteren Kapiteln benötigten Grundlagen, erklärt. Das Kapitel 4 definiert, welche Anforderungen an die Entwicklungsumgebung für die SMOB-Sprache gestellt werden. Schließlich werden im Kapitel 5 die Möglichkeiten der Erstellung einer solchen Entwicklungsumgebung analysiert.

2 SMOB-Plattformarchitektur

In diesem Kapitel wird die Architektur der SMOB-Plattform, sowie ihre sprachliche und funktionelle Komponenten, dargestellt.

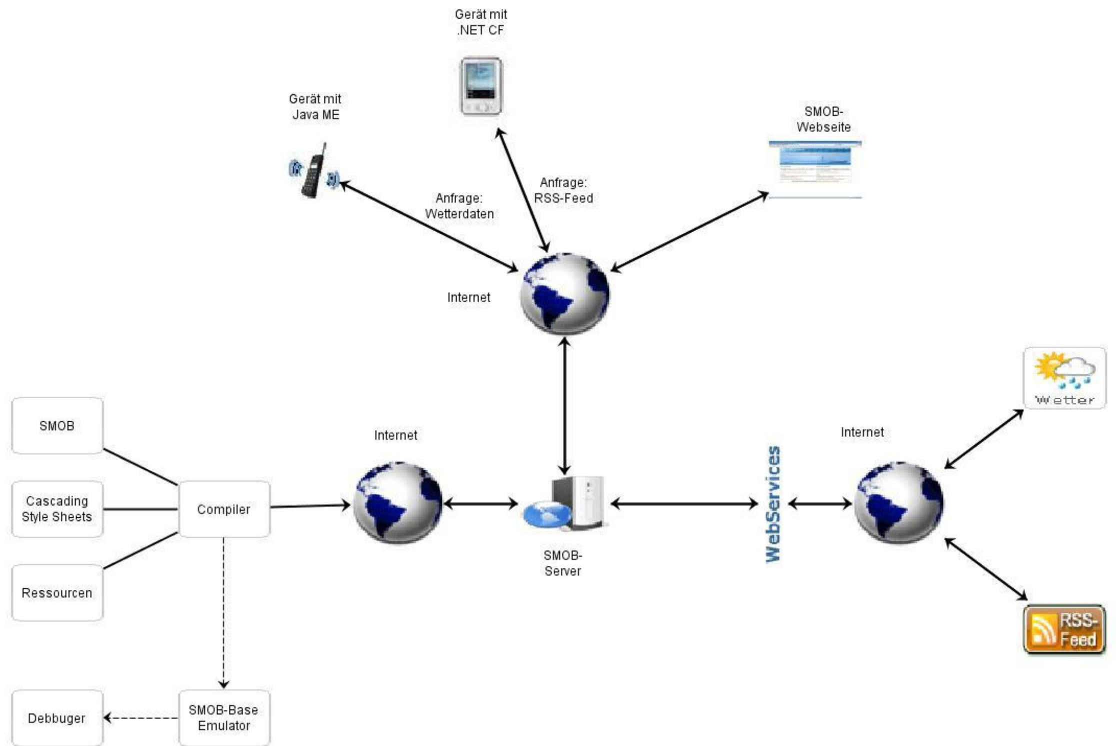


Abbildung 1: Systemarchitektur der SMOB-Sprache

Es wurde festgelegt, dass ein SMOB-Projekt aus drei Teilen bestehen wird, einer SMOB -Code-Datei, einer CSS[5]-Datei und der Ressourcen (Abbildung 1). Nachdem die Anwendung kompiliert wird, kann sie in einem Emulator getestet werden. Dieser Emulator enthält eine Datenbank mit den grafischen Oberflächen verschiedener mobile Geräte. Bei der Emulation werden das Aussehen und das Verhalten des mobilen Gerätes nah am Original dargestellt. So soll der Anwender die programmierten Funktionalitäten testen können. Um mögliche Fehler zu finden und zu korrigieren, kann der Entwickler das Programm durch den Debugger laufen lassen.

Nach dem die Anwendung fertig gestellt und in dem Emulator getestet wurde, hat der Entwickler die Möglichkeit das Programm auf den SMOB-Server hochzuladen. Dieser Server steht nur den registrierten Benutzern zur Verfügung. Die registrierten Benutzer können ihre fertigen Programme zu dem Server schicken und die auf dem Server

[5] CSS - Cascading Style Sheets

enthaltenen Programme, auf ihre mobilen Geräte herunterladen. Die Programme auf dem Server werden nach einer internen Überprüfung für alle Benutzer freigegeben. Nachdem der Benutzer eine Anwendung heruntergeladen und installiert hat, kann er diese ohne Weiteres auf seinem Gerät einsetzen.

Der SMOB-Server hat eine weitere Aufgabe. Die Anwendungen, die mit dem Internet kommunizieren, müssen sich zuerst mit dem SMOB-Server in Verbindung setzen. Über die Webservices verarbeitet der Server die Anfragen, holt die benötigten Informationen ab, bereitet die Daten vor und stellt sie dann den Anwendungen zur Verfügung.

Auf der SMOB-Webseite können die Benutzer nicht nur Programme für ihre Geräte auswählen und herunterladen, sondern auch die Einstellungen für die bereits installierten Programme speichern. Je nach der Einstellung werden verschiedene Daten auf dem Server für den Benutzer gesammelt, auf die er bei der Notwendigkeit sofort zugreifen kann.

3 Grundlagen

Die Anzahl der Nutzer mobiler Geräte wächst vom Jahr zu Jahr und die Geräte werden immer leistungsfähiger. Diese Tatsache lässt den Programmentwicklern die Möglichkeit offen, anspruchsvolle Anwendung zu entwickeln. Dabei stehen ihnen folgende Technologien zur Verfügung: J2ME, .NET Compact Framework, J2ME Polish.

3.1 Java Platform, Micro Edition

Java Platform, Micro Edition, wird seit Java-Version 6.0 offiziell als Java ME bezeichnet. Dies ist eine Umsetzung auf der Basis der Programmiersprache Java. Es dient als Plattform für das Entwickeln der Anwendungen, der sogenannten *„embedded consumer products*“, für solche Geräte wie Mobiltelefone und PDAs (Abbildung 2).

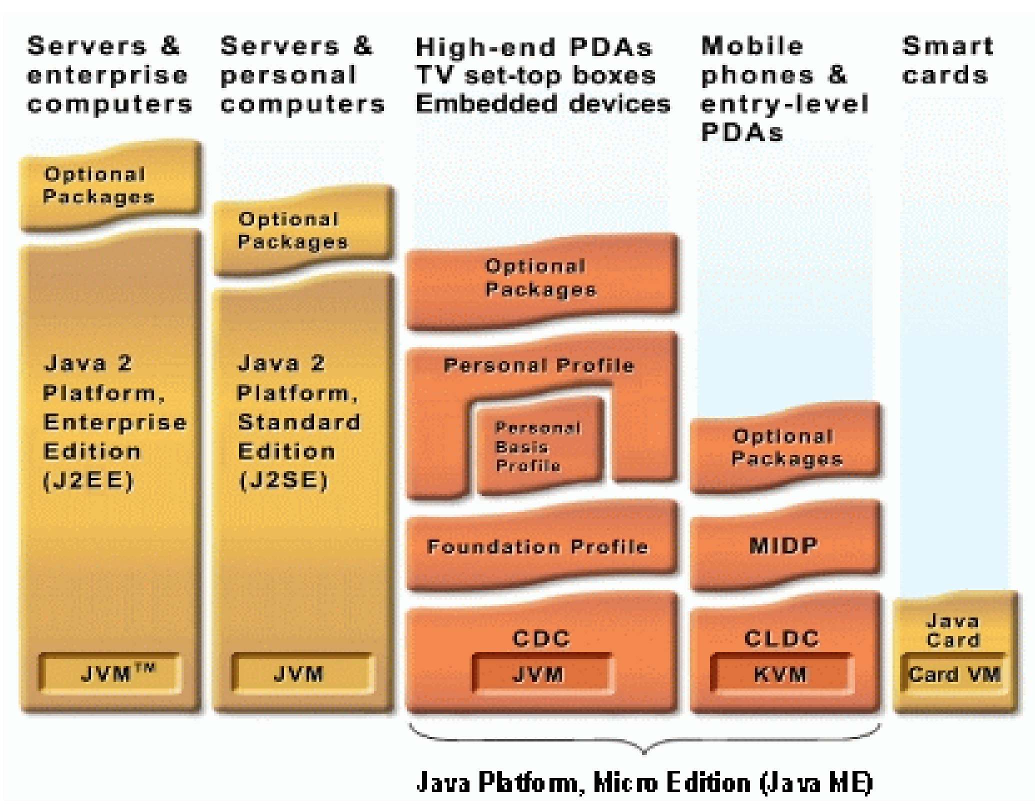


Abbildung 2: Unterteilung der Java 2 Technologie, Quelle [1]

Definiert wurde Java ME im Java Specification Request (JSR) 030 im Jahre 1999.[6] Ein *JSR* ist eine Forderung einer neuen Java-Spezifikation oder wichtigen Änderung einer

[6] Vgl. [2]

existierenden Java-Spezifikation. Diese Forderungen werden im Rahmen des Java Community Process (JCP) an das von Sun Microsystems betriebene *Process Management Office* (PMO) gestellt[7].

Speziell für die Geräte mit den eingeschränkten Ressourcen wurde die *Kilobyte Virtual Machine* (KVM) realisiert. KVM ist eine beschränkte Version der *Java Virtual Machine* (JVM) und braucht nur wenige Kilobyte Speicher um zu funktionieren.

Im Jahre 2003 wurde die Weiterentwicklung der KVM veröffentlicht. Sie trägt den Namen „*CLDC HotSpot™ Implementation Virtual Machine*" und soll schneller und robuster sein als ihre Vorgängerin.[8]

Die Konfigurationen und die Profile bilden die Grundlage von Java ME.

3.1.1 Konfigurationen

Im Allgemeinen beschreiben die Konfigurationen, die zur Verfügung stehenden Grundfunktionen der virtuellen Maschinen, für eine Gruppe von Geräten mit ähnlicher Leistungsfähigkeit. Damit wird die Entwicklung der Software für eine Gerätesparte vereinfacht. Als weitere Erweiterung der Konfigurationen dienen Profile, auf die später eingegangen wird.

Aktuell enthält Java ME zwei Konfiguration, Connected Limited Device Configuration (CLDC) und die Connected Device Configuration (CDC).

3.1.1.1 CLDC

CLDC wurde speziell für Mobiletelefone, PDA's und Pager entwickelt. Diese Konfiguration fordert, dass die Hardware folgende Voraussetzungen erfüllt. Eine 16 oder 32 Bit CPU, 128-512KB Speicher, Batterieversorgung und die Netzverbindung mit max. 9600 Baut/s.

Da diese Konfiguration für die Geräte mit weniger leistungsfähigen Prozessoren bedacht wurde, ist der Funktionsumfang in Vergleich zur J2SE wesentlich beschnitten (Abbildung 3). In der KVM wurden die Mechanismen zur Klassenzertifizierung nicht

---

[7]Vgl. [3]

[8] Vgl. [2]

integriert. Stattdessen werden die Klassendateien auf dem Rechner des Programmierers bei der Erstellung verifizieren. Das entlastet den KVM-*Classloader* bei der Prüfung der Klassen und beschleunigt die Ausführung des Programms.

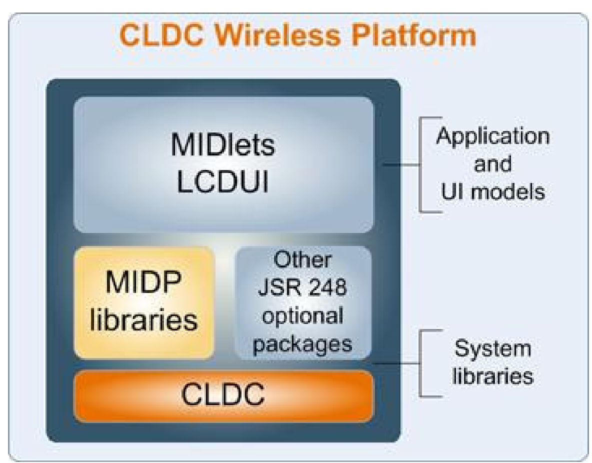


Abbildung 3: Aufbau der CLDC, Quelle [1]

Im Vergleich zu der Standard Edition kann der *Classloader* nicht überschrieben oder anders konfiguriert werden. Des Weiteren werden nur einzelne Threads und keine *Threads-Gruppen* oder *Daemon-Threads* unterstützt. Außerdem enthält KVM keine Funktion zur Objekt-Finalisierung durch den *Java Garbage Collector*. Ebenfalls sind die Möglichkeiten zur Fehlerbehandlung in der KVM limitiert. Auch Java Native Interface (JNI), das dem Entwickler den Zugriff auf Betriebssystemaufrufe erlaubt, wird von KVM nicht unterstützt[9].

3.1.1.2 CDC

Im Gegenteil zu CLDC wurde CDC für Geräte entwickelt, deren Leistung mit dem eines Pocket PCs vergleichbar ist. Dazu zählen auch die Settop-Boxen oder Fahrzeug Telematik Technik. Für diese Konfiguration werden ein 32-Bit Prozessor, mind. 2 MByte Arbeitsspeicher und die Netzverbindungsmöglichkeit, die mehr als 9600 Baut/s erlaubt, vorausgesetzt. Diese Eigenschaften erlauben im Gegensatz zu CLDC den Einsatz einer vollständigen Version Java VM.

[9] Vgl. [2]

3.1.2 Profile

Ein weiterer Bestand von Java ME sind die Profile, Sie basieren auf Konfigurationen und definieren die Programmierschnittstellen (API: *application programming interface*) für eine bestimmte Art von Geräten. Die Profile enthalten Klassen für die Benutzerschnittstelle, Netzverbindungen und persistente Datenspeicherung. Dazu können die Profile, abhängig von der Geräteklasse, für die sie spezifiziert wurden, auch weitere APIs enthalten, beispielweise für: Terminplaner, Adressbuch, Mitteilungen.

3.1.2.3 Mobile Information Device Profile (MIDP)

Bei der CLDC-Konfiguration ist es das Mobile Information Device Profile (MIDP), es wird für Mobiltelefone und PDAs definiert. Diese setzt voraus, dass die Geräte ein Bildschirm mit minimaler Größe von 96*54 Pixel und mindestens 1Bit Farbtiefe aufweisen. Außerdem werden 128 Kbyte flüchtiger und 264 KByte nicht flüchtiger Speicher und Audiounterstützung benötigt.

Java-Applikationen, die auf der Grundlage der MIDP entwickelt wurden, werden auch MIDlets genannt. Dabei kann es sich zum Beispiel um die in Java programmierte Handy-Spiele handeln. Die aktuelle Version 2.0 des MID Profils wurde erweitert, um die Bedürfnisse vor allem die der Spielentwickler zu befriedigen.

Neben der Standard-Klassen, die KVM von der Java SE geerbt hat, enthält MIDP weitere Klassenpakete, die sich in der Package *javax.microdetion* befinden[10].

Das sind:

- *Generetic Connection Framework*, stellt die Umgebung für die Netzwerkverbindungen bereit
- *Record Management System*, speicher die Daten dauerhaft in einer Datenbanken
- *LCDUI*, wird zur Entwicklung von grafischen Oberflachen benutzt.

Diese Klassen werden in folgendem Kapitel genauer beschrieben.

[10] Vgl.[4]

LCDUI

Bei der Beschreibung des *LCDUI,* der Benutzerschnittstelle bei MIDP, wird kein konkretes Layout festgelegt. Dies wurde entschieden, um die größere Portabilität zu erreichen, da die mobile Geräte sich bedeutend in der Ausstattung, in dem Format und in der Bildschirmgröße unterscheiden.

Die LCDUI wird in zwei Ebenen unterteilt, High- und Low-Level-API. High-Level-API abstrahiert die geräteigenen User Interface-Elemente, was der MIDlets höhere Plattformunabhängigkeit ermöglicht. Als Folge haben die Applikationen nur wenig Einfluss auf das Aussehen und Handhabung der Benutzeroberfläche. Das bedeutet, dass das grafische Aussehen der UI-Elementen, ihre Form, Farbe und ihr Zeichensatz, das Durchführen einfachster Interaktionen z.B.: Blättern eines Dialogs oder eine direkte Abfrage der Eingabegeräten, nicht von der Applikation, sondern von der LDCUI selbst bestimmt wird.

Bei der Mehrzahl der LCDUI-Anwendungen geben die Bedienelemente des High-Level-API die nativen UI-Komponenten des Gerätes wieder. Das bedeutet einerseits, dass der Benutzer auf die von ihm, aus nativen Applikationen, bekannte und gewohnte Aussehen und Verhalten trifft, andererseits kann ein und dasselbe MIDlet auf verschiedenen Geräten unterschiedlich aussehen.

Das Abstraktionsniveau des Low-Level-API ist im Vergleich zu High-Level-API sehr niedrig. Diese Schnittstelle bietet den Anwendungen eine größere Kontrolle über die Position und das Aussehen der UI-Elemente und die Möglichkeit die Benutzereingaben direkt zu empfangen. Bei der Benutzung des Low-Level-API können die MIDlets pixelgenau auf das Display zugreifen. Außerdem können die physische Tasten und andere Eingabegeräte nach Ereignissen abgefragt werden. Um das alles zu ermöglichen, macht das Low-Level-API die gerätespezifischen Eigenschaften, wie die Auflösung des Bildschirmes, Farbtiefe und die vorhandenen Tasten für die Anwendung sichtbar. Das führt jedoch dazu, dass die MIDlets, die die Low-Level-API nutzen, in ihrer Portabilität eingeschränkt werden.

Die LCDUI-Klassenbibliothek befindet sich im Paket *javax.microedition.lcdui*. Alle Dialoge werden durch die Klasse *Displayable* repräsentiert. Ein MIDlet kann gleichzeitig nur ein *Displayable*-Objekt darstellen. Viele mobile Geräte können mehrere

MIDlets zur gleichen Zeit ausführen. Das führt dazu, dass der Inhalt eines Displayable-Objektes nicht immer auf dem Bildschirm des Gerätes zu sehen ist.

Nach dem Start können die MIDlets mehrere Zustände annehmen und im Verlauf des Lebenszyklusses zwischen den Zuständen wechseln, wie die Abbildung 4 zeigt. Befindet sich ein MIDlet im Zustand „Active“, wird dieser auf dem Bildschirm angezeigt und erhält auch die Benutzereingaben. Wechselt es zum Zustand “Paused“, ist sein *Displayable*-Objekt nicht sichtbar und die Eingaben des Benutzers können nicht verarbeitet werden.

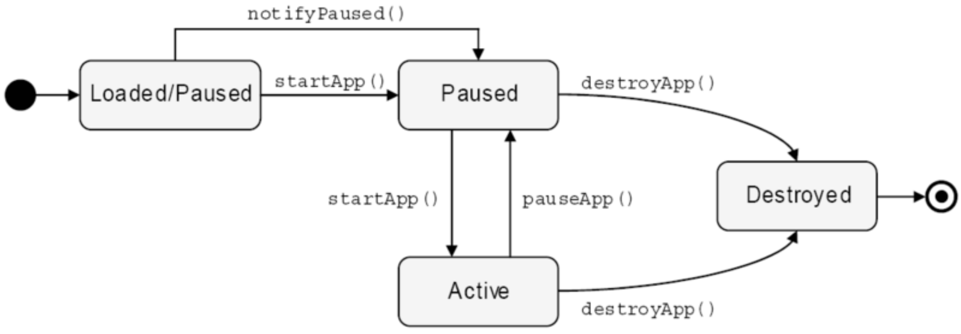


Abbildung 4: Lebenslauf des MIDLets

Die abstrakte Klasse *Displayable*, stellt eine Basisklasse dar und vereinigt die gemeinsamen Eigenschaften der zur Verfügung stehenden Dialogklassen (Abbildung 5). Jede Klasse kann folgende Eigenschaften haben: einen Titel, einen Ticker[11] und keine, eine oder mehrere Kommandos. *Displayable* stellt die Methoden zum Setzen und Abfragen dieser Eigenschaften bereit. Außerdem können bei einem *Displayable*-Objekt die Listener-Funktionen registriert werden. Diese werden automatisch von LCDUI abgefragt, sobald der Benutzer eine Aktion mit dem Kommando ausführt.

An der Spitze der Vererbungshierarchie befindet sich die *Displayable*-Klasse. Die Trennung auf Low- und High-Level-Komponenten findet in der ersten Vererbungsebene statt. Die Low-Level-Elemente erben von der Klasse *Canvas* und die High-Level-Elemente von der Klasse *Screen.*

[11] laufende Textzeile

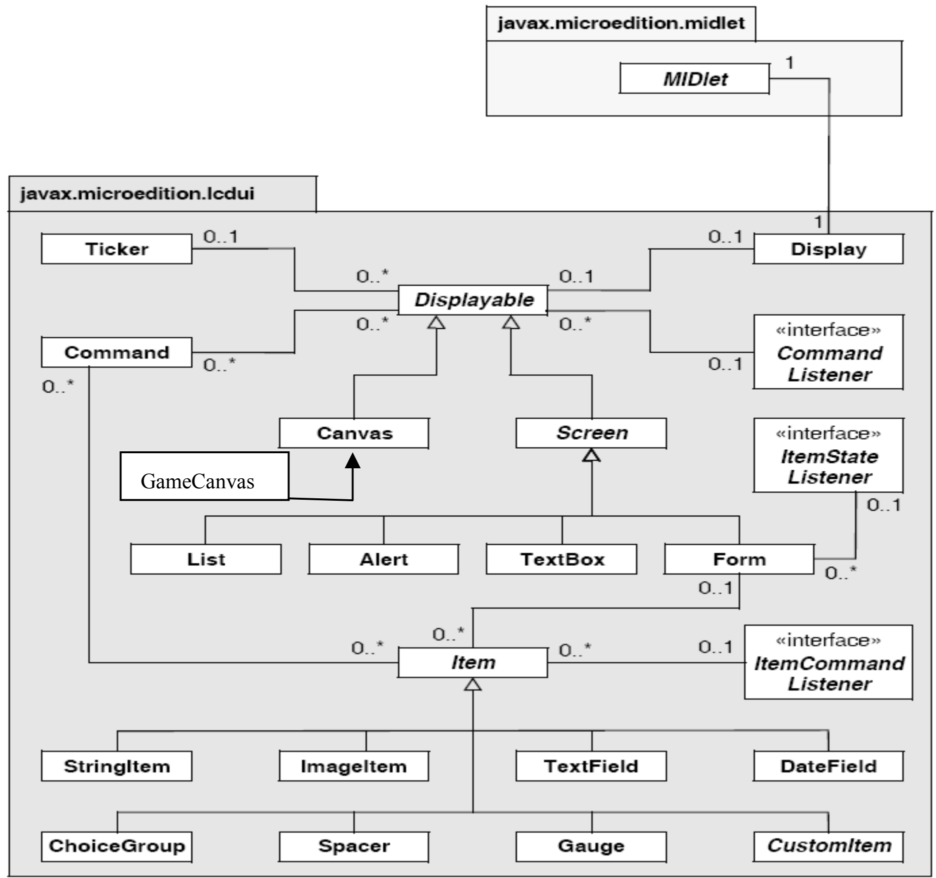


Abbildung 5: Klassendiagramm des LCDUI, Quelle [5]

Die Klasse *Screen* und ihre Unterklassen verarbeiten eigenständig Aufgaben die im Leistungsumfang von High-Level-Elementen enthalten sind. Sie zeichnen und aktualisieren automatisch die Elemente auf dem Display, blättern innerhalb eines Dialogs, interagieren mit dem Benutzer, deuten die Benutzereingaben und wandeln die wichtigen Eingaben in die entsprechenden Listener-Aufrufe um. Die Klassen, die von *Screen* erben, lassen sich in zwei Gruppen unterteilen:

- *List*, *Alert* und *TextBox* stellen die UI-Dialoge mit einer festen Struktur dar. Das vereinfacht die Anwendung dieser Klassen, schließt aber gleichzeitig die Möglichkeit aus, neue Strukturelemente in diese Komponenten einzufügen.

- *Form* dagegen, gestattet dem Entwickler Formulare aus vorhandenen *Item*-Elementen zusammenzustellen. Es ist flexibler, erfordert jedoch einen höheren Programmieraufwand.

Low-Level-API wird nur durch zwei Klassen repräsentiert, von der Klasse *Canvas* und seit MIDP 2.0 auch noch von der Klasse *GameCanvas*[12].

Generic Connection Framework

Dieses Framework sorgt für die Standardisierung der Netzwerkzugriffe. Es besteht aus einer einzigen Klasse und einer Menge von Schnittstellen für die Verbindungstypen.

Record Management System (RMS)

Aus Sicherheitsgründen und um die bestmögliche Plattformunabhängigkeit zu erreichen, ist es unter Java ME nicht möglich, die Daten innerhalb des Datensystems am Mobilgerät zu speichern. Um das trotzdem zu ermöglichen, stellt MIDP für die Speicherung der Daten eine Alternative bereit. RMS befindet sich im Paket *javax.microedition.rms* und ähnelt in ihrer Funktionsweise Java Database Connectivity (JDBC) in Java SE[13].

3.1.2.4 Information Module Profile (IMP)

Ein weiteres Profil ist das Information Module Profile (IMP), welches eine Untermenge der MIDP darstellt und für die *Machine-to-Machine*-Kommunikation gedacht ist. Als dessen Einsatzgebiet kann als Beispiel ein Getränkeautomat gesehen werden, welcher aufgefüllt werden soll und seinen aktuellen Zustand dann über dieses Profil entsprechend meldet.

3.2 .NET Compact Framework

Das *.NET Compact Framework 1.0* kam 2003 zusammen mit Microsoft Visual Studio 2003 auf den Markt. Unterstützt wird das Compact Framework ab *PocketPC2000, das auf WindowsCE 3.0* basiert und ab *WindowsCE.NET 4.1*. Als Programmiersprachen werden die .NET-Sprachen, C++, Visual Basic, C#, JScript, COBOL usw., verwendet.

Das .NET Compact Framework (CF) stellt eine Untermenge des .NET-Frameworks dar. Es ist speziell für die Nutzung auf mobilen Endgeräten wie beispielsweise Pocket PC, Smartphone und PDA unter Verwendung von Windows CE Betriebsystem, ausgerichtet. Das Ziel bei der Entwicklung des .NET Compact Framework war es, den

[12] Vgl. [5]

[13] Vgl. [4]

.NET-Entwicklern dabei zu helfen, Anwendungen für mobile Geräte zu schreiben oder sie auf diese zu portieren. Diese Version des .NET-Frameworks ist im Vergleich zu der Standardausführung um eine größere Anzahl von Klassen reduziert, die für die mobile Geräte nicht von Bedeutung sind oder zu viel Speicherplatz beanspruchen würden[14].

Technisch gesehen ist das Compact Framework eine Laufzeitumgebung, eine sogenannte *Common Language Runtime* (CLR) (Abbildung 6). Das bedeutet, CF dient als Mittelschicht zwischen dem Betriebssystem und den Anwendungen, die darauf laufen sollen. Eine Laufzeitumgebung lässt Programme plattformunabhängig ausführen. Ein bekanntes Beispiel dafür ist *Java Runtime Environment (JRE)*, eine Laufzeitumgebung für die Programme, die in Java geschrieben wurden.

Bei der Programmerstellung für mobile Geräte helfen die *Smart Device Extensions* (SDE). SDE stellen eine Erweiterung für Microsofts Entwicklungsumgebung Visual Studio dar. Sie umfassen folgende Komponente:

- Version von *.NET Compact Framework*, die auf normalen Windows-Versionen lauffähig ist
- Designer für CF-Form
- Emulator für Pocket PC.

Für die Ausführung des verwalteten Codes erbt das Compact-Framework die gesamte .NET Framework-Architektur von der CLR. Es bietet die Kompatibilität zu dem WindowsCE–Betriebssystem eines Gerätes, damit der Programmierer auf die systemeigenen Funktionen zugreifen und die benötigten Funktionen in seine Applikation integrieren kann. Des Weiteren kann der User die verwalteten und systemeigenen Anwendungen gleichzeitig ausführen. Die Instanz der CLR, die für die Ausführung des verwalteten Codes verantwortlich ist, wird von der systemeigenen Anwendung namens Anwendungsdomänenhost, gestartet.

Der allgemeine Aufbau der Plattformarchitektur wird in der Abbildung 6 deutlich. Im Folgenden wird auf die Besonderheiten der einzelnen Plattformteile eingegangen.

[14] Vgl. [6]

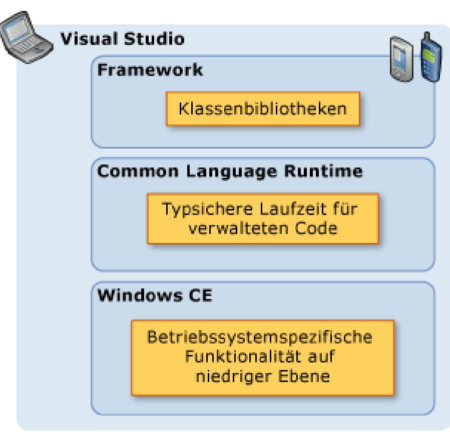


Abbildung 6: Plattformarchitektur von .NET Compact Framework, Quelle [6]

Windows CE

.NET Compact Framework nutzt die Hauptfunktionen und viele gerätespezifische Features des Windows CE-Betriebssystems. Für das neue Framework wurden mehrere Typen und Assemblys, beispielsweise für Windows Forms, Grafiken, Zeichenvorgänge und Webdienste, neu erstellt. In Folge dessen können sie effizient auf Geräten ausgeführt und müssen nicht aus dem vollständigen .NET Framework kopiert werden. In der Zusammenarbeit mit Windows CE bietet das .NET Compact Framework folgende Eigenschaften:

- Kompatibilität mit systemeigenen Sicherheitsmechanismen
- vollständige Implementierung mit der Hilfe von den betriebsystemeigenen Setupprogrammen
- Kooperation mit dem systemeigenen Code mit der Unterstützung von COM-Interop[15] und Plattformaufruf.

Common Language Runtime

Damit die Anwendungen auf den Geräten mit eingeschränkten Ressourcen, wie begrenzter Arbeitspeicher und kleine Akkukapazitäten lauffähig sind, wurde die Common Language Runtime (CLR) von .NET Compact Framework überarbeitet.

Um die von der CLR und von Framework benötigten Dienste, Windows CE-Diensten und -Schnittstellen zuordnen zu können, wurde zwischen Windows CE und der CLR

[15] **COM-Interop** - Component Object Model

eine Anpassungsschicht für Plattformen integriert. In der Abbildung 6 ist diese Schicht nicht dargestellt.

Framework

Das .NET Compact Framework enthält nur einen Teil der Funktionalität von .NET Framework, weist andererseits Features auf, die speziell für .NET Compact Framework entwickelt wurden. Die Features und das Bedienkonzept, die .NET CF zur Verfügung stellt, sollen den Entwicklern von systemeigenen Geräteapplikationen den Einstieg in die Entwicklung unter .NET Framework erleichtern. Zugleich sollen die Entwickler von Desktopapplikationen einfacher in die Entwicklung von Geräteanwendungen einsteigen können.

Visual Studio

Die Entwicklung von Anwendungen für mobile Geräte findet in Microsoft Visual Studio statt. Für diesen Zweck enthält Visual Studio eine Menge von Projekttypen und Emulatoren, mit deren Hilfe die Anwendungen für Pocket PC, Smartphone und Windows CE entwickelt werden.

Im Vergleich zum .NET-Framework enthält Compact Framework nur 28 Prozent dessen Funktionalität, was aber zur einen Größenreduzierung von 92 Prozent des normalen .NET-Frameworks führt. Um dies zu ermöglichen, werden nur die wichtigsten Klassen und Member unterstützt. Dazu enthält Compact Framework zusätzliche Funktionen und Klassen, speziell für mobile und eingebettete Geräte, beispielweise für IrDA[16] und SQL Server CE[17].

3.3 J2ME Polish

J2ME Polish ist eine Sammlung von Werkzeugen für die Programmierung von Java ME Anwendungen. Diese Software, die von Bremer Unternehmen *Enough Software* entwickelt wurde, ermöglicht eine automatische Optimierung der Programme für verschiedene mobile Geräte. Das soll bei der vorhandenen Vielzahl verschiedener Geräte auf dem Markt die plattformübergreifende Entwicklung von J2ME Anwendungen erleichtern.

[16] IrDA - Infrared Data Association

[17] Vgl. [6]

J2ME Polish ist eine Open-Source-Lösung. Sie enthält ein Build-Tool, das auf Ant basiert. Ant stellt ein Werkzeug dar, das automatisch Programme aus dem Quellcode erstellen kann. Des Weiteren enthält J2ME Polish eine integrierte Handy-Datenbank mit den technischen Eigenschaften einer Vielzahl mobiler Geräte. Es bietet auch eine grafische Benutzeroberfläche für Handy-Anwendungen, unterstützt MIDP 2.0 und kann mit der Hilfe von Cascading Style Sheets (CSS) konfiguriert werden. Auch ermöglicht J2ME Polish die Nutzung von MIDP2.0 auf MIDP1.0-Geräten. Damit kann auch die seit MIDP2.0 vorhandene Game-Engine für ältere Geräte genutzt werden. Außerdem stellt J2ME Polish ein Logging-Framework für J2ME bereit.

Die komplette Anwendungsentwicklung vom Preprocessing, über das Compiling, Preverifying und Packaging zu der JAD-Erstellung soll von J2ME-Polish abgedeckt werden. Die Tatsache, dass der Build-Prozess auf Ant basiert, ermöglicht die Nutzung von J2ME Polish in jeder Entwicklungsumgebung. Beim Preprocessing können die benötigten Gerätefunktionen aus der Datenbank abgefragt und genutzt werden. Dadurch können die Anwendungen an verschiedene Geräte angepasst werden, ohne das die Plattformunabhängigkeit verloren geht[18].

Mit der Hilfe von J2ME Polish können die Ressourcen einer Anwendung zugefügt werden. Das erleichtert die Lokalisierung einer Anwendung, da die Übersetzungen und andere lokale Ressourcen eingebunden werden können.

Mit dem Logging-Framework ist es möglich, „ System.out.println()“-Meldungen auf dem echten Endgerät einzusehen. Für spezielle Klassen oder Paketen können verschiedene Logging-Level wie "debug", "warn" oder "error" und oder auch eigene Level als „perfomance“ aktiviert oder deaktiviert werden.

Um den im Kapitel „LCDUI“ angesprochenen Nachteil des High-Level-API, die geräteabhängige Darstellung der Anwendungen auszugleichen, wurde in J2ME Polish eine Wrapper-API implementiert. Sie dient als Verbindungsglied zwischen Anwendung und Geräten-API und wird während des Build-Prozesses in die Anwendung implementiert und auf das Zielgerät angepasst. Damit können mögliche Fehler des Gerätes umgegangen und gerätespezifische APIs besser ausgenutzt werden.

---

[18] Vgl. [7]

Der Hersteller von J2ME Polish, Firma Enough Software vertreibt ihr Produkt sowohl unter GPL als auch unter kommerziellen Lizenzen[19].

3.4 SMOB

Die Sprache, *StarMoney Mobile Basic*, auch SMOB genannt, ist eine auf *Basic*[20] basierende Sprache, die sich bei der Firma StarFinanz GmbH in der Entwicklung befindet. Sie soll dem Programmentwickler ermöglichen, die Anwendungen für die mobilen Geräte verschiedener Art, einfach und ohne großen Aufwand zu erstellen.

Die Abbildung 7 stellt den Ablauf der Entwicklungsprozesse dar. Der Programmcode wird mit von der Entwicklungsumgebung syntaktisch analysiert und schließlich serialisiert. Als Ergebnis werden die AST- Dateien, auch Widgets genannt, erstellt. Die Widgets werden von der SMOB-Runtime, das auf dem jeweiligen mobilen Gerät läuft, interpretiert und zum Laufen gebracht. Die Serialisierung geschieht unter der Benutzung eines objektorientierten Parsergenerator, namens ANTLR[21].

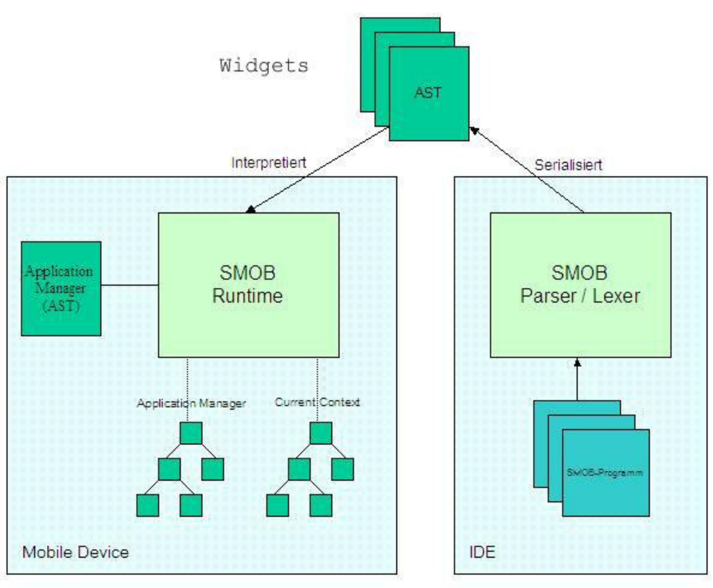


Abbildung 7: Architektur der SMOB-Sprache

---

[19] Vgl. [8]

[20] Vgl. [9]

[21] ANTLR - ANother Tool for Language Recognition

Der in Java geschriebene ANTLR ist als Open Source verfügbar und auf der Java-Plattform sowie auf .NET und Mono lauffähig. Er unterstützt C++, C#, Java und Python als Zielsprachen. Die von ANTLR erzeugten Code-Dateien benötigen Funktionen, die in einer Parser-Bibliothek[22] zur Verfügung gestellt werden. Dabei können abstrakte Syntaxbäume und entsprechende TreeParser automatisch erstellt werden. Die von ANTLR benutzte Sprache ist eine Mischung aus formaler Grammatik und Elementen aus objektorientierten Sprachen[23].

3.5 IDE

Der Hauptbestandteil der Softwareentwicklung ist die integrierte Entwicklungsumgebung. Eine integrierte Entwicklungsumgebung, vom englischen *integrated development environment* kurz IDE, ist ein Computerprogramm, das den Anwender bei der Softwareentwicklung unterstützt. Eine IDE enthält normalerweise folgende Komponenten: Texteditor, Compiler/Interpreter, Linker, Debugger und Quelltextformatierungsfunktion. Als Erweiterung können IDEs auch weitere Komponenten enthalten, unter anderem Versionsverwaltung, Projektmanagement, UML-Modellierung und die Unterstützung zur Erstellung grafischer Benutzeroberflächen.

3.6 Java-Technologien

In diesem Kapitel werden einige Begriffe aus Java-Umgebung erklärt, die in späteren Kapiteln von Bedeutung sein werden.

3.6.1 OSGi

Die *OSGi Alliance* ist ein Industriekonsortium. Es definiert eine hardwareunabhängige und dynamische Softwareplattform, auf der Anwendungen und Dienste auf Komponentenbasis ausgeführt und verwaltet werden können. Die OSGi-Plattform wurde in Java programmiert und setzt die Java Virtual Machine voraus. Darauf aufbauend bietet die OSGi-Plattform eine freie Serviceplattform an, das OSGi-Framework. Ein wichtiges Merkmal des Frameworks ist die Möglichkeit, dynamisch und kontrolliert Service-Anwendungen auch Bundles genannt, zur Laufzeit, je nach

[22] *antlr.runtime.dll*

[23] Vgl. [10]

Notwendigkeit, starten, aktualisieren und wieder schließen zu können. Damit bietet OSGi-Service-Plattform[24] eine Funktionalität an, die es möglich macht, nahezu unabhängige und modulare Anwendungen parallel in der gleichen Virtual Machine laufen zu lassen. Außerdem können diese Anwendungen während des gesamten Lebenszykluses der Applikation administriert, auch fern und aktualisiert werden. Zu der Verwaltung der Versionen und der Abhängigkeiten zwischen der Bundles wird ein intelligenter Mechanismus zur Verfügung gestellt. Diese Architektur reduziert deutlich die Komplexität bei dem Erstellen der Instandhaltung und dem Einsatz von Anwendungen.

3.6.2 Rich Client Platform

Der Begriff Rich Client kommt aus dem EDV-Bereich. In einer Client-Server-Architektur werden *Rich Client* und *Fat Client* jeweils als ein Programm bezeichnet, das die Aufgabe vor Ort erledigt und meistens eine grafische Oberfläche bietet. Im Gegenteil zum Fat Client kann der Rich Client nicht nur ein Problem lösen, sondern auch noch artverwandte oder gar artfremde Probleme lösen. Das resultiert sich aus der Tatsache, dass es sich bei Rich Clients um Frameworks handelt, die durch Module und Plugins erweiterbar sind.

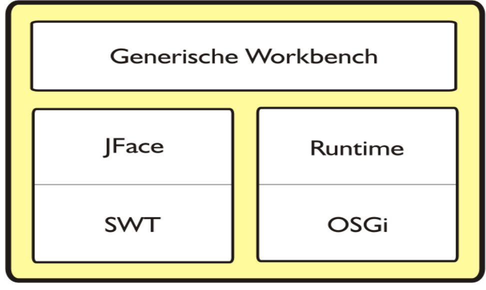


Abbildung 8: Eclipse Rich Client Platform

Die Bezeichnung *Rich Client Platform* wurde mit der Version 3.0 von Eclipse eingeführt und beschreibt ein generisches Framework für die Entwicklung von klientenzentrierten Anwendungen. In der neuen Plattform (Abbildung 8) wurde die Workbench-Funktionalität von der DIE (Runtime/OSGi)- und User Interface (JFace/SWT)-Funktionalität getrennt.

[24] früher "Open Services Gateway initiative", Vgl. [11]

4 Anforderungsanalyse

In dem Kapitel 2 wurde die Plattform-Architektur vorgestellt. Ein Teil dieser Architektur stellt die SMOB-Sprache dar. Im Laufe der Entwicklung wurde festgestellt, dass für diese neue Sprache eine Entwicklungsumgebung, IDE, notwendig ist. Die Entwicklungsumgebung sollte die Programmierarbeit und die spätere Fertigstellung der Anwendungen erleichtern. Im Gespräch zwischen den Entwicklern und der Geschäftsleitung wurden Anforderungen formuliert. Anhand dessen sollte entschieden werden, ob einer der zurzeit auf dem Markt vorhandenen Entwicklungsumgebungen dieser Anforderungen entspricht oder eine neue, auf die Sprache zugeschnittene IDE erforderlich ist. Die entstandenen Anforderungen werden in diesem Kapitel behandelt.

Bei der Anforderungsanalyse wurde zunächst eine grobe Übersicht der Anwendungsfälle angefertigt, dabei wurden einige Anwendungsfälle als „Sammlung“ gekennzeichnet. Die Sammlungen spalten sich bei näherer Betrachtung in untergeordnete Anwendungsfälle auf. Diese werden in ausgelagerten Diagrammen behandelt und näher spezifiziert. Eine Anwendungsfallsammlung besitzt aufgrund dieser Tatsache keinen konkreten Ablauf, da dieser erst in den einzelnen konkreten Anwendungsfällen spezifiziert werden kann. Die Beschreibungen sind als Referenz der Anwendungsfälle zu sehen.

4.1 IDE

Im Hauptanwendungsdiagramm in der Abbildung 9 werden alle Anwendungsfälle beschrieben. Die Sammlungen werden zuerst grob und dann später in den folgenden Kapiteln genauer beschrieben.

Texteditor

Ein Texteditor ist ein Computerprogramm, das dem Anwender die Arbeit mit den Texten auf dem Computer erleichtert. Seine Aufgabe ist es, die Texte, die bearbeitet werden sollen auf dem Bildschirm anzuzeigen.

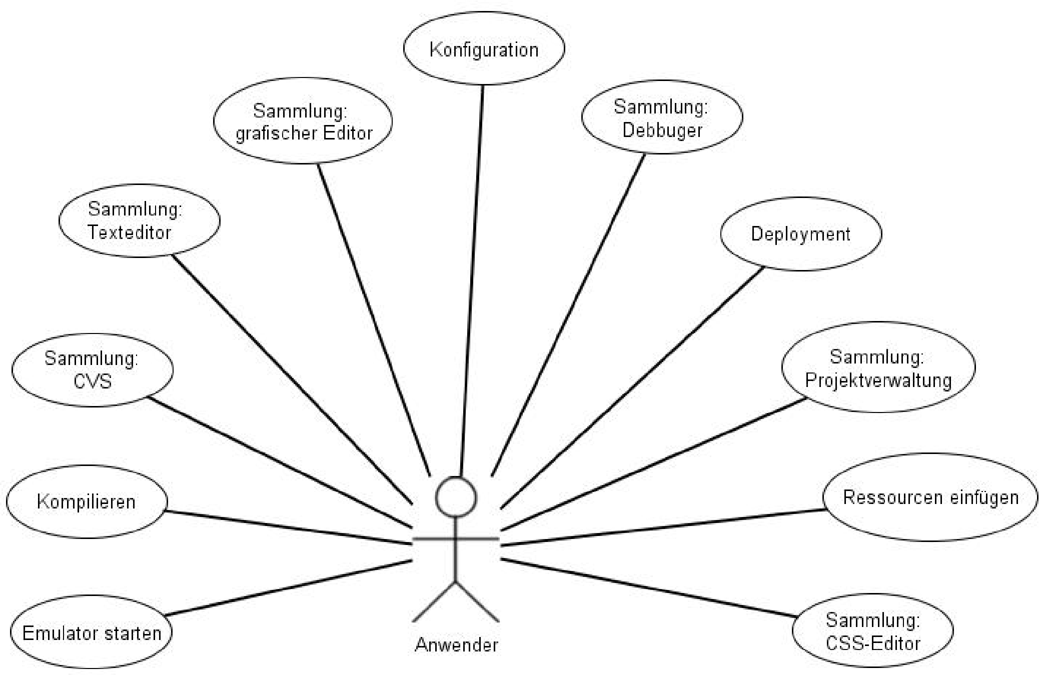


Abbildung 9: Anwendungsfalldiagramm: IDE

Zu seinem minimalen Funktionsumfang sollen solche Funktionen wie Kopieren, Ausschneiden, Einfügen, Suchen des Textes gehören. Die Syntaxhervorhebung (*syntax highlighting*) soll ebenfalls enthalten sein, es hebt den Code nach bestimmtem Muster hervor und ist von Sprache zu Sprache unterschiedlich. Auch die Code-Faltung (*code folding)* gehört dazu. Damit können bestimmte Teile des Codes ein oder ausgeblendet werden, was der besseren Übersichtlichkeit dient. Des Weiteren gehören zu dem Navigationskonzept, die Fenster für die Ausgabe, für die Fehlerbehebung, Outline-Kontrole und für den Projektmanager. Auch die Unterstützung der Tastenkombinationen wäre vom Vorteil.

Grafischer Editor

Ein grafischer Editor wird verwendet um dem Programmierer das Entwickeln von Benutzeroberflächen in der kurzen Zeit und mit weniger Aufwand zu ermöglichen.

Debugger

Bei der Entwicklung einer Software ist es meistens aufwändig einen vorhandenen Fehler zu finden. Der Debugger bittet an dieser Stelle eine Abhilfe. Er erlaubt dem Entwickler den Ablauf des Programms anzuhalten um die möglichen Fehlerquellen zu finden und zu beheben.

CSS-Editor

Das Design der grafischen Elemente auf der Programmbenutzeroberfläche wird mit der Hilfe einer CSS-Datei definiert. Diese Datei wird mit der Unterstützung von CSS-Editor verarbeitet. Der CSS-Editor wird als ein eigenständiges Programm in die IDE integriert.

Ressourcen einfügen

Beim Arbeiten mit IDE soll der Entwickler die Möglichkeit haben bestimmte Dateitypen dem Projekt hinzuzufügen. Dies können Bilder oder Tondateien sein. In welchem Format diese Dateien gespeichert werden dürfen, hangt davon ab, für welches Handymodell das aktuelle Projekt bestimmt ist. Bei Bildern werden es meistens JPEG-Format und bei Tondateien MIDI- oder MP3-Format sein.

Emulator starten

Während des Fertigstellungsprozesses soll dem Entwickler die Möglichkeit gegeben werden, die Funktionsweise oder das Design der zukünftigen Anwendung mit der Hilfe eines Simulators zu prüfen. Die IDE muss eine Datenbank verschiedener Handymodellen enthalten, deren Aussehen und Bedienungskonzept möglichst dem tatsächlich existierenden Handymodell ähneln.

Deployment

Bei der Arbeit mit der Entwicklungsumgebung soll dem Programmentwickler die Möglichkeit gegeben werden, die Kopie des fertigen Produktes auf den Firmenserver der StarFinanz GmbH zu übertragen. Die Anwendung wird verschlüsselt über eine gesicherte Leitung übertragen.

Projektverwaltung

Der Entwickler kann seine Projekte unter einer Oberfläche verwalten. Die Entwicklungsumgebung listet alle vorhandenen Projekte in einer Liste auf. Die Projekte können neu erstellt, geöffnet, geschlossen oder gelöscht werden.

Kompilieren

Der Anwender soll seinen Programmcode kompilieren können. Das bedeutet, dass die Entwicklungsumgebung einen integrierten Compiler zur Verfügung stellen soll. Mit seiner Hilfe wird der Programmcode in einen Maschinen- oder Bytecode umgewandelt.

CVS

Eine weitere Funktion die eine IDE bieten soll, ist die Möglichkeit das Projekt zu sichern. Es soll auch möglich gemacht werden das Projekt jeder Zeit wiederherzustellen oder den Zustand der aktuellen und der gesicherten Dateien zu vergleichen.

Konfiguration

Die IDE soll es erlauben alle notwendigen Einstellungen über ein Menü zu ändern und zu speichern. Unter anderen sollen die Einstellungen für die Auswahl der Sprache, für das Ändern der Farbe, der Größe und des Stils des Textes vorhanden sein.

4.2 Texteditor

Dieses Kapitel beschreibt die Anforderungen an einem Texteditor. Der Anwendungsfall wird in der Abbildung 10 dargestellt.

Syntaxhervorhebung

Die Syntaxhervorhebung bietet die Möglichkeit die Wörter und Zeichenkombinationen in einem Text, in verschiedenen Farben, Schriftarten und –stillen anzuzeigen. Die Darstellung des Textes wird übersichtlicher. Damit wird die Lesbarkeit des Textes verbessert und die Wahrscheinlichkeit von Tippfehlern verringert. Besonders hervorgehoben werden die Schlüsselwörter, Zahlen, Strings, Variablen- und Funktionsnamen. Die IDE bietet die Möglichkeit in dem Menü „Einstellungen„ eine SMOB-Sprachdatei auszuwählen. Das ermöglicht das dynamische Anpassen an den aktuellen Stand der Sprache.

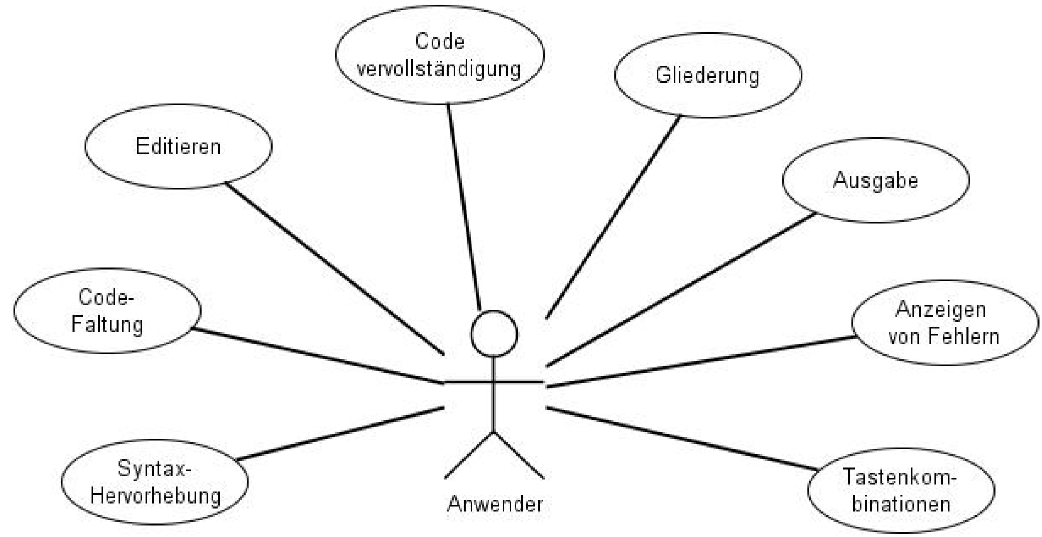


Abbildung 10 : Anwendungsfalldiagramm: Texteditor

Codefaltung

Dieser Anwendungsfall beschreibt die Möglichkeit des Texteditors bestimmte Abschnitte des Textes, die zusammen gehören, z.B. Funktionen und Kommentare so zu gestalten, dass sie je nach Wunsch des Entwicklers ein- oder ausgeblendet werden können. Das können Klassen, Methoden oder mehrzeilige Kommentare sein. Bei der Auswahl dieser Funktion in dem Menü „Einstellungen" werden die oben genannten Textabschnitte wie folgt markiert:

- Am Anfang der ersten Abschnittszeile erscheint ein „Minus"-Symbol.
- Beim Klicken auf das „Minus"-Symbol wird der Textabschnitt bis auf die erste Zeile ausgeblendet, gefaltet. Das Symbol wird als „Plus" dargestellt
- Beim Klicken auf das „Plus"-Symbol wird der ganze Textabschnitt wieder eingeblendet.

Die Darstellung des Symbols als „Minus" oder „Plus" dient nur als Beispiel. Möglich wären auch andere Darstellungen.

Editieren

Der Anwendungsfall „Editieren" stellt die allgemeinen Funktionen des Texteditors dar. Damit ist die einfachste Möglichkeit gemeint, Texte in einem Fenster zu schreiben, zu

löschen, auszuschneiden und einzufügen. Dazu können entweder das Kontextmenü oder die Standard-Windows-Tastenkombinationen benutzt werden.

Codevervollständigung

Der Texteditor soll dem Anwender das Kodieren erleichtern. Beim Drücken bestimmter Tastenkombination sollen dem Entwickler je nach vorhandenen Anfangsbuchstaben die möglichen Wortvarianten angeboten werden. Die Liste wird aus passenden Schlüsselwörtern, Funktionen- und Variablennamen zusammengestellt.

Gliederung

Der Anwendungsfall „Gliederung" beschreibt die Möglichkeit der IDE beim Öffnen oder beim Erstellen eines Programms, die vorhandenen Funktionen- und Variablennamen in einem Fenster in der Form zweier Listen, je für Funktionen und Variablen, anzuzeigen. Desweiteren wird ermöglicht, dass beim Klicken auf einen Funktionsnamen in der Liste, der ganze, zu der Funktion gehörende Textabschnitt markiert und angezeigt wird. Beim Klicken auf den Variablennamen wird die Variable an der Stelle ihrer Definition markiert und diese Stelle angezeigt. Außerdem kann der Entwickler die Einträge in der Liste entweder in der Reihenfolge ihres Auftretens oder nach Alphabet sortieren lassen.

Ausgabe

Die Ausgaben, die während des Programmablaufs im Debbugmodus erzeugt werden, werden in dem Ausgabefenster angezeigt. Diese Ausgaben werden durch spezielle Befehle der Sprache SMOB veranlasst.

Anzeigen von Fehlern

Der Texteditor enthält ein Fenster, in dem, die vom Compiler gefundenen Fehler angezeigt werden. Die Fehler werden in der Reihenfolge ihres Auftretens im Code und in der Form einer Liste dargestellt. Der Anwender kann die Stelle, die den Fehler enthält anzeigen lassen, in dem er den Eintrag in der Liste anklickt. Der behobene Fehler darf nach dem nächsten Compilerlauf nicht mehr in der Liste erscheinen.

Tastenkombinationen

Auch die Unterstützung der Tastenkombinationen, der sogenannten „Hotkeys“, gehört zur Standardausstattung eines Texteditors. Bei einer Tastenkombination werden mittels gleichzeitigen oder aufeinanderfolgenden Drückens mehrerer Tasten bestimmte Befehle angestoßen. Die Tastenkombinationen sollen abgesehen von solchen Standardbefehlen wie „Kopieren“, „Speichern“, „Ausschneiden“ oder „Einfügen“, frei einstellbar sein.

4.3 Debugger

Die einzelnen Anforderungen an den Debugger werden in diesem Kapitel definiert. Die Abbildung 11 zeigt das Anwendungsfalldiagramm für den Debugger.

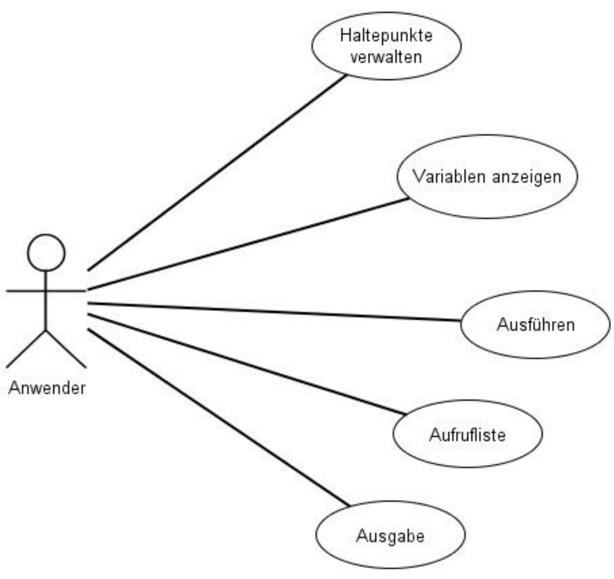


Abbildung 11: Anwendungsfalldiagramm: Debugger

Haltepunkte verwalten

Um das Programm während des Ablaufs anhalten zu können, muss der Entwickler die Haltepunkte setzen können. Das geschieht, in dem der Entwickler ein Doppelklick am Anfang der Zeile durchführt. Daraufhin erscheint am Anfang dieser Zeile eine Markierung. Nach dem Start des Programms stoppt der Debugger beim ersten Haltepunkt, den er trifft. Wenn ein Haltepunkt nicht mehr gebraucht wird, soll dem Entwickler die Möglichkeit gegeben werden, den Haltepunkt zu löschen.

Variablen anzeigen

Während des Debbugens soll es dem Anwender möglich gemacht werden, den Typ und den Inhalt der Variablen sich anzusehen. Der Anwender soll mit dem Klick auf eine Variable, per Kontextmenü auswählen können, ob die Daten sofort in einem neuen Fenster angezeigt oder die Variable in das Fenster, namens „Variablen" eingetragen werden soll. Im Fenster „Variablen" erscheinen die gewählten Variablen in einer Liste, wo nach Möglichkeit alle benötigten Informationen angezeigt werden. Bei Variablen, die mehrere Datensätze enthalten, sollen alle Datensätze angezeigt werden. Auch das Löschen der Variablen aus der Liste soll möglich gemacht werden.

Ausführen

Der Anwendungsfall „Ausführen" beschreibt den Ablauf des Debbugens. Nach dem Start wird die IDE in die Debuggerperspektive geschaltet und das Programm läuft bis zum ersten Haltepunkt. Dann wird das Programm gestoppt und die Codezeile mit dem Haltepunkt wird markiert dargestellt. In diesem Zustand ist es dem Entwickler möglich, die Variablen anzeigen zu lassen oder den Code zu verändern. Bei den Codeveränderungen wird das Debbugen unterbrochen. Die Veränderungen selbst werden erst nach dem Neustart gültig.

Aufrufliste

Das weitere Fenster in der Debuggerperspektive ist das Fenster „Aufrufliste". Darin wird während des Debbugens die Reihenfolge der Funktionsaufrufe in Form einer Liste dargestellt. Die Funktion, in der das Programm sich aktuell befindet, steht in der Liste ganz oben und ist markiert.

Ausgabe

Die Debuggerperspektive soll ein Ausgabefenster enthalten. In diesem Fenster werden die während des Programmablaufs im Debbugmodus erzeugten Ausgaben angezeigt. Das wird durch spezielle Befehle der Sprache SMOB veranlasst.

4.4 Grafischer Editor

Die Abbildung 12 stellt das Anwendungsfalldiagramm für den grafischen Editor dar. Im weiteren Verlauf des Kapitels werden die einzelnen Anwendungsfälle beschrieben.

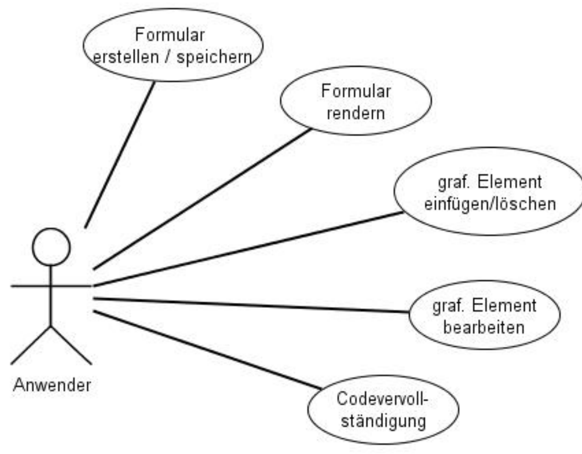


Abbildung 12: Anwendungsfalldiagramm: Grafischer Editor

Formular erstellen / speichern

Wenn der Entwickler ein neues Formular erstellt, dann wird eine Datei im Projekt erzeugt. Gleichzeitig öffnet sich automatisch die Ansicht für den grafischen Editor. Die Ansicht enthält vier Fenster. Diese sind: Ausgabe-, Werkzeug-, Eigenschaft- und das Hauptfenster. In dem Hauptfenster wird das zu bearbeitende Formular angezeigt. Im Anfangszustand ist das Formular leer.

Beim Bearbeiten werden die Änderungen im Code zwischengespeichert. Erst beim Speichern werden die Änderungen übernommen. Die alten Daten, sofern vorhanden, werden überschrieben. Wird die IDE geschlossen, ohne dass die Änderungen gespeichert wurden, wird der Programmcode in einer temporären Datei gesichert, die dann im Projektordner abgelegt wird.

Bei dem Neustart des Projektes soll der Anwender gefragt werden, ob die Datei mit dem nicht gespeicherten Programmcode wiederhergestellt werden soll. Lehnt der Anwender es ab, wird der zuletzt gespeicherte Codezustand angezeigt.

Formular rendern

Der grafische Editor stellt dem Anwender zwei Ansichtsmöglichkeiten zur Verfügung. Während der Anwender am Formular arbeitet, zeigt der Editor die Standardansicht. Diese Ansicht stellt das Formular in einem unveränderten Zustand dar. Das heißt, dass das Design des Formulars und aller in ihm enthaltenen Elementen keine Abweichungen von dem Original zeigt. Sind alle Eigenschaften festgelegt, soll der Benutzer die Möglichkeit haben das fertige Formular zu betrachten. Dazu bietet der Editor eine zweite Ansicht, in der das bereits gerenderte Formular angezeigt wird. In dieser Ansicht sieht das Formular so aus, wie es in der endgültigen Version aussehen würde.

Grafisches Element einfügen / löschen

Die grafischen Elemente werden per „Drag and Drop“ aus dem Werkzeugfenster auf das Formular gezogen. Die Platzierung des Elements soll frei wählbar sein.

Das Löschen des Elementes geschieht entweder über das Kontextmenü oder durch das Drücken der Entfernen-Taste. Bevor das Element gelöscht wird, soll der Anwender den Löschvorgang noch mal bestätigen, um das zufällige Entfernen zu vermeiden.

Grafisches Element bearbeiten

Die Größe des Elementes und seine Positionierung auf dem Formular soll der Anwender mit Hilfe einer Maus oder über die Eigenschaften verändern können. Die Eigenschaften erscheinen in dem Eigenschaftenfenster, sobald das Element markiert wird. Hier können, außer der Größe und der Positionierung weitere Eigenschaften wie der Name des Elementes, der anzuzeigende Text oder das allgemeine Design des Elementes, verändert werden. Das Design des Elementes wird durch die Eigenschaften geprägt, die in einer CSS-Datei gespeichert werden. Diese Datei kann in dem CSS-Editor angesehen oder bearbeitet werden. Die Funktionsweise es CSS-Editors wird in dem Anwendungsfall „CSS-Editor“ näher beschrieben.

Codevervollständigung

Bei dem Erstellen eines Formulars wird von dem grafischen Editor automatisch der entsprechende Code erzeugt. Jedes neu eingefügte Element, jede Änderung des Elementes oder des Formulars selbst, wird im Code festgehalten. Alle Änderungen des Codes werden zunächst zwischengespeichert und nur nach dem Speichern in die

Programmdatei übernommen. Um mit dem Code direkt arbeiten zu können, kann der Entwickler in die Texteditoransicht umschalten. Texteditor wurde im Anwendungsfall „Texteditor“ vorgestellt.

4.5 Projektverwaltung

In dem Diagramm in der Abbildung 13 sind Anwendungsfälle für Projektverwaltung dargestellt.

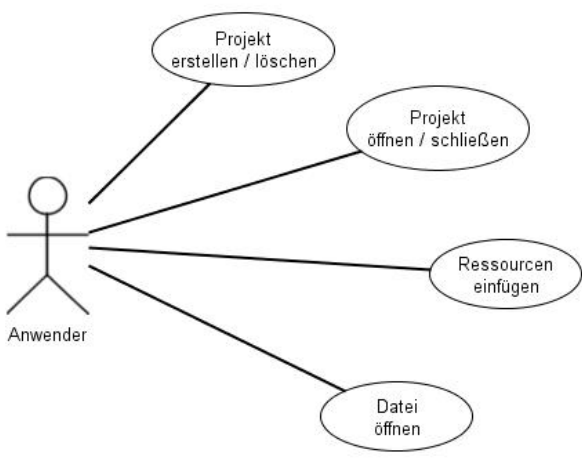


Abbildung 13: Anwendungsfalldiagramm: Projektverwaltung

Projekt erstellen / löschen

Bei der Erstellung eines neuen Projektes muss der Anwender den Namen und den Speicherort des Projektes angeben. Daraufhin wird ein Projekt angelegt. In dem Neuzustand enthält das Projekt nur die Projekt-Datei. Alle Dateien, die später erstellt oder dem Projekt hinzugefügt werden, werden automatisch im Projekt-Ordner gespeichert. Das neue Projekt erscheint in der Liste der Projekte im Projektfenster.

Beim Löschen des Projektes wird der Anwender gefragt, ob das Projekt nur aus der Liste oder komplett gelöscht werden soll. Nach der Bestätigung wird das Projekt aus der Liste der Projekte vollständig entfernt.

Projekt öffnen / schließen

Ein Projekt, das in der Liste enthalten ist, kann über das Kontextmenü geöffnet und wieder geschlossen werden. Nur die geöffneten Projekte können angesehen und bearbeitet werden.

Ressourcen einfügen

Desweiteren soll dem Anwender die Möglichkeit gegeben werden, weitere Dateien ins Projekt einzubinden. Das können Dateien mit den Bild- oder Toninformationen sein. Über Kontextmenü werden die gewünschten Dateien ausgewählt und dann ins Projekt-Verzeichnis kopiert. Daraufhin wird die Datei in der Liste des Projektes angezeigt.

Datei öffnen

Eine Datei kann per Enter-Taste oder Kontextmenü geöffnet werden. Die Programmcode-Dateien öffnen sich im Hauptfenster der Texteditoransicht. Die Ressourcen-Dateien werden mit den systemeigenen Programmen geöffnet.

4.6 Versionsverwaltung

Die Abbildung 14 stellt das Diagramm für den Anwendungsfall, Versionsverwaltung, dar.

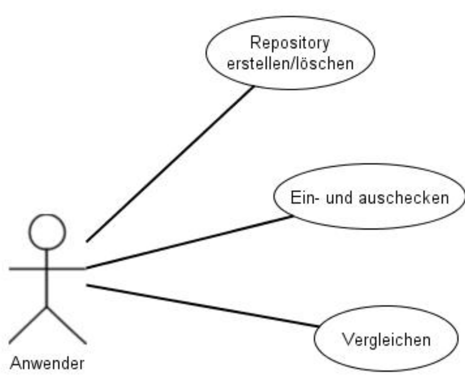


Abbildung 14: Anwendungsfalldiagramm: Versionsverwaltung

Repository erstellen/löschen

Die Entwicklungsumgebung soll dem Benutzer erlauben sein Projekt zu sichern. Jede wichtige Änderung des Projektes soll an einem Aufbewahrungsort, dem sogenannten Repository, gespeichert werden. Über das Kontextmenü soll der Anwender auswählen können, wo und unter welchem Namen die Repository erstellt werden soll. Dann wird das Projekt dort gespeichert. Wird die Repository nicht mehr gebraucht, kann der Benutzer sie über das Kontextmenü löschen.

Ein- und Auschecken

Die Versionsverwaltung soll die Möglichkeit bereitstellen, ganze Projekte oder einzelne Dateien eines Projektes zu sicheren, einchecken und wiederherzustellen, auschecken. Bei der Sicherung sollen nur die Unterschiede zur vorhandenen, bereits gesicherten Versionen gespeichert werden. Jede neue Version des Projektes soll mit einer eindeutigen Versionsnummer und einem Zeitstempel versehen werden. Unter Annahme, dass der Anwender als Einziger die Versionsverwaltung nutzt, soll auf die Sicherheitsmaßnahmen verzichtet werden. Das heißt, die aktuellen Daten werden ohne Überprüfung in die Repository gespeichert.

Im Falle, dass der Benutzer eine Wiederherstellung des Projektes wünscht, hat er eine Auswahl zwischen der Wiederherstellung des ganzen Projektes oder nur bestimmter Dateien. Bei der Wiederherstellung werden die aktuellen Dateien des Projektes überschrieben.

Vergleichen

Die Versionsverwaltung soll einen Vergleich zwischen den aktuellen und den gesicherten Dateien ermöglichen. Dabei werden die bestehenden Unterschiede angezeigt. Zur Erhöhung der Benutzerfreundlichkeit sollen die aktuellen und die gesicherten Dateien in einem Fenster parallel angezeigt werden. Der Benutzer soll zwischen den Versionen wählen können.

4.7 CSS-Editor

Das Aussehen der SMOB-Anwendung soll von dem Anwender über eine Formatvorlage-Sprache bestimmt werden. Die Entwickler der Firma StarFinanz haben sich auf die CSS-Sprache geeinigt. Folgend soll jedes Projekt eine CSS-Datei enthalten, wo die Daten über alle grafischen Elemente gespeichert sind. Deswegen soll die IDE neben dem Texteditor und dem grafischen Editor über eine weitere Ansicht für den CSS-Editor verfügen. Die zugehörige Ansicht umfasst zwei Fenster und zwar ein Hauptfenster und ein Gliederungsfenster. Die Abbildung 15 zeigt das Anwendungsfalldiagram für den CSS-Editor.

Syntaxhervorhebung

Der CSS-Editor stellt einen vereinfachten Texteditor dar. In dem Hauptfenster wird der in der CSS-Datei enthaltener Code, angezeigt. Der CSS-Editor soll über eine Syntaxhervorhebung verfügen, die auf XML-Sprache abgestimmt ist. Das soll den Anwender beim Lesen und Bearbeiten des Codes unterstützen.

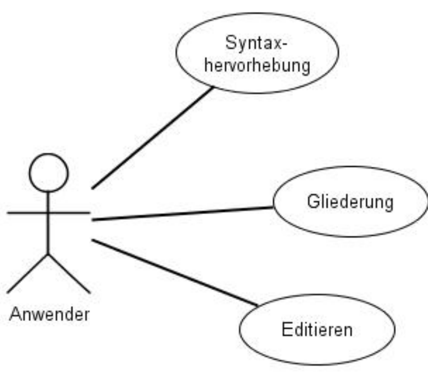


Abbildung 15: Anwendungsfalldiagramm: CSS-Editor

Gliederung

Das zweite Fenster in der Ansicht soll die Gliederung der Datei anzeigen. In diesem Fenster sollen alle vorhandenen Elemente in der Reihenfolge ihres Auftretens aufgelistet werden. Bei dem Klick auf ein Element, das sich in der Liste befindet, wird es im Hauptfenster markiert angezeigt.

Editieren

Das CSS-Dokument wird im Hauptfenster bearbeitet. Der Code soll editierbar sein. Es sollen die Standardfunktionen, wie Speichern, Kopieren, Ausschneiden und Einfügen zur Verfügung stehen. Das Ausführen dieser Funktionen soll auf drei Wegen möglich sein, über das Kontextmenü, über das Menü in der Menüleiste oder mit Hilfe der Tastenkombinationen.

5 Analyse

Die Laufzeitumgebungen ermöglichen den Benutzern die Programme auf verschiedenen Systemplattformen auszuführen. Die aktuell am meisten Verbreiteten sind die *JRE* und die *.NET CLR*. Bei dieser Analyse wird der Akzent auf RCP-Entwicklung gemacht, daher werden nur die IDEs betrachtet, die auf der Programmiersprache Java basieren. Die Entwicklungsumgebungen sollen frei verfügbar sein und eine große Benutzer-Akzeptanz besitzen. Die Entwicklungsumgebungen, die diesen Anforderungen entsprechen und das Erstellen von RCP-Anwendungen ermöglichen, sind Eclipse und NetBeans. In diesem Kapitel werden die beiden Entwicklungsumgebungen nach ihren RCP-Fähigkeiten, Gemeinsamkeiten, sowie ihren Unterschieden verglichen.

Bei dieser Analyse werden zuerst die Entwicklungsumgebungen, ihre Strukturaufbau und Funktionsweise vorgestellt, anschließend werden die Plattformen unter dem Aspekt von RCP-Entwicklung verglichen. Dabei werden die Möglichkeiten überprüft eine RCP-Anwendung zu entwickeln, die möglichst alle in dem Kapitel 3 vorgestellte Anforderungen, erfüllen.

5.1 Plattformenvergleich

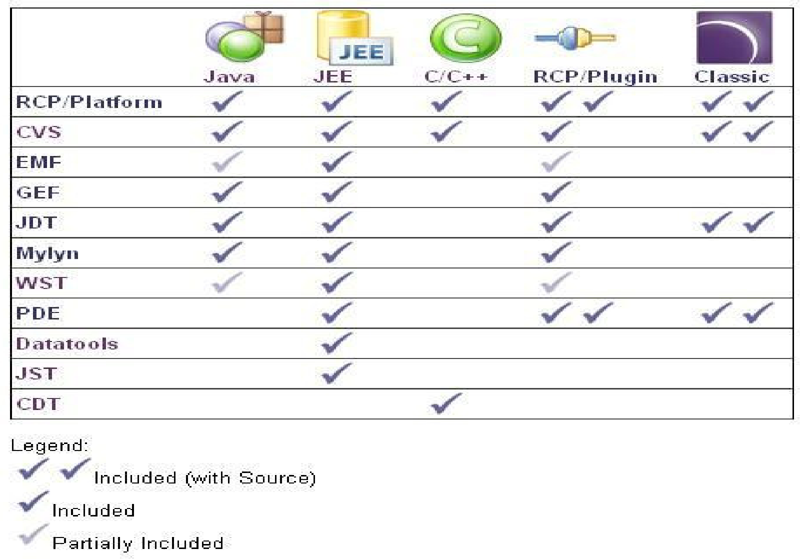
In diesem Kapitel werden die grundlegenden Strukturenunterschiede der beiden Plattformen betrachtet.

5.1.1 Installation

Beide Entwicklungsumgebungen werden kostenlos zur Verfügung gestellt. Sie werden im Rahmen des GNU-Projektes unter Open-Source-Lizenzen, *General Public License* (GPL) bei NetBeans und *Common Public License* (CPL) bei Eclipse vermarktet.

Eclipse

Auf der Webseite von Eclipse wird mittlerweile die Version 3.3 von Eclipse angeboten, auch Eclipse Europa genannt. Zur Auswahl stehen mehrere vordefinierte Pakete bereit. Für diese Analyse wurde das Paket „Eclipse for RCP/Plug-in Developers“ verwendet. Die Eclipse RCP-Version ist 154 MB groß und umfasst alle Funktionen, die zum Entwickeln einer RCP-Anwendung benötigt werden. Die im Paket enthaltenen Funktionen sind in der Abbildung 16 dargestellt.



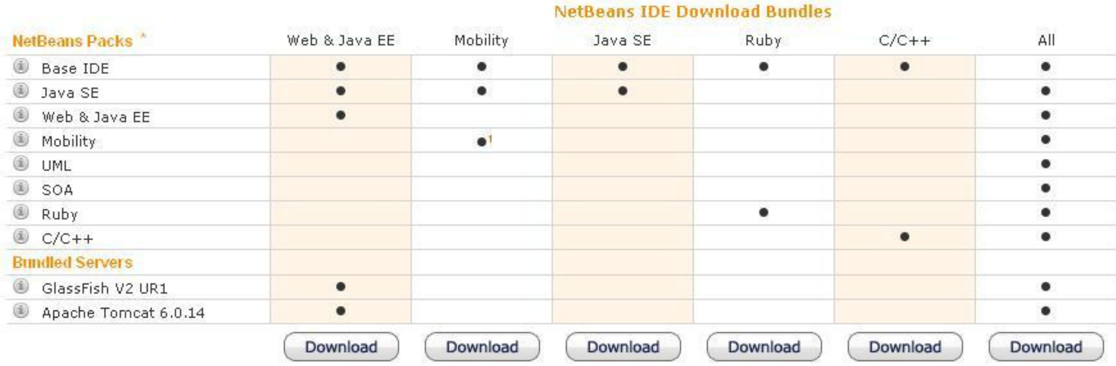
	Java	JEE	C/C++	RCP/Plugin	Classic
RCP/Platform	✓	✓	✓	✓✓	✓✓
CVS	✓	✓	✓	✓	✓✓
EMF	(✓)	✓		(✓)	
GEF	✓	✓		✓	
JDT	✓	✓		✓	✓✓
Mylyn	✓	✓		✓	
WST	(✓)	✓		(✓)	
PDE		✓		✓✓	✓✓
Datatools		✓			
JST		✓			
CDT			✓		

Legend:
✓✓ Included (with Source)
✓ Included
(✓) Partially Included

Abbildung. 16 Vergleich der Eclipse-Distibutionen. Quelle [12]

NetBeans

Auf der Webseite von NetBeans gibt es im Downloadbereich keinen Hinweis auf eine RCP-spezifische Variante. In dem Webseitenbereich, in dem die Plattformeigenschaften erläutert werden, wird jedoch auch erwähnt, dass für die RCP-Entwicklung die Standardversion ausreicht[25]. Anhand dieses Hinweises wurde für den Vergleich die „Java SE"-Version ausgewählt (Abbildung 17). Dieses Paket ist 49 MB groß und enthält außer der Grundfunktionen, den NetBeans GUI-Builder, auch Matisse genannt. Die heruntergeladene NetBeans Plattform hat die Versionsnummer 6.0.



	NetBeans IDE Download Bundles					
NetBeans Packs *	Web & Java EE	Mobility	Java SE	Ruby	C/C++	All
Base IDE	●	●	●	●	●	●
Java SE	●	●	●			●
Web & Java EE	●					●
Mobility		●1				●
UML						●
SOA						●
Ruby				●		●
C/C++					●	●
Bundled Servers						
GlassFish V2 UR1	●					●
Apache Tomcat 6.0.14	●					●
	Download	Download	Download	Download	Download	Download

Abbildung. 17 Vergleich der NetBeans-Distributionen, Quelle [13]

Nach der Installation kann der Benutzer mit beiden Entwicklungsumgebungen ohne weitere Vorbereitungen in die RCP-Applikationen-Entwicklung einsteigen. Allerdings

[25] Vgl. [13]

wenn bei Eclipse eine grafische Oberfläche erstellt werden soll, muss der Visual Editor(VE), der GUI-Builder von Eclipse, über die Eclipse Update-Funktion in System integriert werden.

5.1.2 Architektur

In diesem Kapitel wird der Architekturaufbau der beiden Plattformen genauer betrachtet. Zuerst wird die IDE von Eclipse betrachtet und anschließend die NetBeans IDE.

Eclipse

Bis einschließlich der Version 2.1 war Eclipse als eine erweiterbare IDE konzipiert. Ab Version 3.0 wurde Eclipse Plattform neu organisiert. Die Plattformelemente wurden von einander abgekoppelt. Die Eclipse Plattform wurde selbst zu einem Kern, von dem die einzelnen Komponenten der Plattform, *Plugins,* geladen und verwaltet werden. Die Plugins stellen dann die eigentliche Funktionalität des Systems zur Verfügung. Diese Funktionsweise ist charakteristisch für eine *Rich Client Platform* und basiert bei Eclipse auf der OSGi-Spezifikation.

Um diese Reorganisierung der Eclipse-Plattform zu ermöglichen, wurde Equinox entwickelt. Seit der Version 3.0 basiert Eclipse *Platform Runtime* (Abbildung 18) auf diesem Framework. Es ist für die flexible Registrierung und für die Versions- und Abhängigkeitenverwaltung von Plugins verantwortlich. Ausgehend von OSGi-Spezifikation wird jedes Eclipse Plugin auch als OSGi-Bundle bezeichnet.

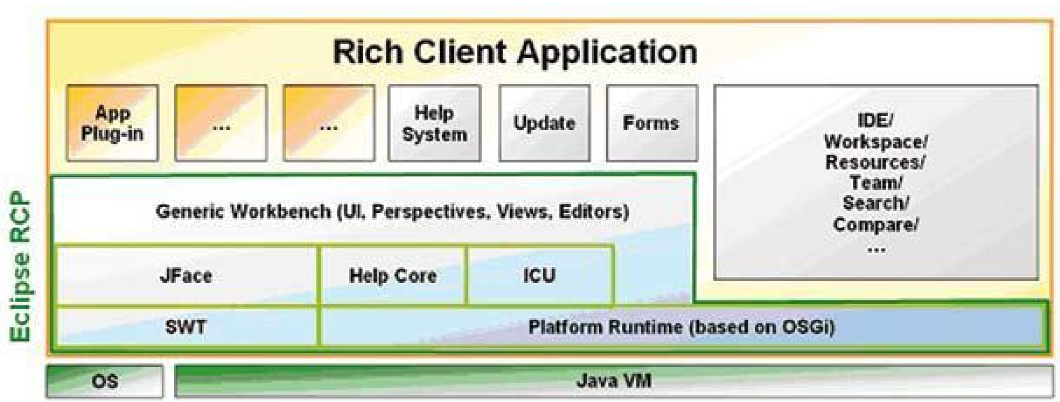


Abbildung 18: Die Architektur der Eclipse Rich Client Plattform, Quelle [15]

Obwohl Eclipse IDE in Java programmiert wurde, nutzt sie für das User-Interface ein eigenes Framework, *Standard Widget Toolkit* (SWT). SWT greift auf die nativen GUI-

Elemente des Betriebsystems zu. Eclipse ist daher nicht plattformunabhängig. Die entsprechenden SWT-Implementierungen werden aber für 14 verschiedene Systeme und Architekturen bereitgestellt. Durch die Nutzung von SWT soll den Anwendungen ein besseres „Look and Feel[26]" des jeweiligen Betriebsystems verleiht werden. SWT stellt eine Basis-Schicht, mit den grundlegenden Widgets und Utilities, dar.

Auf SWT basierend liegt eine weitere Schicht, System-Komponente, nämlich *JFace*. JFace bildet aus im SWT vorhandenen Komponenten komplexere Widgets zusammen. Es bietet eine Abstraktionsschicht (Viewer) für den Zugriff auf die Komponenten und die vielen Hilfsklassen. Unter anderem Viewer für Bäume und Tabellen, Editoren, Assistenten (Wizards) und Formulare.

Eine weitere Schicht ist die *Generic Workbench. Es* stellt ein Rahmenwerk für die weiteren GUI-Komponenten zur Verfügung, Views, Editors, Perspectives, Preferences, Actions (Tool- und Menu-Bars) und Window-Management.

Die weiteren Komponenten sind *Help Core* und *International Components for Unicode*(ICU). ICU ist verantwortlich für die Lokalisierung und Internationalisierung der RCP-Anwendungen[27].

NetBeans

Der Grundentwurf der NetBeans Plattform basiert auf den Modulen. Ein Modul ist eine Ansammlung von Klassen mit einer Definition, welche Schnittstellen dieses Modul anderen Modulen anbieten soll und welche Module es zur seiner Ausführung braucht. Die Plattform selbst und alle darauf basierende Anwendungen sind in die Module aufgeteilt. In ihrem Konzept und Funktionalität sind die NetBeans Module vergleichbar mit Plugins von Eclipse. Der Aufbau der NetBeans Plattform ist spezifisch für die *Rich Client Platform*[28].

Die Plattform basiert auf *Java Virtual Machine* (Abbildung. 19) und wird dabei komplett von dem Betriebssystem abgekoppelt. Der Grundbaustein der NetBeans-

---

[26] Look and Feel - dt. Aussehen und Handhabung

[27] Vgl. [15]

[28] Vgl [16] S.26

Architektur ist das Laufzeitsystem, *Platform Runtime*. Dieses System verwaltet die Module, sowie ihre Abhängigkeiten und ihren Lebenslaufzyklus. Es ist auch für die verschiedenen Registries und User-Interface-Schichten verantwortlich.

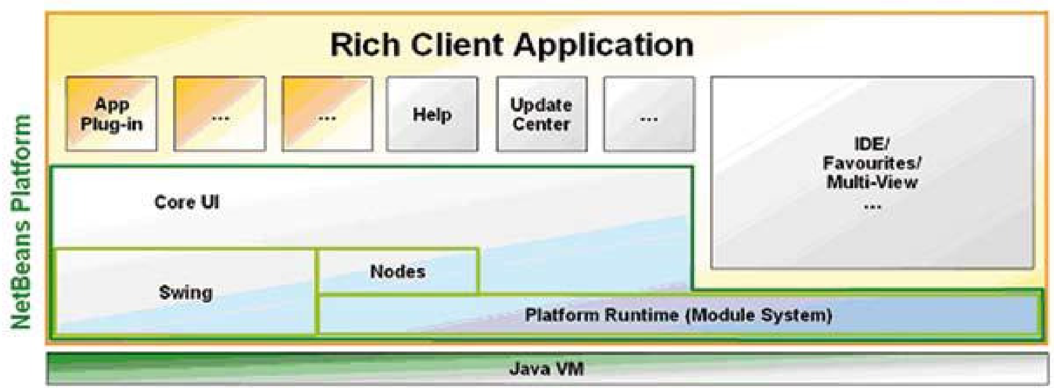


Abbildung 19: Die Architektur der NetBeans Plattform, Quelle [15]

Das User-Interface-System der NetBeans Plattform setzt auf *Swing*. Swing ist ein Teil der *Java Standard Edition* und ist daher direkt verfügbar. Die Swing-Komponenten werden direkt von Java gebildet und sind nicht von den nativen Betriebssystemkomponenten abhängig. Das macht die NetBeans Plattform selbst und die Anwendungen, die auf der Plattform basieren, betriebsystemtunabhängig. Die einzige Voraussetzung ist, dass das Betriebsystem mit einer zu Java 2 kompatiblen virtuellen Maschine, Version 1.3 und höher, ausgerüstet ist.

Ein weiteres wichtiges Element der NetBeans Plattform ist das Nodes-System. Es stellt ein Mechanismus zur Verfügung um die Geschäftsobjekte (*Business Object*)[29] von der Repräsentation im User-Interface zu lösen. Ein Node wird verwendet um die Daten an die grafische Benutzeroberfläche zu repräsentieren und dem Benutzer Aktionen, Funktionalitäten und Eigenschaften zur Verfügung zu stellen[30].

Core UI-Modul enthält u.a. Docking-System, Wizards und Komponenten. Das Docking-System stellt das Layout der unterschiedlichen Fenster zur Verfügung, die zu einer Anwendung gehören, z.B. die Editoren und die Views. Die Fenster lassen sich

[29] Geschäftsobjekte werden in OOP benutzt, um Objekte der geschäftlichen Welt innerhalb von Informationssystemen zu repräsentieren.

[30] Vgl. [15]

sowohl in ihrer Größe verändern als auch verschieben, stapeln und aus der Applikation herausziehen[31].

5.1.3 Rich-Client-Platform

Bei der Erstellung einer Rich-Client-Anwendung gehen die Plattformen unterschiedliche Wege. Um eine RCP-Anwendung effizienter entwickeln zu können ist es für den Programmierer hilfreich zu wissen wie das zukünftige Programm strukturiert ist. Das wird in diesem Kapitel näher erläutert.

5.1.3.1 Komponentenstruktur

In diesem Kapitel werden die Ansätze beider Plattformen hinsichtlich der Komponenten-Struktur einer RCP-Anwendung deutlich.

Eclipse

Die kleinste Einheit einer Eclipse RCP-basierten Anwendung ist Plugin. Plugins können verschiedene Funktionen einer Anwendung implementieren, z.B. Grundlogik (*Business Logic*) oder User-Interface-Funktionalität enthalten. Plugins mit einem ähnlichen Aufgabenbereich können zu größeren Funktionalitätsblöcken zusammengefasst werden. Eclipse enthält einen entsprechenden Mechanismus namens *Feature.* Ein bekanntes Beispiel dafür ist *Java Development Tooling* (JDT) von Eclipse IDE. Es umfasst alle Werkzeuge, die für die Java-Entwicklung benötigt werden, als ein Feature. Eine bedeutende Eigenschaft von Features ist, dass sie über einen Eclipse-internen Update-Mechanismus verfügen, mit dessen Hilfe diese auf den neuesten Stand gebracht werden können. Die einzelnen Plugins in Eclipse, die nicht zu einer Feature gehören, haben diese Möglichkeit nicht. Daher muss eine Eclipse RCP-Anwendung mindestens eine Feature enthalten, wenn ihre Anforderungen eine Update-Funktion voraussetzen.

Desweiteren stellt Eclipse eine *Branding*-Funktion zur Verfügung. Sie ermöglicht dem Entwickler das Aussehen des Programms individuell zu gestalten. Der Entwickler kann den Startbildschirm (*Splash Screen*), Icons und Namen für den Programmstarter (*Launcher*), Icons für das Programm-Fenster und Bilder und Texte für den *About*-

[31] Vgl. [15]

Dialog[32], bestimmen. Die Branding-Funktion in Eclipse unterstützt sowohl die ganze Anwendung (*Product Branding*), als auch die einzelnen Features (*Feature Branding*).

NetBeans

In der Komponenten-Struktur einer Anwendung, die auf NetBeans Plattform basiert, stellt *NetBeans Module* die kleinste Einheit dar. NetBeans Module ist äquivalent zu dem Eclipse Plugin, kann aber im Gegensatz dazu, über eigene Update-Funktion separat aktualisiert werden. Damit der Benutzer programmierte Module einfacher ausführen, testen, Abhängigkeiten und Bibliotheken definieren kann und später alle Module als komplette und selbstständige Anwendung erstellen kann, bietet NetBeans Plattform *Module Suite*. Eine Module Suite ist ein Container für die Module, der auch die Produktkonfiguration bereitstellt. Eine NetBeans Anwendung besitzt nur eine Module Suite. Die Branding-Funktion bietet die Plattform nur für die Anwendung als Ganzes an. Es werden Splash Screen, die Applikations-Bezeichnung und das Fenster Icon unterstützt. Als weiteres Feature bietet NetBeans IDE einen visuellen Splash Screen Editor.

5.1.3.2 Projektaufbau

Beide Plattformen gehen bei der Erstellung von Software-Projekten unterschiedlich vor. Um eine RCP-Anwendung effizienter entwickeln zu können, ist es für den Programmierer hilfreich zu wissen wie das Software-Projekt strukturiert ist. Das wird in diesem Kapitel ausführlich dargestellt.

NetBeans

NetBeans Plattform baut ihre Projekte mit der Hilfe von Ant. Ant arbeitet auf ähnliche Weise wie *make*, ein in C- und C++-Entwicklerkreisen bekanntes Programm. Im Gegenteil zu *make* wurde Ant in Java entwickelt und setzt eine Java-Laufzeitumgebung(JRE) voraus. Ant wird von einer XML-Datei gesteuert, die standardgemäß *build.xml* heißt (Abbildung 20).

[32] About-Dialog - ein Dialogfenster, das dem Entwickler einer Software sowie der Software selbst gewidmet ist.

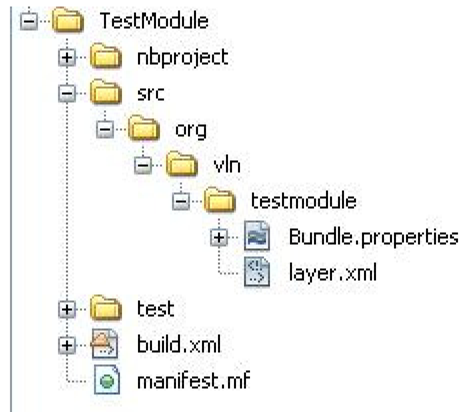


Abbildung 20: NetBeans Module-Architektur

In dieser Datei wird ein Projekt definiert, das zugleich auch ein Wurzelelement ist. Ein Softwareprojekt darf nur eine Build-Datei haben, was bedeutet, dass es nur ein Ant-Projekt enthalten kann. Als Softwareprojekte sind NetBeans Module Suite, sowie NetBeans Module zu verstehen.

Ein Modul besteht, in allgemeiner Form, aus folgenden Teilen:

- Manifest-Datei (manifest.mf)
- Layer-Datei (layer.xml)
- Anwendungsklassen
- Ressourcen (Icons, Properties Bundles, Helpsets etc.).

Von den oben genannten Teilen ist nur die Manifest-Datei erforderlich, weil durch diese das Modul identifiziert wird. Beim Laden eines Moduls wird die Manifest-Datei als Erstes ausgelesen. Sie enthält die Beschreibung des Moduls, seine Schnittstellen und Umgebung sowie die vorhanden Abhängigkeiten von anderen Modulen. Das Vorhandensein weiterer Bestandteile wird durch die Aufgabe des Moduls bestimmt. Soll das Modul z.B. eine Bibliothek darstellen, ist die Layer-Datei nicht erforderlich.

Die Layer-Datei ist eine zentrale Konfigurationsdatei eines Moduls. In dieser Datei wird die ganze Funktionalität, die ein Modul der Plattform hinzufügen möchte, definiert. Die

Layer-Datei stellt die Schnittstelle eines Moduls zu der Plattform dar und dokumentiert die Integration des Moduls in die Plattform[33].

Eclipse

Eclipse benutzt kein Ant, sondern verwendet einen eigenen Compiler. Die erzeugten Klassen werden direkt in einem Ausgangsverzeichnis erfasst.

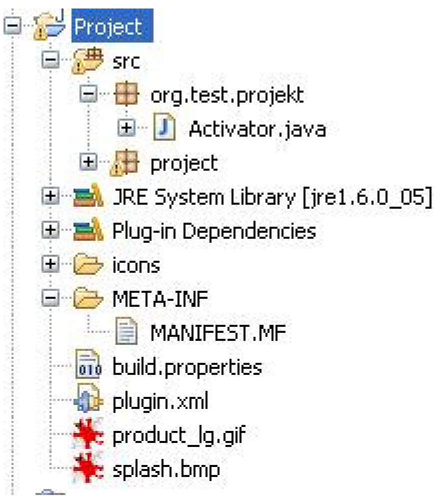


Abbildung 21: Eclipse Plugin-Architektur

Der Eclipse Plugin-Aufbau ähnelt dem von NetBeans Modul:

- OSGi-Manifest-Datei (MANIFEST.MF)
- Plugin-Manifest-Datei (plugin.xml)
- Anwendungsklassen
- Ressourcen.

Es wird eine OSGi-Manifest-Datei vorausgesetzt. In dieser Datei werden die Eigenschaften des Plugins selbst sowie seine Abhängigkeiten von anderen Plugins beschrieben. Darin wird auch festgesetzt, welche Packages des Plugins für andere Plugins sichtbar sein sollen. Plugin-Manifest-Datei wird benötigt, wenn Plugin nicht nur eine deklarative Funktion haben soll, sondern auch aktive Programme enthält. In dieser *plugin.xml*-Datei (Abbildung 21) wird definiert, über welche Erweiterungspunkte das

[33] Vgl. [16] S.44

Plugin seine Funktionalität der Plattform und den anderen Plugins bereitstellt und an welchen Erweiterungspunkten das Plugin seinerseits erweitert werden kann.

5.2 Konzept für die Entwicklung einer SMOB-IDE

Die Entwicklungsumgebung für die SMOB-Sprache soll folgende Funktionalität vorweisen:

- modularer Aufbau
- Erweiterbarkeit durch weitere Module/Plugins
- Aktualisierbarkeit.

Der Kern der IDE soll den minimalen Funktionsumfang anbieten um die IDE-Größe klein zu halten. Die Funktionen, die in den Anforderungen beschrieben wurden, gehören zu dem Funktionsumfang der SMOB-DIE. Sie sollen gleich nach der Installation dem Benutzer bereitgestellt werden. Alle weiteren Funktionalitäten, die außerhalb der Anforderungen liegen, sollen als Plugins, bei Notwendigkeit, angebunden werden.

Wie der genaue IDE-Aufbau aussehen soll, ob es eine Plugin-Struktur nach OSGi-Standard wie bei Eclipse haben wird oder einen modularen Aufbau wie bei NetBeans, hängt davon ab, welchen der beiden Plattformen den Vorzug gegeben wird. Diese Arbeit soll bei der Entscheidung helfen.

Wegen des begrenzten Zeitrahmens einer Diplomarbeit konnte nur ein Teil der Anforderungen praktisch implementiert werden. Bei beiden Plattformen wurde ein textueller Editor, nach den folgenden Anforderungen erstellt:

- Editieren (NetBeans/Eclipse)
- Syntaxhervorhebung (NetBeans/Eclipse)
- Codevervollständigung (Eclipse)
- Gliederung. (Eclipse).

5.3 RCP-Entwicklung

In diesem Kapitel werden die Entwicklungsumgebungen bei der Erstellung einer RCP-Anwendung nach, in Kapitel 4 gestellten Anforderungen, geprüft.

5.3.1 Texteditor

Der Texteditor ist einer der wichtigsten Bestandteilen einer Entwicklungsumgebung. Von seinen Eigenschaften hängt es ab, wie schnell, qualitativ und entwicklerfreundlich eine Applikation erstellt wird. Dieses Kapitel stellt die Möglichkeiten beider Plattformen in der Erstellung eines Texteditors.

5.3.1.1 Eclipse

Um eine funktionsfähige Grundbasis für einen Editor zu schaffen, muss zuerst ein Plugin-Projekt als eine Rich Client Application erstellt werden. In diesem Beispiel wurde eine RCP Anwendung-Vorlage mit bereits implementierter Sicht (View) ausgewählt (Abbildung 22).

In dem Eclipse Plugin wird eine Klasse angelegt sein, die von dem *TextEditor* ableitet wird. Der *TextEditor* wird seinerseits von dem *AbstractTextEditor* abgeleitet und stellt eine Basisimplementierung eines Texteditors im Eclipse Framework dar. Der so erstellte Texteditor verfügt damit über eine Sammlung von Standardfunktionalitäten zur Textbearbeitung sowie Kopier-, Ausschneide- und Einfügeoperationen oder Speichern von Änderungen.

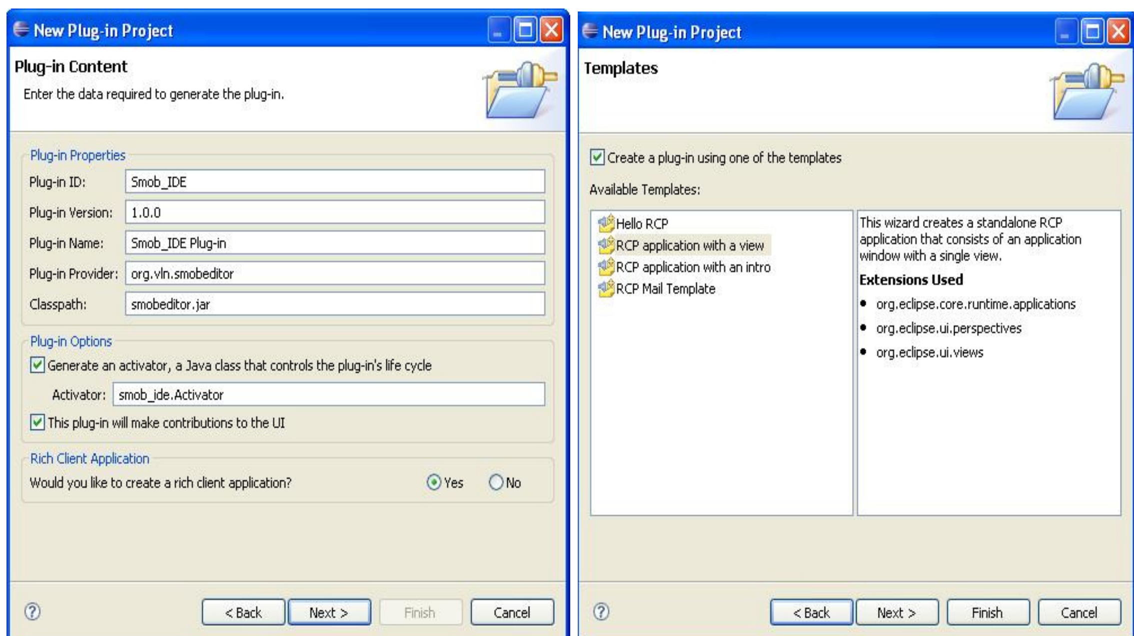


Abbildung 22: Zwei Schritte zur Erstellung einer Eclipse RCP-Anwendung

Er muss in der Manifest-Datei, *plugin.xml,* deklariert sein (Listing 1). Außerdem können hier Icons definiert werden, für die mit dem Editor assoziierten Ressourcen und

Workbech-Windows. Im Attribut *extensions* werden Datentypen festgelegt, für die der Editor verwendet werden soll.

Im Attribut *contributorClass* wird die *Contibutor*-Klasse festgelegt. Sie hat die Aufgabe, die Ereignisse von der Benutzerschnittstelle (Menü, Symbolleiste, Tastaturkürzel) an den aktiven Editor zu leiten.

```
<extension
      point="org.eclipse.ui.editors">
   <editor
         class="com.starfinanz.smob.ide.editor.SmobEditor"
         contributorClass="org.eclipse.ui.texteditor.
                                    BasicTextEditorActionContributor"
         extensions="bas"
         icon="icons/sample.gif"
         id="com.starfinanz.smob.ide.editors.SmobEditor"
         name="Smob-Editor"/>
</extension>
```

Listing 1: Deklaration eines Editors(Ausschnitt aus der plugin.xml)

Wenn eine Ressource geöffnet wird, arbeitet der Editor nicht direkt darauf, sondern auf einem ***Dokument,*** welches das Interface *IDocument* implementieren muss. Die Beschaffenheit jedes Dokumentes hängt von dem jeweiligen Datentyp ab. Die Workbench kann dem Editor nur dann ein Dokument zu Verfügung stellen, wenn diese eine entsprechende Implementierung eines *DocumentProvider,* das Interface *IDocumentProvider* implementiert, enthält. Der Editor muss einen DocumentProvider festlegen (Listing2). Für jeden Datentyp besitzt Workbench jeweils nur eine Instanz des zuständigen DocumentProviders.

```
public SmobEditor() {
      colorManager = new ColorManager();
      setSourceViewerConfiguration(
            new SmobEditorConfiguration(this, colorManager));
      setDocumentProvider(new SmobDocumentProvider());
}
```

Listing 2: Deklaration eines Editors, entnommen aus der SmobEditor.java (vgl. Anhang 1 A)

Alle Editoren in der Eclipse Workbench bauen auf der abstrakten Klasse *EditorPath* auf. Diese Klasse benutzt das Interface *IEditorInput* für die Beschreibung von Inhaltsquellen für den Editor. Aus diesem Grund wird eine Ressource dem Texteditor als IEditorInput zur Verfügung gestellt. Die Aufgabe des DocumentProvider ist es, aus dem Input ein entsprechendes Dokument zu erstellen, was in der Methode `createDocument` geschieht[34]. Diese Methode liefert ein Objekt von Typ *IDocument* zurück, das dann an

[34] Vgl. Anhang 1 B

den Editor übergeben wird. Damit ist ein funktionsfähiger Standrad-Text-Editor in Eclipse erstellt. Er verfügt bereits über die üblichen Funktionen zur Textbearbeitung wie, Editieren, Kopieren, Einfügen und Ausschneiden. Damit ist die in Kapitel 3 beschriebene Anforderung "Editieren" erfühlt.

In Eclipse Plattform werden einige Funktionalitäten des Texteditors mit der Hilfe einer *SourceViewerConfiguration* eingerichtet. Es sind unter anderen solche Funktionen, wie:

- Syntaxhervorhebung
- Modellabgleich - dient der automatischen Aktualisierung eines Dokumentmodells. Nach der Änderung des Quelltextes im Editor wird dieser für die Gliederungsansicht benötigt
- Inhaltsassistent - unterstützt bei der Codevervollständigung.

Syntaxhervorhebung

Die Registrierung von *SourceViewerConfiguration* geschieht in dem Konstruktor des Editors durch den Aufruf der Methode *setSourceViewerConfiguration* (Listing 2). Wie der Editor das Dokument darstellen soll, wird in der Methode *getPresentationReconciler(..)* festgelegt. Diese Methode liefert einen *PresentationReconciler* zurück. Dieser enthält zwei Objekte, einen *Damager* und einen *Repairer*. Sie sind dafür verantwortlich, dass bei jeder Änderung, die richtigen Stellen im Dokument neu gefärbt werden. Der Damager bestimmt, welche Textregion nach einer Änderung aktualisiert werden muss. Der Repairer färbt dagegen den entsprechenden Bereich[35].

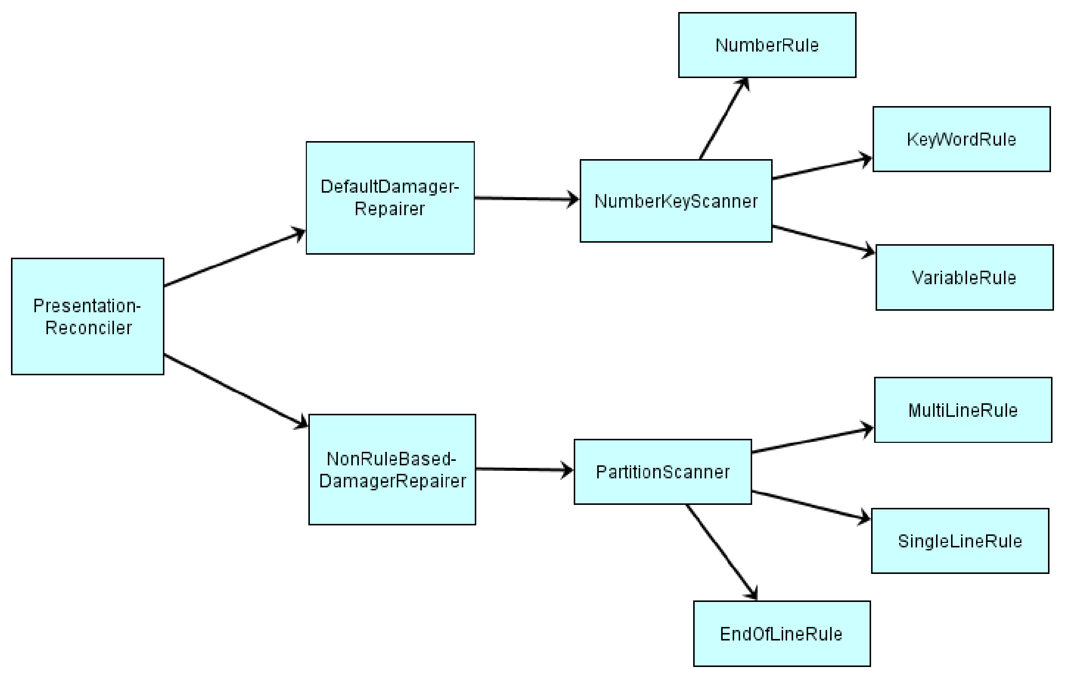
Weiter werden zwei Klassen verwendet, *DefaultDamagerRepairer* und *NonRuleBasedDamagerRepairer* (Abbildung 23). Der *DefaultDamagerRepairer* wird mit einem *ItokenScanner* konfiguriert, welcher den Text in Abschnitte gleicher Art zerteilt und die Darstellung der Abschnitte über ihre *TextAtributte* im *Token* beschreibt. Als Scanner wird hier der *NumberKeyScanner* verwendet[36]. Er ist von *RuleBaseScanner* abgeleitet und durchsucht das Dokument unter Verwendung sprachspezifischer Regeln.

[35] Vgl. Anhang 1 C

[36] Vgl. Anhang 2 A

Auf den Anforderungen basierend wurden für *NumberKeyScanner* die folgenden drei Regeln festgelegt:

- NumberRule, bestimmt die Zahlen im Quellcode.
- KeyWordRule, bestimmt die Schlüsselwörter und Funktionsnamen.
- VariableRule, bestimmt die Variablennamen.



Ab

Abbildung 23: Funktionsweise des IpresentationReconciler

Die Beschreibungen für Zahlen, Funktions- und Variablennamen sind fest im Quellcode definiert. Die Schlüsselwörter können jedoch von dem Benutzer neu definiert werden. Dazu bietet die Eclipse Plattform eine Möglichkeit die Einstellungen (*Preferences*) zu verändern. Im Fall des SMOB-Texteditors wird eine Code-Datei *LanguagesPreferencePage.java* erstellt[37]. Sie ermöglicht das Auswählen und das Laden einer Sprachdatei in dem Menü, *Preferences-> Language.*

Gliederung

In der Anforderung wird spezifiziert, dass Variablen- und Funktionsnamen in einem weiteren Fenster innerhalb der IDE angezeigt werden. Eclipse Plattform schreibt vor, dass zuerst ein Modell der Quellcode-Datei erstellt werden muss. Anhand dieses

[37] Vgl. Anhang 3

Modells kann die SMOB-IDE die einzelnen Fragmente(Variablen, Funktionen) einer SMOB-Code-Datei herausfiltern und in dem Fenster anzeigen. Dafür wird zuerst eine Basisklasse *ProgrammElement* definiert, von der alle weiteren Klassen des Modells erben. Das *ProgrammElement*[38] enthält Informationen über den ihm entsprechenden Textabschnitt (*getOffset()* und *getLength()),* sowie über seine Unterelemente (*getChildren()*).Das beschriebene Modell wurde speziell für die SMOB-Code-Datei erstellt. Die Vorgehensweise ist jedoch für Eclipse Plattform typisch und hängt nur von verschiedenen Anforderungen ab.

Der *ProgrammParser* baut aus dem Dokument ein Modell der Code-Datei und liefert schließlich ein Objekt der Klasse *Programm*[39]. Als zentraler Zugriffspunkt besitzt der SMOB-Texteditor eine Instanz der *Programm*-Klasse. Ein *Programm*-Objekt enthält die Instanzen der Klassen *SubroutinesSection* und *VariablesSection* (Abbildung 24). Die beiden „*Sections*" enthalten entsprechende Listen der vorhandenen *Subroutines* und *Variables*.

Wenn das Dokument verändert wird, ändert sich auch das zugehörige Modell. Damit die Gliederungsansicht immer aktuell bleibt, muss diese mit einem neuen Modell verglichen werden. Das Parsen und das Erstellen eines neuen Modells laufen in einem Hintergrund-Thread ab.

Dieses Thread wird aktiv, wenn das Dokument eine Zeit lang nicht verändert wird. Dazu stellt die Eclipse Plattform die *Reconsiling*-Infrastruktur zur Verfügung. Um eine Verbindung zu der Infrastruktur zu schaffen, muss die Methode *getReconciler(..)* in *SmobEditorConfiguration* überschrieben werden[40].

In dieser Methode wird ein *Reconciler*-Objekt erstellt. Er verwaltet das Hintergrund-Thread und unterstützt nur eine *ReconcilingStrategy* für das ganze Dokument. Mit der Hilfe von *ProgressMonitor* kann die festgelegte *ReconcilingStrategy* ihren Fortstschritt melden oder unterbrochen werden. Außerdem wird im *ProgressMonitor* die Wartezeit

---

[38] Vgl. Anhang 4 A

[39] Vgl. Anhang 4 B

[40] Vgl. Anhang 1 C

für das Hintergrund-Thread bestimmt, nach deren Ablauf der Vergleich des Modells beginnt (Abbildung 25).

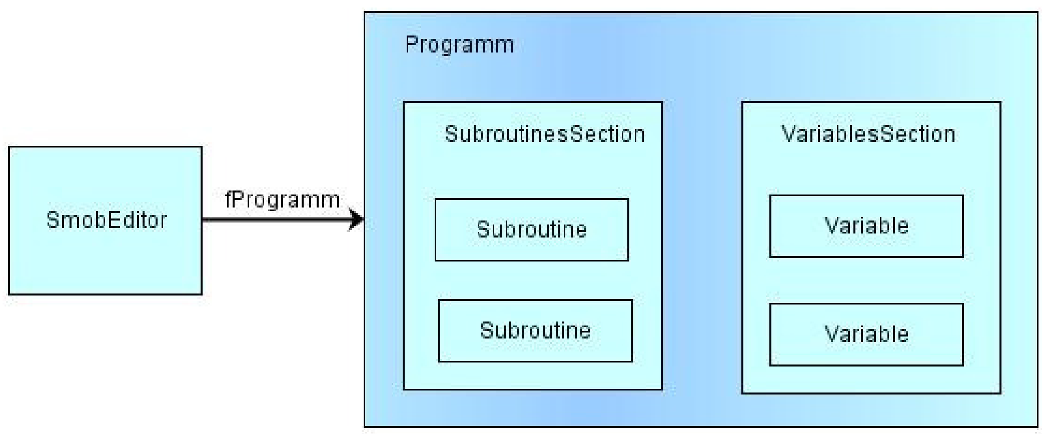


Abbildung 24: Das Modell einer SMOB-Code-Datei()

Die ProgrammReconcilingStrategy implementiert die Interfaces IReconcilingStrategy und IreconcilingStrategyExtension. Der Reconciler überreicht zuerst das Dokument und ein Objekt von Typ ProgressMonitor an die *ProgrammReconcilingStrategy* und ruft während des Modellabgleichs ihre Methode *reconcile(..)* auf[41].

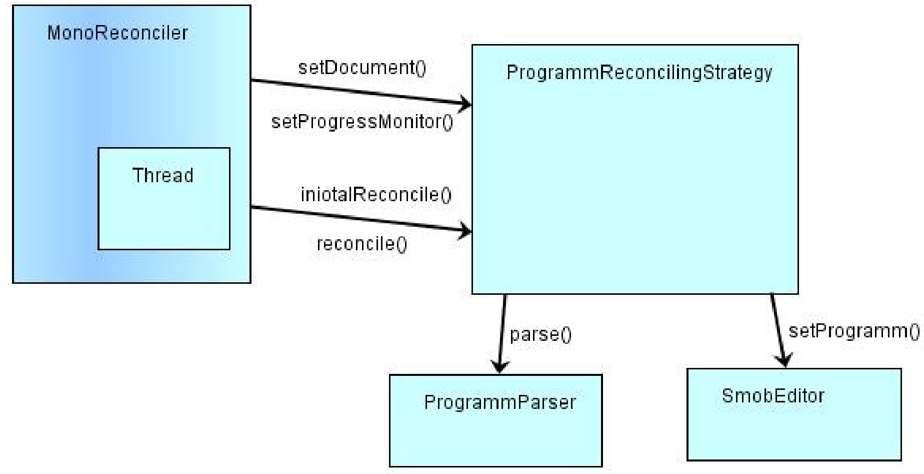


Abbildung 25: Modellabgleich – Übersicht

[41] Vgl. Anhang 4 D

Damit das erstellte Modell im Editor angezeigt werden kann, stellt Eclipse Plattform eine Gliederungsansicht bereit. Diese Ansicht zeigt immer die, zum aktiven Editor gehörende, Gliederung an. Die Verbindung von Editor zu der Ansicht findet über einen Adapter im Editor statt. Bei der Erstellung eines neuen Editors wird in der Methode *getAdapter (IContentOutlinePage.class)* nach einer Gliederungsseite gefragt und diese dann angezeigt.

Der SMOB-Texteditor fügt den Adapter für *IContentOutlinePage.class* hinzu und legt in der Methode *setProgramm(..)* fest, dass die Aktualisierung des Modells an die Gliederungsansicht weitergeleitet werden soll[42].

Für den SMOB-Texteditor wird in *ProgrammOutlinePage* eine eigene Gliederungsansicht implementiert. Die Gliederungsansicht wird in der Methode *createControl(..)* erzeugt. Die Implementierung von *ProgrammContentProvider* delegiert an die Methoden der Modelklasse *ProgrammElement*. Die Klasse *ProgrammLabelProvider* legt die Icons für die einzelnen Gliederungselemente fest[43].

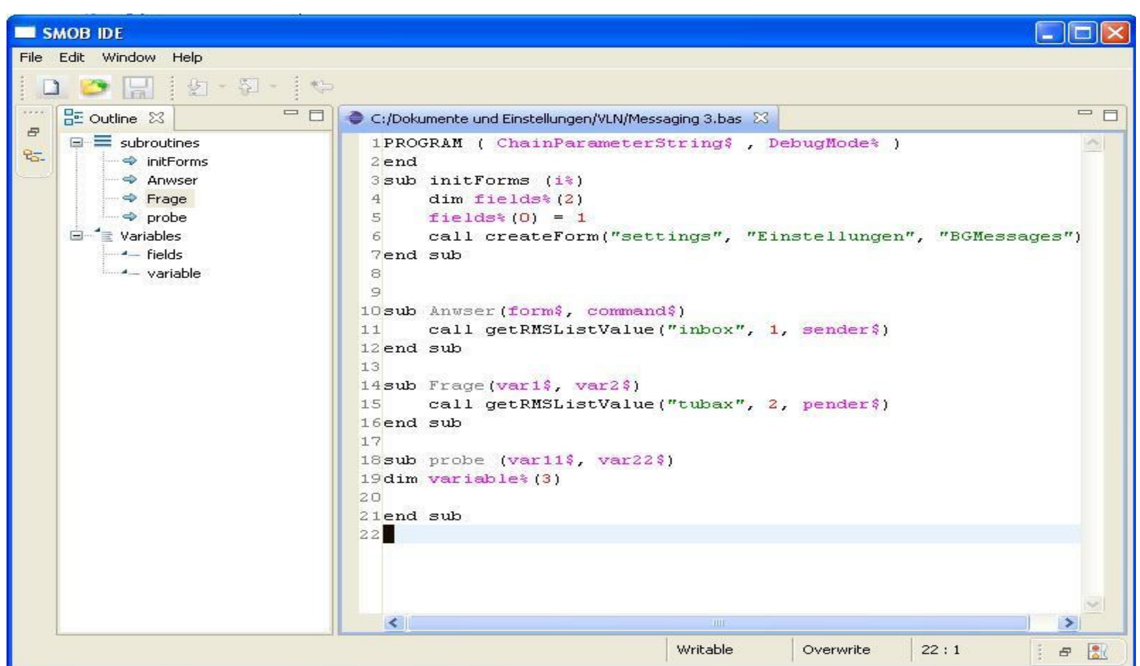


Abbildung 26: Gliederung einer SMOB-Code-Datei

Die Abbildung 26 zeigt ein Beispiel einer Gliederungsansicht. Die Variablen und die Funktionen werden in zwei untereinander gereihten Listen angezeigt.

[42] Vgl. Anhang 1 A

[43] Vgl. Anhang 4 E

Codevervollständigung

Für die Implementierung der Codevervollständigung bietet Eclipse Plattform ein Hilfswerkzeug, namens *ContentAssistant*, an. Es wird in der *SmobEditorConfiguration* konfiguriert, in der Funktion *getContentAssistant(..).* Dem *ContentAssistant* wird ein Objekt von Typ *ProgrammCompletionProcessor* übergeben. In *ProgrammCompletion-Processor* werden die Vorschläge für die möglichen Komplettierungen berechnet. Seine Funktion *computeCompletionProposals(..)* liest aus dem aktuellen Modell der SMOB-Code-Datei die vorhandenen Funktionen- und Variablennamen sowie die zur SMOB-Sprache gehörende Schlüsselwörter aus[44]. Damit die Komplettierungsliste angezeigt werden kann, muss eine zugehörige Action bei dem Texteditor registriert werden. Dies geschieht in dem SMOB-Texteditor in der Funktion createAction(..). Sie wird bei der Initialisierung des Editors aufgerufen und erzeugt eine Action von Typ *TextOperationAction*. In *Contributor* Klasse (Listing 1) wird die Aktion dem Komplettierungsassistenten zugeordnet.

5.3.1.2 NetBeans

Für das Erstellen einer RCP-Anwendung unter der NetBeans Plattform wird eine *Module-Suite* benutzt[45]. Sie wird unter *New Project*-> *NetBeans Modules*-> *Module Suite* erstellt. In den Eigenschaften der Suite, in der Kategorie *Build* wird „*Create Standalone Application*“ ausgewählt und dann in einem neuen Dialog mit *„exclude“* bestätigt. Das bewirkt, dass an die Anwendung nur die dringend benötigten Bibliotheken gebunden werden. Somit ist eine NetBeans Basis RCP-Anwendung fertig.

Datentyp

Der SMOB-Texteditor soll sich auf das Bearbeiten von SMOB-Sprachdateien spezialisieren. Dafür muss als Erstes ein neuer Datentyp für die SMOB-Code-Dateien definiert werden.

Es wird in dem Modul *SmobEditorSupport* neue Package erzeugt (Abbildung 27), die die Typ-Definition enthalten wird. Folgend wird in dieser Package ein neues *File Type*

---

[44] Vgl. Anhang 4 E

[45] Vgl. [15]

erstellt, in dem MIME[46]-Typ (`text/x-smob`) und die Erweiterung(`.bas`) festgelegt werden. Daraufhin erstellt die Plattform automatisch eine Reihe von Dateien, in denen die Eigenschaften von neuem Datentyp festgehalten werden.

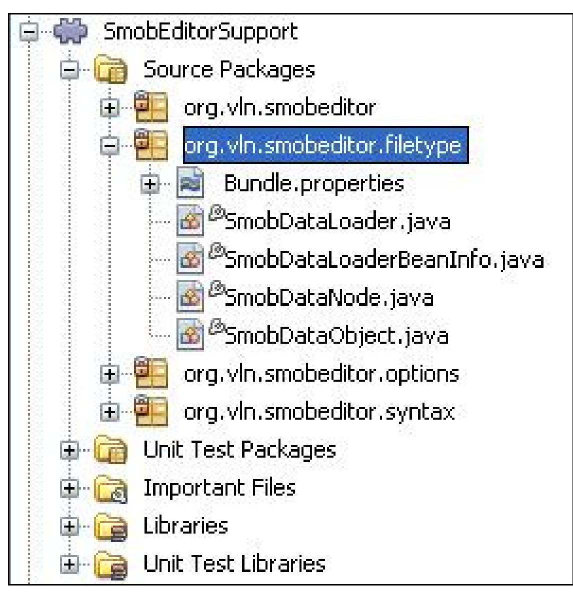


Abbildung 27: Übersicht des SmobFiletype-Moduls

Diese Dateien haben folgende Aufgaben[47]:

- *SmobDataLoader.java* - identifiziert den *text/x-smob* MIME-Typ und erzeugt Objekte der Klasse *SmobtDataObject.*
- *SmobResolver.xml* - bildet die *.bas* Erweiterung auf den MIME-Typ ab. Der *SmobDataLoader* identifiziert zwar den MIME-Typ, er weißt aber nichts über die Dateiendung.
- *SmobDataLoaderBeanInfo.java* – kontrolliert das Erscheinungsbild des *Loaders* in der Kategorie *Object Types* im Optionen-Fenster.
- *SmobtDataObject.java* – stellt einen Wrapper für *FileObject* dar. Dieser wird von *SmobDataLoader* erzeugt.
- *SmobDataNode.java* – bietet dem zugehörigen *FileObject* solche IDE-Funktionalitäten wie, Aktionen, Icons und lokale Namen.

Die Datentypen, die in der NetBeans Plattform identifiziert sind, haben ihre eigenen Icons, Menüeinträge und Verhalten. Die Dateien, die angezeigt werden, sind

[46] **MIME** - Multipurpose Internet Mail Extensions

[47] Vgl. Anhang 4 A-E

FileObjects, sogenannte Wrapper, die rund um die Java-Bibliothek `java.oi.File` gebildet wurden. In der Abbildung 28 dargestellte Nodes stellen für die Objekte, z.B.: *FileObjects,* solche Funktionalitäten wie Aktionen und lokale Namen bereit.

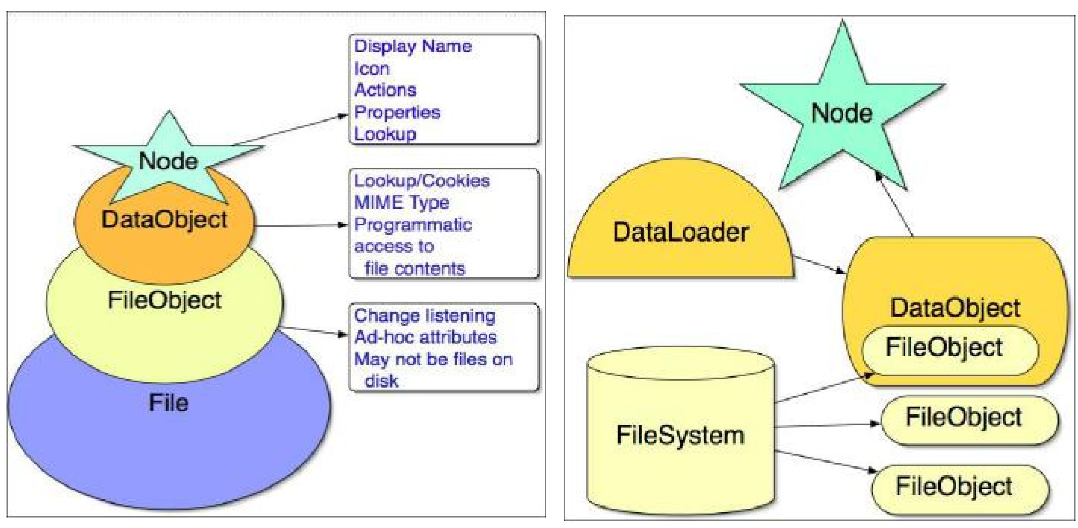


Abbildung 28: NetBeans Node System, links: Funktionalitäten, die jedes Objekt zur Verfügung stell; rechts, die Beziehungen zwischen den Objekten, Quelle [14]

Zwischen den *Nodes* und den *FileObjects* befinden sich die *DataObjects*. Sie sind den *FileObjects* ähnlich, verfügen aber im Gegenteil hierzu, Informationen darüber, welcher Art die Datei, die aktuell angezeigt wird, ist. Für die Dateien mit verschiedenen Endungen und für die XML-Dateien mit verschiedenen Namensräumen existieren verschiedene *DataObjects*. Jedes einzelne *DataObject* wird von einem eigenen Modul präsentiert, wobei jedes Modul einen oder mehrere Datentypen unterstützt. Ein Modul enthält einen *DataLoader*, der mit Hilfe der Fabrikmethode[48] einen Dateityp für das spezifische *DataObject* erstellt. Beim Öffnen einer Datei fragt die IDE bei allen bekannten *Loader* nach, ob der Dateityp bei ihnen registriert ist. Bekommt sie eine positive Antwort, wird von dem entsprechenden *Loader* ein *DataObject* erzeugt. Damit eine Datei tatsächlich angezeigt werden kann, wird von dem System die Funktion *getNodeDelegate()* in dem zugehörigen *DataObject* aufgerufen. Der neue erstellte *Node* präsentiert dann die Datei in der IDE.

[48] Fabrikmethode – Entwurfsmuster, beschreibt, wie ein Objekt durch Aufruf einer Methode anstatt durch direkten Aufruf eines Konstruktors erzeugt wird.

EditorKit

Nachdem der SMOB-Datentyp definiert wurde, muss ein *Editorkit*, das mit dem SMOB-Datentyp assoziiert ist, erstellt werden.

Ein EditorKit stellt Mitteln zur Verfügung, mit deren Hilfe ein funktionierender Texteditor für verschiedene Texttypen entwickelt werden kann. Durch den Aufruf der *setComponent(..)*-Methode eines *JTextComponent* wird eine Beziehung zwischen Editorkit und dem *JTextComponent* aufgebaut. Die EditorKit-Instanz speichert den Bearbeitungsstatus des Dokuments. Sie verwaltet alle Aktionen, die mit dem Datentyp verbunden sind. Ein Texteditor unter NetBeans Plattform wird erstellt dadurch, dass einem existierenden *JTextComponent* ein bestimmter EditorKit zugewiesen wird. So übernimmt der Texteditor alle Eigenschaften des *EditorKits*.

Für die Implementierung des SMOB-Editors wurde ein *SmobEditorKit*[49] auf Basis von *NbEditorKit* erstellt. *NbEditorKit* vereint die Eigenschaften von *BaseKit* und *ExtKit. Es* liefert ein Dokument von Typ `javax.swing.text.Document.` Dieser Dokumenttyp enthält als Inhalt eine lineare Abfolge von Zeichen. Der Zugriff auf die Zeichenketten oder auf einzelne Zeichen, findet durch die Angaben von Anfangsstelle (*offset*) und der gewünschten Länge statt (Abbildung 29).

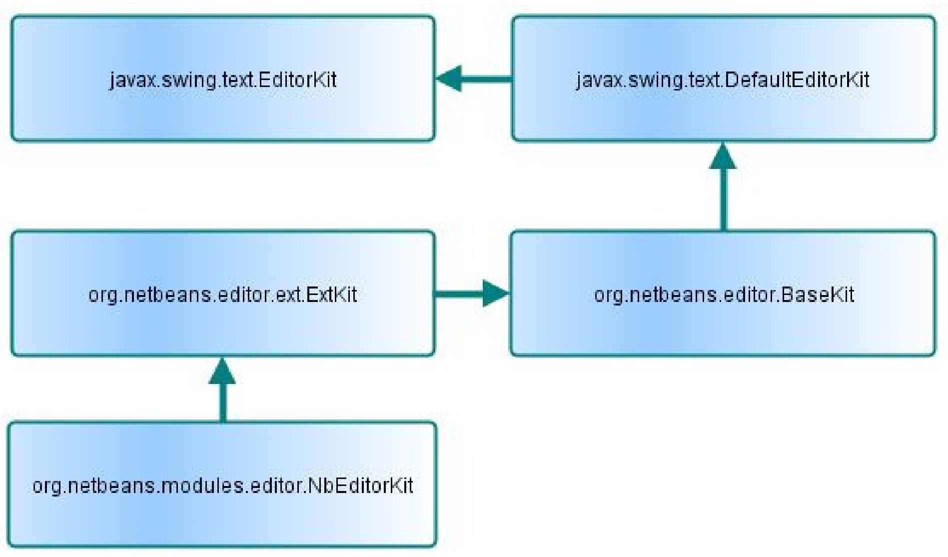


Abbildung 29: Übersicht der NetBeans EditorKit-Typen

[49] Vgl. Anhang 5 A

Außerdem bietet *NbEditorKit* eine Vielzahl von vordefinierten Funktionen zum Bearbeiten des Textes, z.B.: lösche bis zum Zeilenende, gehe zu dem nächsten oder vorherigen Wort.

```
@Override
public String getContentType(){
       return "text/x-smob";
   }
```

Listing3: Festelegung des MIME-Typs entnommen aus SbomEditorKit.java, Vgl. Anhang 5 A

Das *SmobEditorKit* wird in der Methode *getContent()* mit dem SMOB-Datentyp assoziiert (Listing 3). Anschließend muss er bei der Plattform registriert werden damit die NetBeans Plattform den SMOB-Datentyp diesem EditorKit zuordnen kann,. Das geschieht in der Projekt-Layer-Datei *layer.xml* (Listing 4). In diesem projekteigenen Layer wird ein neuer Ordner für den SMOB-Dateityp erstellt (Listing 4 <1>). Der IDE wird eine neue Datei, die Instanz der Klasse *SmobEditorKit,* hinzugefügt. Als Attribut wird der volle Name der Klasse übergeben (Listing 4 <2>). Anhand dessen wird die IDE informiert, welches EditorKit sie benutzen soll, wenn eine Datei von SMOB-Datentyp geöffnet wird.

```
<folder name="Editors">
  <folder name="text">
    <folder name="x-smob"> 1.
       <file name="Settings.settings"
                 url="options/SmobEditorOptions.settings" />3.
      <file name="EditorKit.instance"> 2.
         <attr name="instanceClass"
             stringvalue="org.vln.smobeditor.SmobEditorKit" />
      </file>
     </folder>
   </folder>
 </folder>
```

Listing 4: Registrierung des SmobEditorKits, entnommen aus layer.xml

Optionen

Als nächstes werden die Optionen für einen SMOB-Editor definiert. Dafür stellt NetBeans Plattform die *Options-API*, die die System-Optionen verwaltet, bereit. Diese Optionen werden für das einzelne Modul gesetzt und können über einen speziellen

Optionsdialog verändert werden. Die Optionen werden von der IDE automatisch auf der Festplatte gespeichert.

Für die Implementierung wird die Java-Klasse *SmobEditorOptions*, die von der Klasse *BaseOptions* ableitet, erstellt. *BaseOptions* ist ein Java-Bean. Es stellt die vordefinierten Optinonen, wie z.B. die Tastaturbelegung, die Blinkhäufigkeit des Kursors oder Anzeige der Statusleiste, für einen EditorKit bereit. Außerdem können eigene Eigenschaften definiert werden. In *SmobEditorOptions* ist das *mySampleProperty.* Es speichert einen Integer-Wert und liefert diesen zurück[50].

Um die EditorKit-Optionen in dem Optionsfenster anzeigen zu können, muss die Optionen-Bezeichnung festgelegt werden. Das geschieht in der *Bundle.properties* (Listing 5).

```
# Resource Bundle file for Smob Editor Options.
OPTIONS_smob=Smob Editor Options
```

Listing 4: Bezeichnung der SmobEditorKit-Optionen, entnommen aus Bundle.properties

Schließlich muss die Klasse bei der IDE registriert werden. Dazu wird in der Layer-XML-Datei des SmobEditorKits der „Settings.settings"-Eintrag gemacht (Listing 4 <3>). Dieser enthält die Url der Datei, namens *SmobEditorOptions.settings*. In dieser Datei werden die Einstellungen für das Modul, „*org.vln.smobeditor*", spezifiziert[51]. Wie es in der fertigen Anwendung aussieht, zeigt die Abbildung 30.

[50] Vgl. Anhang 6 A

[51] Vgl. Anhang 6 B

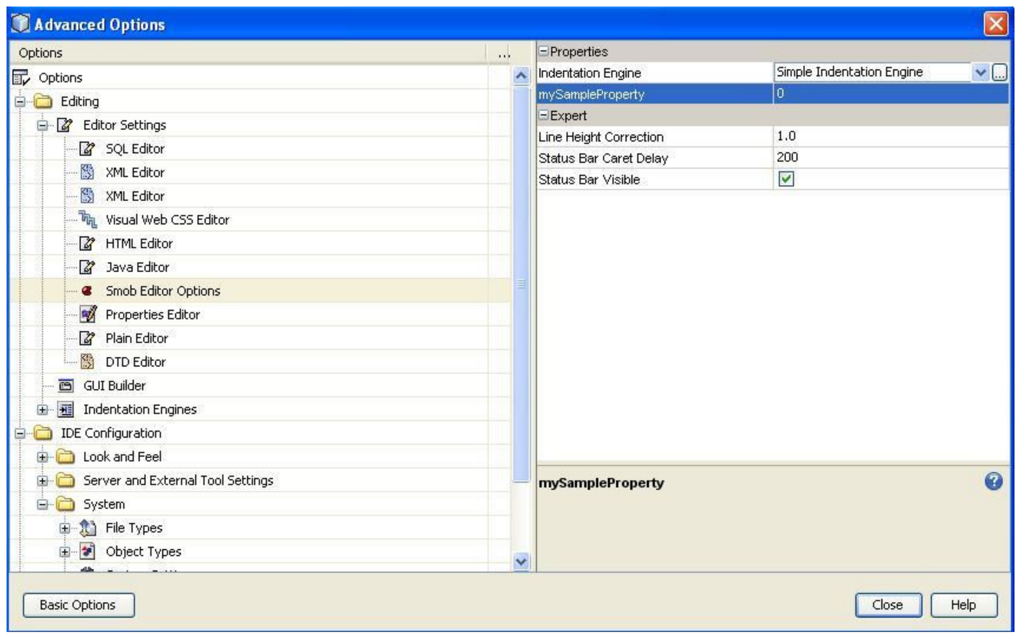


Abbildung 30: Zeigt Options-Fenster des SmobEditorKits

Syntaxhervorhebung

Nach den vorherigen Implementierungsschritten kann IDE beim Öffnen einer SMOB-Datei schon den richtigen EditorKit benutzen, nämlich SmobEditorKit. Der Code wird aber nur als einfacher Text dargestellt, ohne jegliche syntaxischen Trennung. Damit das erreicht werden kann, muss zuerst eine Syntax, speziell für die SMOB-Code, erstellt werden.

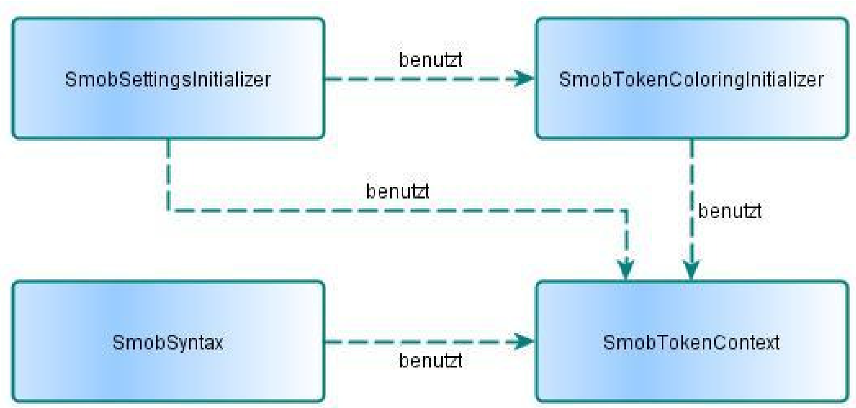


Abbildung 31: Übersicht der Klassen für Syntaxhervorhebung des SMOB-Codes

Die Abbildung 31 zeigt die Klassen, die für die Syntaxhervorhebung bei SmobeditorKit benötigt werden. Zuerst wird die Klasse *SmobTokenContext* angelegt. Ihre Aufgabe ist

es, die spezifische Tokens zu definieren. Für den SMOB-Code werden die Tokens für die Variablen und die Strings festgesetzt[52].

Natürlich muss die IDE unterscheiden können, welcher Teil des Textes zum welchen Token gehört. Auf welche Weise die IDE die Tokens erkennt, wird in der Klasse *SmobSyntax* definiert[53]. In dieser Klasse werden die Variablen festgelegt, die als Zustände fundieren. Mit der Hilfe dieser Zustände wird der Text lexikalisch analysiert. Es wird, angefangen mit „*INIT*"-Zustand, jeder Character des Textes untersucht und entschieden, ob der Character eine Zustandsänderung herbeiführt oder nicht (Abbildung 32). Dafür ruft die IDE die Methode *doParseToken()* auf. Wird ein Token erkannt, liefert die Methode die passende *TokenID* zurück. Diese *TokenID's* sind in der *SmobTokenContext* festgelegt.

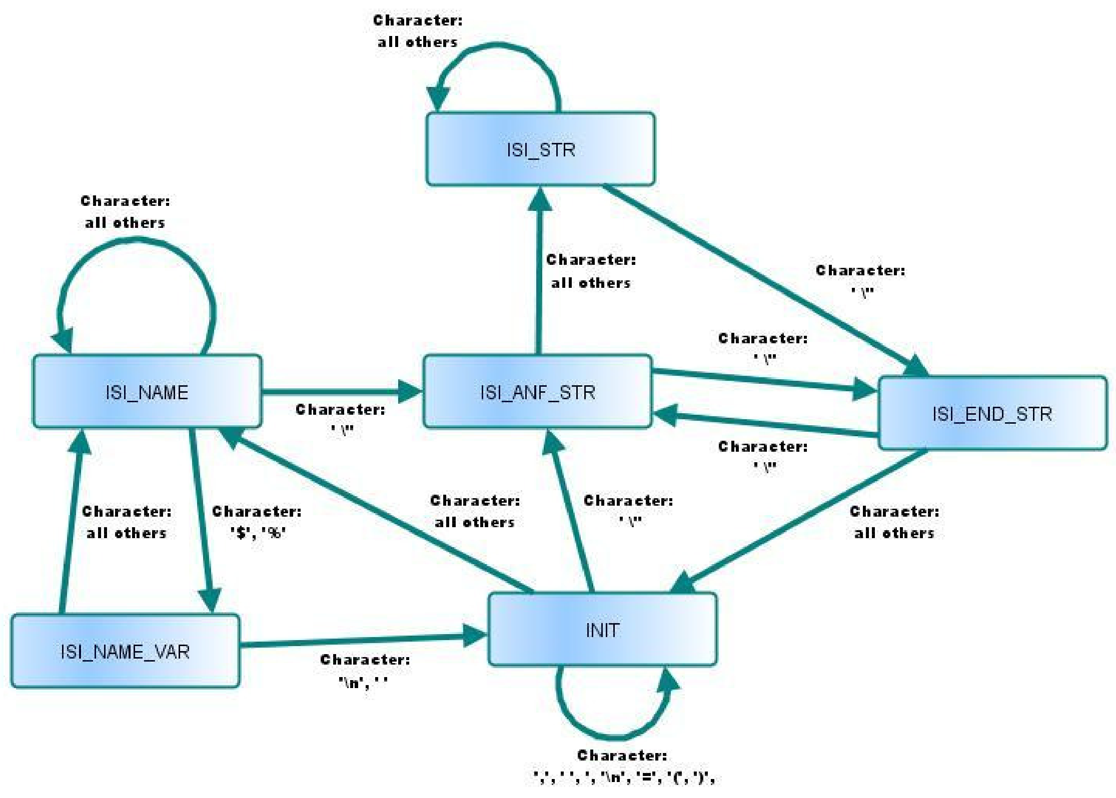


Abbildung 32: Automat für die lexikalische Analyse des SMOB-Codes

NetBeans Plattform stellt die Mittel zur Verfügung, die es dem Entwickler erlauben, erstellte Syntax ohne des Modulstarts zu prüfen. Dazu soll in der *Unit Test Package*

[52] Vgl.Anhang 7 A

[53] Vgl. Anhang 7 B

eine Klasse erstellt werden, die sich von der Klasse *TestCase* ableitet. In Fall der Smob-Syntaxprüfung ist es *SmobSyntaxTest*[54]. Die Klasse bekommt als Syntax die SMOB-Syntax zugewiesen. In der Methode *testNextToken()* werden dann der Test-String und die erwarteten Tokens, in der Reihenfolge ihres Auftretens, definiert. Wenn die Zustände und ihre Übergange richtig definiert wurden, läuft der Test erfolgreich durch.

Anschließend müssen noch die Farben für die Tokens bestimmt werden. Das geschieht in der Klasse *SmobSettingsInitializer*[55]. Dank der Unterstützung der Options-Funktionen können die Farben für die Tokens später in dem Optionsfenster vom Benutzer geändert werden.

```
Manifest-Version: 1.0
OpenIDE-Module: org.vln.smobeditor
OpenIDE-Module-Layer: org/vln/smobeditor/layer.xml
OpenIDE-Module-Localizing-Bundle: org/vln/smobeditor/Bundle.properties
OpenIDE-Module-Specification-Version: 1.0
OpenIDE-Module-Install: org/vln/smobeditor/Installer.class

Name: org/vln/smobeditor/filetype/SmobDataLoader.class
OpenIDE-Module-Class: Loader
```

Listing 5: Übersicht der Modul-Manifest-Datei

In der Methode *createSyntax(..)* wird dem *SmobeEditorKit*[56] die SMOB-Syntax zugewiesen. Des Weiteren wird in dem Modul-Installer, *Installer*[57], mit der Hilfe der Methode *addInitializer()*, *SmobSettingsInitializer* der IDE bekannt gemacht. Es bewirkt, dass gleich beim Starten des Modules der SmobEditorKit und seine Settings bei der IDE registriert werden. Um der IDE die Existenz des Modul-Installers bekannt zu geben, muss er in die Manifest-Datei des Moduls eingetragen werden (Listing 5).

Der entstandene EditorKit kann innerhalb der IDE benutzt werden. Bei der Öffnung einer SMOB-Code-Datei wird sie automatisch den SmobEditorKit dem Editor zuweisen. Der Entwickler kann außerdem den erstellten EditorKit in einen Standalone-

---

[54] Vgl. Anhang 7 D

[55] Vgl. Anhang 7 C

[56] Vgl. Anhang 5 A

[57] Vgl. Anhang 5 B

Editor integrieren. Dafür muss in der Programmoberfläche eine *JeditorPane* erstellt und in ihren Eigenschaften der SmobEditorKit als EditorKit eingestellt werden.

5.4 Schlussfolgerung

Das Ziel dieser Arbeit ist es, zu analysieren, welche Mittel, zur Erstellung einer Entwicklungsumgebung, die Plattformen dem Entwickler zur Verfügung stellen. Im Verlauf dieser Arbeit wurden zwei Entwicklungsumgebungen, die Eclipse und die NetBeans betrachtet. Es wurden mittels beider Plattformen die Rich Client Platform Anwendungen erstellt. In beiden Anwendungen wurde jeweils ein Texteditor implementiert, der die gestellten Anforderungen[58] erfüllen sollte. Im Prozess der Anwendungserstellung wurden Erfahrungen mit den Entwicklungsumgebungen gesammelt.

Bei der Entwicklung der RCP Anwendungen bietet die Eclipse Plattform dem Entwickler eine Unterstützung in Form von Assistenten an. Als Beispiel werden bei der Erstellung eines Editors Editor-Vorlagen geboten. Wählt der Entwickler eine der Vorlagen, wird von der Plattform der entsprechende Editor erzeugt. Die erstellte RCP Anwendung mit dem Mustereditor kann gestartet und gestestet werden. Diese Unterstützung erleichtert den Einstieg in die Entwicklung der RCP Anwendungen. Besonders für die Entwickler, die mit der Eclipse wenig Erfahrung haben, bedeutet dies einen geringeren Einarbeitungsaufwand. Andererseits wird dem Entwickler dadurch nur ein oberflächlicher Einblick in die internen Abläufe gewährt.

Bei der NetBeans Plattform muss der Entwickler sich länger einarbeiten. Im Falle, dass ein Editor erstellt werden soll, müssen bestimmte Schritte eingehalten werden[59]. Ich bin der Meinung, dass dieses Vorgehen dem Entwickler den Vorteil bietet, eine tiefere Einsicht in die Funktion der Plattform zu gewinnen. Seit der Version 6.0 bietet auch die NetBeans Plattform Code-Beispiele an. Die Beispiele lassen sich aber nur als neue Projekte erstellen. Die Lösung von Eclipse, die Code-Beispiele in eigene Anwendung integrieren zu können, halte ich für sinnvoller.

---

[58] Vgl. (Kapitel 4)

[59] Vgl.(Kapitel 5.5.1.2)

Wegen des begrenzten Zeitrahmens einer Diplomarbeit konnten nicht alle gestellten Anforderungen implementiert werden. Die Implementierung des SMOB-Editors hat gezeigt, dass die Plattformen bei der Erstellung einer RCP Anwendung unterschiedliche Ansätze verfolgen. Es ließen sich dennoch mit beiden Plattformen die RCP-Anwendungen erstellen. Auch aus den Vergleichen[60] der beiden Plattformen, wie die Onlinerecherche ergab, lässt sich keine Schlussfolgerung ziehen. Die Autoren distanzieren sich von der Aussage, eine IDE sei besser als die Andere. Es werden verschiedene Strategien der Plattformen bei der Entwicklung der RCP Anwendungen festgestellt. Beide Plattformen stellen eine Vielzahl an Funktionalitäten zur Verfügung und gestatten dem Entwickler professionelle RCP Anwendungen, mit Menüs, Views und Editoren, zu programmieren. Bei der endgültigen Entscheidung muss der Entwickler sich darauf festlegen, welcher internen Struktur der betrachteten Plattformen die zukünftige RCP Anwendung entsprechen soll[61]. Die Softwareentwicklung für mobile Geräte wird von beiden Plattformen unterstützt. Die Eclipse Plattform bietet dafür die *embedded Rich Client Platform* (eRCP) an. Die NetBeans Plattform stellt den Entwickler ihrerseits das NetBeans Mobility Pack zur Verfügung[62].

Nach der durchgeführten Analyse lautet die Empfehlung zu Gunsten der Eclipse Plattform. Diese Empfehlung beruht auf den Erfahrungen, die im Verlauf der Diplomarbeit gesammelt wurden. Die Onlinerecherche ergab, dass zur Eclipse Plattform eine viel größere Anzahl von Hilfestellungen, sowohl in englischer als auch in deutscher Sprache, zur Verfügung gestellt werden. Darunter sind Tutoriums und Codebeispiele zu verstehen. Bei der NetBeans Plattform wurde die Hilfesuche durch mangelnde Präsenz von Anleitungen und Muster-Anwendungen erschwert. Als einer der Beispiele kann hier die Anleitung für die Syntaxhervorhebung eines Manifest-Dateien Editors[63], auf der NetBeans-Webseite genannt werden. Trotz des Befolgens aller Arbeitsschritte der Anleitung präsentierte die erstellte Anwendung keine funktionsfähige Syntaxhervorhebung. Auch die zugehörende Muster-Applikation von der NetBeans-Webseite funktionierte nicht erwartungsgemäß. In der NetBeans Version 6.0, ließ sich

[60] Vgl. Artikel[18], [19], [15]

[61] Vgl. [15]

[62] Vgl. [20], [21]

[63] Vgl. [17]

diese Anwendung zwar starten, jedoch funktionierte die entsprechende Syntaxhervorhebung nicht. Die genannten Tatsachen haben die Entwicklung des SMOB-Editors unter NetBeans wesentlich verzögert. Dies soll jedoch nicht als qualitative Abwertung der NetBeans Plattform gewertet werden, sondern als Hinweis dienen, dass die NetBeans Plattform sich noch in der Entwicklung befindet. Daher besteht bei den Hilfestellungen, Dokumentationen und Tutoriums noch Verbesserungsbedarf.

Ein weiterer Grund für die Eclipse Empfehlung ist der Einsatz der Eclipse Plattform in der Firma StarFinanz. Die Umstellung auf NetBeans würde, unter anderem aus oben genannten Gründen, einen zusätzlichen Zeitaufwand bedeuten.

Außerdem besitzt die Eclipse große Community.[64] Dadurch besteht die Möglichkeit, wenn die IDE in der SMOB-Sprache in der Eclipse entwickelt wird, dass eine größere Anzahl der potentiellen Benutzer bereits vorhanden ist.

6 Ausblick

Im Verlauf dieser Arbeit, wurde das Wissen über die beiden Plattformen, Eclipse und NetBeans, angeeignet. Es wurde die Einsicht in die Funktionsweise der Plattformen in der Hinsicht auf die Erstellung von RCP-Anwendungen gewonnen. Die durchgeführte Analyse ergab, dass sich mit beiden Plattformen qualitative Rich Client Platform Anwendungen erstellen lassen. Unbeachtet dessen, stellt die Entwicklungsumgebung von NetBeans eine ernst zunehmende Konkurrenz zur Eclipse dar. Es ist zu erwarten, dass mit weiteren Versionen die NetBeans Plattform ihre Position auf dem Markt weiter verstärken kann.

[64] Vgl. [21]

Glossar

ANT: ein in Java geschriebenes Tool zum automatisierten Erzeugen von Programmen aus Quelltext

ANTLR: ANother Tool for Language Recognition, ein objektorientierter Parsergenerator.

Branding: die immaterielle Summe der Attribute eines Produkts: sein Name, Verpackung und Preis, seine Geschichte, seine Reputation und die Art, wie es beworben wird.

Canvas: Klasse welche den Eingangspunkt für grafische Oberflächen bietet.

CDC: Connected Device Configuration

CLDC: Connected Limited Device Configuration

CSS: Cascading Style Sheets, Sprache zur optischen Formatierung von XML-Dialekten.

CPL: Common Public License, eine Open-Source- Lizenz.

Eclipse: Open-Source Entwicklungsumgebung.

Equinox: ein Java-basiertes Framework, eine Implementierung der OSGi R4 Core Framework-Spezifikation.

Telematik: eine Technologie, welche die Technologiebereiche Telekommunikation und die Informatik verknüpft.

GPL: General Public License, eine Open-Source-Linzenz.

Framework: ist ein Programmiergerüst, welches in der Softwaretechnik insbesondere im Rahmen der objektorientierten Softwareentwicklung sowie bei komponentenbasierten Entwicklungsansätzen verwendet wird.

HTML: Hypertext Markup Language, Sprache zur Darstellung von Inhalten in Form von Webseiten.

JFace: ein UI-Toolkit auf Basis von SWT

JPEG: Joint Photographic Experts Group, ein verlustbehafteter Bild-kompressionsformat.

IP: Internet Protocol, Vermittlungs-Protokoll, siehe RFC 791.

JCP: Java Community Process, ein Verfahren, das bei der Weiterentwicklung der Programmiersprache Java und ihrer Standardbibliothek angewendet wird.

JSR: Java Specification Request, eine Anforderung einer neuen Java-Spezifikation oder einer wichtigen Änderung einer existierenden Java-Spezifikation.

J2ME: Java 2 Micro Edition

J2ME-Polish: ein Werkzeug zur visuellen Aufbereitung und Modifikation von J2ME GUI Komponenten.

J2SE: Java 2 Standard Edition

KVM: Kilobyte Virtual Machine, besonders kleine virtuelle Maschinen

Make: ist ein Computerprogramm, das einem Kommandointerpreter ähnlich Befehle in Abhängigkeit von Bedingungen ausführt.

MIDI: musical instrument digital interface, ein Datenübertragungs-Protokoll, das Übermittlung, Aufzeichnung und Wiedergabe von musikalischen Steuerinformationen mit einem PC ermöglicht. Enthält folgende Formate: SMF 0, SMF 1 und SMF 2.

MIDP: Mobile Information Device Profile

Mono: .NET-kompatible Entwicklungs- und Laufzeitumgebung für plattform-unabhängige Software

MP3: ein verlustbehafteter Audiodatenkompressionformat.

Online-Banking: ermöglicht es dem Anwender Bankgeschäfte über eine Online-Verbindung zu tätigen.

Smartphones: Handys mit zusätzlicher Funktionalität, welche über die eines normalen Handys

SWT: Standard Widget Toolkit, eine Bibliothek für die Erstellung grafischer Oberflächen mit Java. Sie wurde von IBM für die Entwicklungsumgebung Eclipse entwickelt.

Swing: eine Programmierschnittstelle und Grafikbibliothek zum Programmieren von grafischen Benutzeroberflächen.

TCP: Transmission Control Protocol, verbindungsorientiertes Transportprotokoll.

URL: Uniform Resource Locator, Standard zur Adressierung von Dokumenten im Internet.

WAP: Wireless Application Protocol

WebBanking: ermöglicht dem Anwender Online-Banking über einen Browser zu betreiben. Der Anwender verbindet sich mit dem Online-Banking-Server seiner Bank über Webseiten.

Web-Client: nimmt die Verbindung mit dem Web-Server auf und tauscht mit ihm die Nachrichten.

Widget: ein kleines eigenständiges Programm, das auf der grafischen Oberfläche des Betriebssystems angezeigt wird.

Wrapper: ein Programm, das als Schnittstelle zwischen dem aufrufenden und dem umschlossenen Programmcode agiert

Literaturverzeichnis

[1] Sun Microsystems Inc. The Java Community Process

Quelle: http://java.sun.com/javame/technology/index.jsp (siehe CD)

[2] Java Me Spezifikation,

Quelle: http://www.jcp.org/en/jsr/detail?id=30 (siehe CD)

[3] Java Community Process,

Quelle: http://www.jcp.org

[4] Mobile Information Device Profile,

Quelle: http://java.sun.com/products/midp/ (siehe CD)

[5] Klaus-Dieter Schmatz: "Java Micro Edition. Entwicklung mobiler JavaME-Anwendungen mit CLDC und MIDP" 2007 dpunkt.verlag GmbH

Kapitel 2 aus dem Buch Java 2 Mircoedition von Klaus-Dieter Schmatz

Quelle: http://www.dpunkt.de/leseproben/3-89864-271-2/Kapitel_2.pdf (siehe CD)

[6] .NET Compact Framework,

Quelle: http://msdn.microsoft.com/de-de/library/9s7k7ce5(VS.80).aspx (siehe CD)

[7] J2ME Polish,

Quelle: http://www.j2mepolish.org

[8] J2ME Polish, Artikel in JavaSpektrum 2/2005, „Mobile Anwendungen mit J2ME Polish" Autor: Thomas Kraft (siehe CD)

[9]BASIC- Programmiersprache

Quelle: http://www.bitsavers.org/pdf/dartmouth/BASIC_Oct64.pdf (siehe CD)

[10] ANTLR,

Quelle: http://www.antlr.org/

[11] OSGi Alliance

Quelle: http://www.osgi.org

[12] Eclipse-Webseite,

Quelle: http://www.eclipse.org/downloads/moreinfo/compare.php

[13] NetBeans-Webseite,

Quelle: http://download.netbeans.org/netbeans/6.0/final/

[14] NetBeans-Webseite, File Type Tutorial

Quelle: http://platform.netbeans.org/tutorials/nbm-filetype.html, (siehe CD)

[15] Artikel in Eclipse-Magazine Vol.12, Kai Tödter: „Das Rad muss nicht neu erfunden werden!" (siehe CD)

[16] Autor: Heiko Böck, „NetBeans Plattform 6 Rich-Client-Entwicklung mit Java“ Galileo Press, Bonn 2008, 1. Auflage

[17] Tutorial auf der NetBeans-Webseite „Manifest File Syntax Highlighting Module“ Quelle: http://platform.netbeans.org/tutorials/nbm-mfsyntax.html (siehe CD)

[18] Artikel**: "**Eclipse, NetBeans, and IntelliJ: Assessing the Survivors of the Java IDE Wars”, Quelle: http://www.devx.com/Java/Article/34009 (siehe CD)

[19]Linux-Magazin, Ausgabe 03/08, Artikel: “NetBeans 6.0 und Ecluipse 3.3 im Vergleich” Autor: Markus Franz (siehe CD)

[20] Eclipse-Webseite für embedded Rich Client Platform (eRCP), Quelle: http://www.eclipse.org/ercp/

[21] Webseite für NetBeans IDE Support for Java ME Developers, Quelle: http://mobility.netbeans.org/

Anhang 1

A

SmobEditor.java

```
package com.starfinanz.smob.ide.editor;
import java.util.Enumeration;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.text.source.ISourceViewer;
import org.eclipse.jface.text.source.projection.ProjectionSupport;
import org.eclipse.ui.editors.text.TextEditor;
import org.eclipse.ui.texteditor.ITextEditorActionDefinitionIds;
import org.eclipse.ui.texteditor.TextOperationAction;
import org.eclipse.ui.views.contentoutline.IContentOutlinePage;
import com.starfinanz.smob.ide.editor.syntax.ColorManager;
import com.starfinanz.smob.ide.model.Programm;
import com.starfinanz.smob.ide.outline.ProgrammOccurrencesUpdater;
import com.starfinanz.smob.ide.outline.ProgrammOutlinePage;
import com.starfinanz.smob.ide.start.Activator;


public class SmobEditor extends TextEditor {
	private ColorManager colorManager;
	private Programm fProgramm;
	private ProgrammOutlinePage fOutlinePage;
	private ProjectionSupport fProjectionSupport;
	private ProgrammOccurrencesUpdater fOccurrencesUpdater;
	public SmobEditor() {
		colorManager = new ColorManager();
		setSourceViewerConfiguration(new             SmobEditorConfiguration(this,
colorManager));
		setDocumentProvider(new SmobDocumentProvider());
	}
	public void dispose() {
		colorManager.dispose();
		super.dispose();
	}
	public Programm getProgramm() {
		return fProgramm;
	}
	public void setProgramm(Programm programm) {
		fProgramm = programm;
		if (fOutlinePage != null)
			fOutlinePage.setProgramm(programm);

		if (fOccurrencesUpdater != null)
			fOccurrencesUpdater.update(getSourceViewer());
	}
	protected void performSaveAs(IProgressMonitor progressMonitor) {
		super.performSaveAs(progressMonitor);
	}
	protected void performSave(boolean overwrite, IProgressMonitor progressMonitor) {

		super.performSave(overwrite, progressMonitor);
	}
```

```
	/*
	 * @see org.eclipse.core.runtime.IAdaptable#getAdapter(java.lang.Class)
 */
public Object getAdapter(Class required) {
		if (IContentOutlinePage.class.equals(required)) {
			if (fOutlinePage == null)
				fOutlinePage= new ProgrammOutlinePage(this);
			return fOutlinePage;
		}
		if (fProjectionSupport != null) {
			Object adapter= fProjectionSupport.getAdapter(getSourceViewer(),
required);
			if (adapter != null)
				return adapter;
		}
		return super.getAdapter(required);
	}
	public void outlinePageClosed() {
		fOutlinePage= null;
	}
	protected void createActions() {
		super.createActions();
	Enumeration<String> temp =
Activator.getDefault().getResourceBundle().getKeys();
		String strtemp;
		IAction action= new TextOperationAction(
				Activator.getDefault().getResourceBundle(),
				"ContentAssistProposal.",
				this,
				ISourceViewer.CONTENTASSIST_PROPOSALS);
		action.setActionDefinitionId(
ITextEditorActionDefinitionIds.CONTENT_ASSIST_PROPOSALS);
		setAction("ContentAssist.", action);
		markAsStateDependentAction("ContentAssist.", true);

		while (temp.hasMoreElements()) {
			strtemp=temp.nextElement();
			if (strtemp.equals("sprache")){
				System.out.println(strtemp);
			}else {
				System.out.println("Fehler beim Lesen der Datei");
			}
		}

	}
}
```

B

SmobDocumentProvider.java

```
package com.starfinanz.smob.ide.editor;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.Reader;
import java.io.Writer;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IPath;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.core.runtime.IStatus;
import org.eclipse.core.runtime.Status;
import org.eclipse.jface.operation.IRunnableContext;
import org.eclipse.jface.text.Document;
import org.eclipse.jface.text.IDocument;
import org.eclipse.jface.text.source.IAnnotationModel;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IPathEditorInput;
import org.eclipse.ui.texteditor.AbstractDocumentProvider;
import org.eclipse.jface.text.IDocumentPartitioner;
import org.eclipse.jface.text.rules.FastPartitioner;
import com.starfinanz.smob.ide.editor.syntax.PartitionScanner;
/**
 * A document provider that reads can handle <code>IPathEditorInput</code>
 * editor inputs. Documents are created by reading them in from the file that
 * the <code>IPath</code> contained in the editor input points to.
 *
 * @since 3.0
 */
public class SmobDocumentProvider extends AbstractDocumentProvider {
        /**
         *                                                                        @see
org.eclipse.ui.texteditor.AbstractDocumentProvider#createDocument(java.lang.Object)
         */
        protected IDocument createDocument(Object element)
throws CoreException {
                IDocument document= new Document();
                if (document != null) {
                        IDocumentPartitioner partitioner =
                                new FastPartitioner(
                                        new PartitionScanner(),
                                        new String[] {
                                                PartitionScanner.COMMENT });
                        partitioner.connect(document);
                        document.setDocumentPartitioner(partitioner);
                }
                if (element instanceof IEditorInput) {
                        setDocumentContent(document, (IEditorInput) element);
                        return document;
                }
                return null;
        }
        /**
         * Tries to read the file pointed at by <code>input</code> if it is an
         * <code>IPathEditorInput</code>. If the file does not exist, <code>true</code>
         * is returned.
         *
         * @param document the document to fill with the contents of <code>input</code>
         * @param input the editor input
         *  @return <code>true</code> if setting the content was successful or no file
exists, <code>false</code> otherwise
```

```
	 * @throws CoreException if reading the file fails
	 */
	private boolean setDocumentContent(IDocument document,
IEditorInput input) throws CoreException {
		// XXX handle encoding
		Reader reader;
		try {
			if (input instanceof IPathEditorInput)
				reader= new FileReader((
(IPathEditorInput)input).getPath().toFile());
			else
				return false;
		} catch (FileNotFoundException e) {
			// return empty document and save later
			return true;
		}
		try {
			setDocumentContent(document, reader);
			return true;
		} catch (IOException e) {
			throw new CoreException(
new Status(IStatus.ERROR,
"org.eclipse.ui.examples.rcp.texteditor",
IStatus.OK, "error reading file", e));
		}
	}
	/**
	 * Reads in document content from a reader and fills <code>document</code>
	 *
	 * @param document the document to fill
	 * @param reader the source
	 * @throws IOException if reading fails
	 */
	private void setDocumentContent(IDocument document, Reader reader)
throws IOException {
		Reader in= new BufferedReader(reader);
		try {
			StringBuffer buffer= new StringBuffer(512);
			char[] readBuffer= new char[512];
			int n= in.read(readBuffer);
			while (n > 0) {
				buffer.append(readBuffer, 0, n);
				n= in.read(readBuffer);
			}
			document.set(buffer.toString());
		} finally {
			in.close();
		}
	}
	protected IAnnotationModel createAnnotationModel(Object element)
throws CoreException {
		return null;
	}
	protected void doSaveDocument(IProgressMonitor monitor,
Object element, IDocument document, boolean overwrite)
throws CoreException {
		if (element instanceof IPathEditorInput) {
			IPathEditorInput pei= (IPathEditorInput) element;
			IPath path= pei.getPath();
			File file= path.toFile();
			try {
				file.createNewFile();
				if (file.exists()) {
					if (file.canWrite()) {
						Writer writer= new FileWriter(file);
						writeDocumentContent(document,
writer, monitor);
					} else {
						throw new CoreException(
new Status(IStatus.ERROR,
```

```
"com.starfinanz.smob.ide.editor",
IStatus.OK, "file is read-only", null));
                                        }
                                } else {
                                        throw new CoreException(
new Status(IStatus.ERROR,
"com.starfinanz.smob.ide.editor",
IStatus.OK, "error creating file", null
                                }
                        } catch (IOException e) {
                                throw new CoreException(
new Status(IStatus.ERROR,
"com.starfinanz.smob.ide.editor",
IStatus.OK, "error when saving file", e));
                        }
                }
        }
        /**
         * Saves the document contents to a stream.
         *
         * @param document the document to save
         * @param writer the stream to save it to
         * @param monitor a progress monitor to report progress
         * @throws IOException if writing fails
         */
private void writeDocumentContent(IDocument document, Writer writer,
IProgressMonitor monitor) throws IOException {
                Writer out= new BufferedWriter(writer);
                try {
                        out.write(document.get());
                } finally {
                        out.close();
                }
        }
        protected IRunnableContext getOperationRunner(
IProgressMonitor monitor) {
                return null;
        }

        public boolean isModifiable(Object element) {
                if (element instanceof IPathEditorInput) {
                        IPathEditorInput pei= (IPathEditorInput) element;
                        File file= pei.getPath().toFile();
                        return file.canWrite() || !file.exists();
// Allow to edit new files
                }
                return false;
        }

        public boolean isReadOnly(Object element) {
                return !isModifiable(element);
        }

        public boolean isStateValidated(Object element) {
                return true;
        }
}
```

C

SmobEditorConfiguration.java

```
package com.starfinanz.smob.ide.editor;

import org.eclipse.core.runtime.NullProgressMonitor;
import org.eclipse.jface.text.IDocument;
import org.eclipse.jface.text.ITextDoubleClickStrategy;
import org.eclipse.jface.text.TextAttribute;
import org.eclipse.jface.text.contentassist.ContentAssistant;
import org.eclipse.jface.text.contentassist.IContentAssistProcessor;
import org.eclipse.jface.text.contentassist.IContentAssistant;
import org.eclipse.jface.text.presentation.IPresentationReconciler;
import org.eclipse.jface.text.presentation.PresentationReconciler;
import org.eclipse.jface.text.reconciler.IReconciler;
import org.eclipse.jface.text.reconciler.MonoReconciler;
import org.eclipse.jface.text.rules.DefaultDamagerRepairer;
import org.eclipse.jface.text.rules.ITokenScanner;
import org.eclipse.jface.text.rules.RuleBasedScanner;
import org.eclipse.jface.text.rules.Token;
import org.eclipse.jface.text.source.ISourceViewer;
import org.eclipse.jface.text.source.SourceViewerConfiguration;

import com.starfinanz.smob.ide.editor.assist.ProgrammCompletionProcessor;
import com.starfinanz.smob.ide.editor.syntax.ColorManager;
import com.starfinanz.smob.ide.editor.syntax.IColorConstants;
import com.starfinanz.smob.ide.editor.syntax.NonRuleBasedDamagerRepairer;
import com.starfinanz.smob.ide.editor.syntax.NumberKeyScanner;
import com.starfinanz.smob.ide.editor.syntax.PartitionScanner;
import com.starfinanz.smob.ide.editor.syntax.Scanner;
import com.starfinanz.smob.ide.editor.syntax.TagScanner;

public class SmobEditorConfiguration extends SourceViewerConfiguration {
	private DoubleClickStrategy doubleClickStrategy;
	private TagScanner tagScanner;
	private Scanner scanner;
	private ColorManager colorManager;
	private SmobEditor fEditor;

	public SmobEditorConfiguration(ColorManager colorManager) {
		this.colorManager = colorManager;
	}

public SmobEditorConfiguration(SmobEditor editor, ColorManager colors){

		this.fEditor = editor;
		this.colorManager = colors;
}
public String[] getConfiguredContentTypes(
ISourceViewer sourceViewer){
		return new String[] {
			IDocument.DEFAULT_CONTENT_TYPE,
			PartitionScanner.COMMENT };
}
public ITextDoubleClickStrategy getDoubleClickStrategy(
		ISourceViewer sourceViewer,
		String contentType) {
		if (doubleClickStrategy == null)
			doubleClickStrategy = new DoubleClickStrategy();
		return doubleClickStrategy;
}
```

```
public IPresentationReconciler getPresentationReconciler(
ISourceViewer sourceViewer) {
		PresentationReconciler reconciler =
new PresentationReconciler();

		DefaultDamagerRepairer dr =
			new DefaultDamagerRepairer(getScanner());
		reconciler.setDamager(dr, IDocument.DEFAULT_CONTENT_TYPE);
		reconciler.setRepairer(dr, IDocument.DEFAULT_CONTENT_TYPE);

		NonRuleBasedDamagerRepairer ndr =
			new NonRuleBasedDamagerRepairer(
				new TextAttribute(
				colorManager.getColor(IColorConstants.COMMENT)));
		reconciler.setDamager(ndr, PartitionScanner.COMMENT);
		reconciler.setRepairer(ndr, PartitionScanner.COMMENT);

		return reconciler;
}

private ITokenScanner getScanner() {
		RuleBasedScanner scanner= new NumberKeyScanner();
		return scanner;
}

public IReconciler getReconciler(ISourceViewer sourceViewer) {
		ProgrammReconcilingStrategy strategy=
new ProgrammReconcilingStrategy(fEditor);
		MonoReconciler reconciler= new MonoReconciler(strategy, false);
		reconciler.setProgressMonitor(new NullProgressMonitor());
		reconciler.setDelay(500);

		return reconciler;
}

public IContentAssistant getContentAssistant(
ISourceViewer sourceViewer) {
		ContentAssistant assistant= new ContentAssistant();

		IContentAssistProcessor processor=
new ProgrammCompletionProcessor(fEditor);
		assistant.setContentAssistProcessor(processor,
IDocument.DEFAULT_CONTENT_TYPE);

		assistant.setContextInformationPopupOrientation(
IContentAssistant.CONTEXT_INFO_ABOVE);
		assistant.setInformationControlCreator(
getInformationControlCreator(sourceViewer));
		assistant.enableAutoInsert(true);

		return assistant;
}
}
```

Anhang 2

A

NumberKeyScanner.java

```
package com.starfinanz.smob.ide.editor.syntax;

import org.eclipse.jface.text.TextAttribute;
import org.eclipse.jface.text.rules.IRule;
import org.eclipse.jface.text.rules.IToken;
import org.eclipse.jface.text.rules.RuleBasedScanner;
import org.eclipse.jface.text.rules.Token;
import org.eclipse.jface.text.rules.WordRule;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Display;


public class NumberKeyScanner extends RuleBasedScanner {

        public NumberKeyScanner() {
                //---
                this.fDefaultReturnToken          =          new          Token(new
TextAttribute(Display.getCurrent()
                                .getSystemColor(SWT.COLOR_BLACK)));

                IRule[] rules= new IRule[3];
                rules[0]= createKeyWordRule();
                rules[1]= createVariableRule();
                rules[2]= createNumberRule();

                setRules(rules);
        }

        private IRule createNumberRule() {
                IToken red = new Token(new TextAttribute(Display.getCurrent()
                                .getSystemColor(SWT.COLOR_RED)));
                WordRule numberwordRule = new WordRule(new NumberDetecter(),red);
                return numberwordRule;
        }

        private IRule createKeyWordRule() {
                IToken blue = new Token(new TextAttribute(Display.getCurrent()
                                .getSystemColor(SWT.COLOR_BLUE)));
                IToken gray = new Token(new TextAttribute(Display.getCurrent()
                                .getSystemColor(SWT.COLOR_DARK_GRAY)));
                KeyRule keywordRule = new KeyWordRule(new KeyWordDetector(),blue,gray
        return keywordRule;

        }

        private IRule createVariableRule() {
                IToken pink = new Token(new TextAttribute(Display.getCurrent()
                                .getSystemColor(SWT.COLOR_MAGENTA)));
                VariableRule vRule = new VariableRule(new VariableDetector(),pink);
                return vRule;
        }


}
```

Anhang 3

LanguagesPreferencePage.java

```
package com.starfinanz.smob.ide.start;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.preference.PreferencePage;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.FileDialog;
import org.eclipse.swt.widgets.Label;
import org.eclipse.swt.widgets.List;
import org.eclipse.swt.widgets.Text;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.IWorkbenchPreferencePage;

public class LanguagesPreferencePage extends PreferencePage
                                     implements IWorkbenchPreferencePage {
        String nameEingabedatei = "";
        String nameAusgabedatei = "";
        String zeile;
        File eingabedatei;
        File ausgabedatei;
        FileReader fr;
        FileWriter fw;
        BufferedReader br;
        BufferedWriter bw;
        String newline = System.getProperty("line.separator");
        Composite parent;
private HashSet<String> reservedWords = new HashSet<String>();
        private String[] keywords = null;
        //The list that displays the current bad words
        private List badWordList;
        //The newEntryText is the text where new bad words are specified
        private Text newEntryText;
        /*
         * @see PreferencePage#createContents(Composite)
         */
        protected Control createContents(Composite parent) {
                Composite entryTable = new Composite(parent, SWT.NULL);
                this.parent = parent;

                //Create a data that takes up the extra space in the dialog .
                GridData data = new GridData(SWT.FILL);
                data.grabExcessHorizontalSpace = true;
                entryTable.setLayoutData(data);
                GridLayout layout = new GridLayout();
```

```
		entryTable.setLayout(layout);
		//Add in a dummy label for spacing
		new Label(entryTable,SWT.NONE);
		badWordList = new List(entryTable, SWT.HORIZONTAL | SWT.V_SCROLL );
		badWordList.setItems(Activator.getDefault().getLanguagePreference());
		//Create a data that takes up the extra space in the dialog
			and spans both columns.
		data = new GridData(GridData.FILL_BOTH);
		badWordList.setLayoutData(data);

		Composite buttonComposite = new Composite(entryTable,SWT.NULL);

		GridLayout buttonLayout = new GridLayout();
		buttonLayout.numColumns = 2;
		buttonComposite.setLayout(buttonLayout);
		//Create a data that takes up the extra space in the dialog
			and spans both columns.
		data = new GridData(GridData.FILL | GridData.BEGINNING);
		buttonComposite.setLayoutData(data);

		Button addButton = new Button(buttonComposite, SWT.PUSH | SWT.CENTER);
		addButton.setText("Durchsuchen"); //$NON-NLS-1$
		addButton.addSelectionListener(new SelectionAdapter() {
			public void widgetSelected(SelectionEvent event) {
				badWordList.add(newEntryText.getText(),
				badWordList.getItemCount());
				newEntryText.setText(languageFile());
				LanguagePattern(newEntryText.getText());
				copySetToList();
			}
		});
		newEntryText = new Text(buttonComposite, SWT.BORDER);
		//Create a data that takes up the extra space in the dialog .
		data = new GridData(GridData.FILL_HORIZONTAL);
		data.grabExcessHorizontalSpace = true;
		newEntryText.setLayoutData(data);
		return entryTable;
	}
	/*
	 * @see IWorkbenchPreferencePage#init(IWorkbench)
	 */
	public void init(IWorkbench workbench) {
		//Initialize the preference store we wish to use
		setPreferenceStore(Activator.getDefault().getPreferenceStore());
	}
	/**
	 * Performs special processing when this page's Restore Defaults
* button has been pressed.
	 * Sets the contents of the nameEntry field to
	 * be the default
	 */
	protected void performDefaults() {

	badWordList.setItems(Activator.getDefault().getDefaulLanguagePreference());
		Activator.getDefault().setLanguagesPreference(badWordList.getItems());
	}
    /**
     * Performs special processing when this page's Apply button has been pressed.
     * This is a framework hook method for sublcasses to do special things when
     * the Apply button has been pressed.
     * The default implementation of this framework method simply calls
     */
	protected void performApply() {
		Activator.getDefault().setLanguagesPreference(this.keywords);
		badWordList.setItems(Activator.getDefault().getLanguagePreference());
	}
	/**
	 * Method declared on IPreferencePage. Save the
	 * author name to the preference store.
	 */
```

```java
	public boolean performOk() {
		if(this.keywords != null) {
			Activator.getDefault().setLanguagesPreference(this.keywords);
		}
		return super.performOk();
	}
	public void selectionChanged(IAction action, ISelection selection) {}
	public String languageFile() {
		FileDialog dialog= new FileDialog(this.parent.getShell(), SWT.OPEN);
		dialog.setText("Öffnen"); //$NON-NLS-1$
		String path= dialog.open();
		if (path != null && path.length() > 0)
			return path;
		return null;
	}
	public String[] copySetToList() {

		int i = 0;
		Iterator it = this.reservedWords.iterator();
		String next = null;

		this.keywords = new String[this.reservedWords.size()];
		while(it.hasNext()) {
			next = it.next().toString();
				if( next != null) {
					this.keywords[i] = next;
				}
			i++;
		}
		return this.keywords;
	}
	private void LanguagePattern(String file) {
		Pattern p = Pattern.compile("\"([^\"]*)\"[!^]");
		nameEingabedatei = file;
		nameAusgabedatei = nameEingabedatei.substring(
			nameEingabedatei.lastIndexOf("/")+1,
				nameEingabedatei.lastIndexOf(".")) + ".keys";
		try {
			eingabedatei = new File(nameEingabedatei);
			ausgabedatei = new File(nameAusgabedatei);
			fr = new FileReader(eingabedatei);
			fw = new FileWriter(ausgabedatei);
			br = new BufferedReader(fr);
			bw = new BufferedWriter(fw);
			while ((zeile = br.readLine()) != null) {
				for( String r : findMatches(p, zeile) )
				{
					if (!reservedWords.contains(r)) {
						reservedWords.add(r);
					}
				}
			}
			System.out.println(reservedWords);
			br.close();
			bw.close();
		} catch (ArrayIndexOutOfBoundsException aioobe) {
			System.out.println("Aufruf mit");
		} catch (FileNotFoundException fnfe) {
			System.out.println("Habe gefangen: " + fnfe);
		} catch (IOException ioe) {
			System.out.println("Habe gefangen: " + ioe);
		}
	}
	public static ArrayList<String> findMatches( Pattern p, CharSequence s){
		ArrayList<String> results = new ArrayList<String>();
		for ( Matcher m = p.matcher(s); m.find(); )
			results.add( m.group(1));
		return results;
	}
}
```

Anhang 3

A

ProgrammElement.java

```
package com.starfinanz.smob.ide.model;

public abstract class ProgrammElement {
	protected static ProgrammElement[] NO_CHILDREN = new ProgrammElement[0];

	private ProgrammElement fParent;
	private String fName;
	private int fOffset;
	private int fLength;

	ProgrammElement(ProgrammElement parent, String name, int offset, int length) {
		fParent= parent;
		fName= name;
		fOffset= offset;
		fLength= length;
	}

	public ProgrammElement getParent() {
		return fParent;
	}

	public abstract ProgrammElement[] getChildren();

	public String getName() {
		return fName;
	}

	public int getOffset() {
		return fOffset;
	}

	public int getLength() {
		return fLength;
	}
}
```

B

Programm.java

```
package com.starfinanz.smob.ide.model;

import java.util.ArrayList;

public class Programm extends ProgrammElement {

	private SubroutinesSection fSubroutinesSection;
	private VariablesSection fVariablesSection;

	Programm() {
		super(null, null, -1, -1);
	}

	public SubroutinesSection getSubroutinesSection() {
		return fSubroutinesSection;
	}

	void setSubroutinesSection(SubroutinesSection subroutinesSection) {
		fSubroutinesSection= subroutinesSection;
	}

	public VariablesSection getVariablesSection() {
		return fVariablesSection;
	}

	void setVariablesSection(VariablesSection variablesSection) {
		fVariablesSection= variablesSection;
	}

	public Variable[] getVariables() {
		return getVariablesSection() != null ?
getVariablesSection().getVariables() : null;
	}

	public Subroutine[] getSubroutines() {
		return getSubroutinesSection() != null ?
getSubroutinesSection().getSubroutines() : null;
	}

	public ProgrammElement[] getChildren() {
		ArrayList<ProgrammElement> result= new ArrayList<ProgrammElement>(2);
		if (fSubroutinesSection != null)
			result.add(fSubroutinesSection);
		if (fVariablesSection != null)
			result.add(fVariablesSection);
		return          (ProgrammElement[])          result.toArray(new
ProgrammElement[result.size()]);
	}
}
```

C

ProgrammParser.java

```
package com.starfinanz.smob.ide.model;

import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;
import org.eclipse.jface.text.BadLocationException;
import org.eclipse.jface.text.IDocument;
import org.eclipse.jface.text.IRegion;

public class ProgrammParser {
	private IDocument fDocument;
	/**
	 * Next line to read.
	 */
	private int fLine;
	private int fLineCount;
	public Programm parse(IDocument document) {
		try {
			fDocument= document;
			fLine= 0;
			fLineCount= fDocument.getNumberOfLines();
			return parseProgramm();
		} catch (BadLocationException e) {
			e.printStackTrace();
			return null;
		}
	}
	private Programm parseProgramm() throws BadLocationException {
		Programm programm= new Programm();
		fLine= 0;
		fLineCount= fDocument.getNumberOfLines();
		IRegion region= fDocument.getLineInformation(fLine);
		if (programm.getSubroutinesSection() == null) {
			Subroutine[] subroutines= parseSubroutines(programm);
			SubroutinesSection section=
new SubroutinesSection(programm, region.getOffset(),
fDocument.getLineOffset(fLine - 1) - region.getOffset());
			section.setSubroutines(subroutines);
			programm.setSubroutinesSection(section);
		}
		fLine= 0;
		fLineCount= fDocument.getNumberOfLines();
		region= fDocument.getLineInformation(fLine);

		if (programm.getVariablesSection() == null) {
			Variable[] variables= parseVariables(programm);
			VariablesSection section=
new VariablesSection(programm, region.getOffset(),
fDocument.getLineOffset(fLine - 1) - region.getOffset());
			section.setSteps(variables);
			programm.setVariablesSection(section);
		}
		return programm;
	}
```

```
private Subroutine[] parseSubroutines(Programm programm) throws BadLocationException {
		List<Subroutine> subroutines= new ArrayList<Subroutine>();
		int offset= -1;
		int length= -1;
		String name= "";
		while (fLine < fLineCount) {
			IRegion region= fDocument.getLineInformation(fLine);
			String		text=		fDocument.get(region.getOffset(),
region.getLength());
			if (text.startsWith(SubroutinesSection.TITLE)) {
				if (offset != -1) {
					subroutines.add(
new Subroutine(programm, offset, length, name));
				}
				name= parseSubroutineName(text);
				offset= region.getOffset();
				length= region.getLength();
			} else if (offset != -1 && text.trim().length() > 0) {
				length= region.getOffset() + region.getLength() - offset;
			} else{
				// skip line
			}
			fLine++;
		}
		if (offset != -1) {
			subroutines.add(new Subroutine(programm, offset, length, name));
		}
		return			(Subroutine[])			subroutines.toArray(new
Subroutine[subroutines.size()]);
	}
	private String parseSubroutineName(String text) {
		String offs = text.substring(SubroutinesSection.TITLE.length() + 1);
		StringTokenizer tok= new StringTokenizer(offs, "- (:\t\n\r\f");
		if (tok.hasMoreTokens())
			return tok.nextToken();
		else
			return "";
	}
	private Variable[] parseVariables(Programm programm) throws BadLocationException
{
		ArrayList<Variable> tVariables= new ArrayList<Variable>();
		int offset = 0;
		int wLenght = -1;
		int indexDim = -1;
		String tWord = "";
		StringTokenizer tok = null;
		char tChar ;
		while (fLine < fLineCount) {
			IRegion region = fDocument.getLineInformation(fLine);
			String		text		=		fDocument.get(region.getOffset(),
region.getLength());
			offset = region.getOffset();
			indexDim = text.indexOf(VariablesSection.TITLE_FELD);
			if (text.startsWith(SubroutinesSection.TITLE_PROG)) {
				tok = new StringTokenizer(text);
				wLenght = -1;
				while (tok.hasMoreTokens()) {
					tWord = tok.nextToken();
					offset += wLenght + 1;
					wLenght = tWord.length();
					tChar = tWord.charAt(tWord.length() - 1 );
					if (tChar == '$' || tChar == '%') {
						tWord  = tWord.substring(0, tWord.length() -
1);
						tVariables.add(new		Variable(programm,
offset,
tWord.length(), tWord));
					}
				}
			}
```

```
				else if( indexDim != -1) {
					tok = new StringTokenizer(text, "( )\t\n");
					offset += indexDim;
					wLenght = -1;
					while (tok.hasMoreTokens()) {
						tWord = tok.nextToken();
						offset += wLenght + 1;
						wLenght = tWord.length();
						tChar = tWord.charAt(tWord.length() - 1 );
						if (tChar == '$' || tChar == '%') {
							tWord  = tWord.substring(0,
											tWord.length() - 1);
							tVariables.add(new Variable(programm,
									offset,tWord.length(), tWord));
						}
					}
				}
				else {
					offset = region.getOffset();
				}
				fLine++;
			}
			return (Variable[]) tVariables.toArray(new Variable[tVariables.size()]);
		}
	}
```

D

ProgrammReconcilingStrategy.java

```
package com.starfinanz.smob.ide.editor;

import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.jface.text.IDocument;
import org.eclipse.jface.text.IRegion;
import org.eclipse.jface.text.reconciler.DirtyRegion;
import org.eclipse.jface.text.reconciler.IReconcilingStrategy;
import org.eclipse.jface.text.reconciler.IReconcilingStrategyExtension;
import org.eclipse.swt.widgets.Shell;

import com.starfinanz.smob.ide.model.Programm;
import com.starfinanz.smob.ide.model.ProgrammParser;


public class ProgrammReconcilingStrategy implements IReconcilingStrategy,
IReconcilingStrategyExtension {

        private SmobEditor fEditor;
        private IDocument fDocument;
        private IProgressMonitor fProgressMonitor;
        private ProgrammParser fParser;
        private ProgrammFoldingStructureProvider fFoldingStructureProvider;
        public ProgrammReconcilingStrategy(SmobEditor editor) {
                fEditor= editor;
                fParser= new ProgrammParser();
                fFoldingStructureProvider= new ProgrammFoldingStructureProvider(editor);
        }
        public void setDocument(IDocument document) {
                fDocument= document;
                fFoldingStructureProvider.setDocument(fDocument);
        }
        public void setProgressMonitor(IProgressMonitor monitor) {
                fProgressMonitor= monitor;
                fFoldingStructureProvider.setProgressMonitor(fProgressMonitor);
        }
public void reconcile(DirtyRegion dirtyRegion, IRegion subRegion) {
                reconcile();
        }
        public void reconcile(IRegion partition) {
                reconcile();
        }
        public void initialReconcile() {
                reconcile();
        }
        private void reconcile() {
                final Programm programm= fParser.parse(fDocument);
                if (programm == null)
                        return;

                Shell shell= fEditor.getSite().getShell();
                if (shell == null || shell.isDisposed())
                        return;

                shell.getDisplay().asyncExec(new Runnable() {
                        public void run() {
                                fEditor.setProgramm(programm);
                        }
                });
                fFoldingStructureProvider.updateFoldingRegions(programm);
        }
}
```

E

ProgrammOutlinePage.java

```
package com.starfinanz.smob.ide.outline;
import java.net.MalformedURLException;
import java.net.URL;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.viewers.AbstractTreeViewer;
import org.eclipse.jface.viewers.ISelectionChangedListener;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.viewers.ITreeContentProvider;
import org.eclipse.jface.viewers.LabelProvider;
import org.eclipse.jface.viewers.SelectionChangedEvent;
import org.eclipse.jface.viewers.TreeViewer;
import org.eclipse.jface.viewers.Viewer;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.views.contentoutline.ContentOutlinePage;
import com.starfinanz.smob.ide.editor.SmobEditor;
import com.starfinanz.smob.ide.model.Programm;
import com.starfinanz.smob.ide.model.ProgrammElement;
import com.starfinanz.smob.ide.model.Subroutine;
import com.starfinanz.smob.ide.model.SubroutinesSection;
import com.starfinanz.smob.ide.model.Variable;
import com.starfinanz.smob.ide.model.VariablesSection;
import com.starfinanz.smob.ide.start.Activator;

public class ProgrammOutlinePage extends ContentOutlinePage {

        private SmobEditor fEditor;

        private static class ProgrammContentProvider implements ITreeContentProvider {
                public Object[] getChildren(Object parentElement) {
                        return ((ProgrammElement) parentElement).getChildren();
                }
                public Object getParent(Object element) {
                        return ((ProgrammElement) element).getParent();
                }
                public boolean hasChildren(Object element) {
                        Object[] children= getChildren(element);
                        return children != null && children.length != 0;
                }
                public Object[] getElements(Object inputElement) {
                        return getChildren(inputElement);
                }
                public void dispose() {
                        // do nothing
                }
                public void inputChanged(Viewer viewer, Object oldInput, Object newInput)
                         {      // do nothing  }
        }
        private static class ProgrammLabelProvider extends LabelProvider {
                private static URL fgIconBaseURL =
Activator.getDefault().getBundle().getEntry("/icons/");

                private final Image fVariablesSectionIcon = createImage("variables.gif");
                private final Image fVariableIcon = createImage("variable.gif");
                private         final           Image           fSubroutineSectionIcon          =
createImage("subroutines.gif");
                private final Image fSubroutineIcon = createImage("subroutine.gif");
                public static Image createImage(String icon) {
                        try {
                                ImageDescriptor id =
ImageDescriptor.createFromURL(new URL(fgIconBaseURL, icon));
                                return id.createImage();
                        } catch (MalformedURLException e) {
```

```
                        // no icon ...
                }
                return null;
        }
        public String getText(Object element) {
                return ((ProgrammElement) element).getName();
        }
                        public Image getImage(Object element) {
                if (element instanceof VariablesSection)
                        return fVariablesSectionIcon;
                else if (element instanceof SubroutinesSection)
                        return fSubroutineSectionIcon;
                else if (element instanceof Variable)
                        return fVariableIcon;
                else if (element instanceof Subroutine)
                        return fSubroutineIcon;
                return super.getImage(element);
        }
                        public void dispose() {
                super.dispose();
                if (fVariablesSectionIcon != null)
                        fVariablesSectionIcon.dispose();
                if (fVariableIcon != null)
                        fVariableIcon.dispose();
                if (fSubroutineSectionIcon != null)
                        fSubroutineSectionIcon.dispose();
                if (fSubroutineIcon != null)
                        fSubroutineIcon.dispose();
        }
    }
        public ProgrammOutlinePage(SmobEditor editor) {
        fEditor= editor;
    }
    public void createControl(Composite parent) {
        super.createControl(parent);
        TreeViewer treeViewer= getTreeViewer();
        treeViewer.setLabelProvider(new ProgrammLabelProvider());
        treeViewer.setContentProvider(new ProgrammContentProvider());
        treeViewer.setAutoExpandLevel(AbstractTreeViewer.ALL_LEVELS);
        treeViewer.addSelectionChangedListener(new ISelectionChangedListener() {
                public void selectionChanged(SelectionChangedEvent event) {
                        if (! (event.getSelection()
                                        instanceof IStructuredSelection))
                                return;
                        IStructuredSelection selection= (IStructuredSelection)
                                                event.getSelection();
                        if (selection.size() != 1)
                                return;
                        Object element= selection.getFirstElement();
                        if (! (element instanceof ProgrammElement))
                                return;
                        ProgrammElement ProgrammElement= (ProgrammElement) element;
                        fEditor.selectAndReveal(ProgrammElement.getOffset(),
                                                        ProgrammElement.getLength());
                }
        });
        setProgramm(fEditor.getProgramm());
    }
    public void setProgramm(Programm programm) {
        getTreeViewer().setInput(programm);
    }
    public void dispose() {
        super.dispose();
        fEditor.outlinePageClosed();
        fEditor= null;
    }
}
```

E

ProgrammCompletionProcessor.java

```java
package com.starfinanz.smob.ide.editor.assist;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import org.eclipse.jface.resource.ImageDescriptor;
import org.eclipse.jface.resource.ImageRegistry;
import org.eclipse.jface.text.BadLocationException;
import org.eclipse.jface.text.IDocument;
import org.eclipse.jface.text.IRegion;
import org.eclipse.jface.text.ITextSelection;
import org.eclipse.jface.text.ITextViewer;
import org.eclipse.jface.text.Region;
import org.eclipse.jface.text.contentassist.CompletionProposal;
import org.eclipse.jface.text.contentassist.ICompletionProposal;
import org.eclipse.jface.text.contentassist.IContentAssistProcessor;
import org.eclipse.jface.text.contentassist.IContextInformation;
import org.eclipse.jface.text.contentassist.IContextInformationValidator;
import org.eclipse.jface.text.templates.DocumentTemplateContext;
import org.eclipse.jface.text.templates.Template;
import org.eclipse.jface.text.templates.TemplateContext;
import org.eclipse.jface.text.templates.TemplateContextType;
import org.eclipse.jface.text.templates.TemplateException;
import org.eclipse.jface.text.templates.TemplateProposal;
import org.eclipse.swt.graphics.Image;
import com.starfinanz.smob.ide.editor.SmobEditor;
import com.starfinanz.smob.ide.model.Programm;
import com.starfinanz.smob.ide.model.Subroutine;
import com.starfinanz.smob.ide.model.Variable;
import com.starfinanz.smob.ide.start.Activator;

public class ProgrammCompletionProcessor implements IContentAssistProcessor {

        private static final String TEMPLATE_ICON= "icons/template.gif";
        private static final String PREPARATION_TEMPLATE_CTX=
                        "org.eclipse.ui.examples.RcpEditor.preparation";
        private static final String SUBROUTINES_TEMPLATE_CTX=
                        "com.starfinanz.smob.ide.editor.subroutines";
        private static final class ProposalComparator implements Comparator {
                public int compare(Object o1, Object o2) {
                        return ((TemplateProposal) o2).getRelevance() -
                                        ((TemplateProposal) o1).getRelevance();
                }
        }
        private static final Comparator fgProposalComparator= new ProposalComparator();
        private final SmobEditor fEditor;
        public ProgrammCompletionProcessor(SmobEditor editor) {
                fEditor= editor;
        }
        public ICompletionProposal[] computeCompletionProposals(
                                        ITextViewer viewer, int offset) {
                Programm programm= fEditor.getProgramm();
                if (programm == null)
                        return null;
                ITextSelection selection= (ITextSelection)
                                viewer.getSelectionProvider().getSelection();
                if (selection.getOffset() != offset)
                        offset= selection.getOffset();
                String prefix= getPrefix(viewer, offset);
                Region region= new Region(offset - prefix.length(),
                                prefix.length() + selection.getLength());
```

```java
		ICompletionProposal[] computeKeysProposals=
			computeKeysProposals(viewer, offset);
		ICompletionProposal[] templateProposals=
			computeTemplateProposals(viewer, region, programm, prefix);
		ICompletionProposal[] ingredientProposals=
			computeSubroutinesProposals(viewer, region, programm, prefix);

		List<ICompletionProposal> result= new ArrayList<ICompletionProposal>();
		result.addAll(Arrays.asList(computeKeysProposals));
		result.addAll(Arrays.asList(ingredientProposals));
		result.addAll(Arrays.asList(templateProposals));
		return            (ICompletionProposal[])            result.toArray(new
ICompletionProposal[result.size()]);
	}
	private ICompletionProposal[] computeSubroutinesProposals(ITextViewer viewer,
				IRegion region, Programm programm, String prefix) {
		Variable[] variables= programm.getVariables();
		if (variables == null || variables.length == 0)
			return new ICompletionProposal[0];

		int offset= region.getOffset();
		Variable first= variables[0];
		if (offset < first.getOffset())
			return new ICompletionProposal[0];

		Subroutine[] ingredients= programm.getSubroutines();
		if (ingredients == null)
			return new ICompletionProposal[0];

		prefix= prefix.toLowerCase();

		ArrayList<CompletionProposal>              proposals=              new
ArrayList<CompletionProposal>();
		for (int i= 0; i < ingredients.length; i++) {
			String ingredient= ingredients[i].getName();
			if (ingredient.toLowerCase().startsWith(prefix))
				proposals.add(new  CompletionProposal(ingredient,  offset,
region.getLength(), ingredient.length()));
		}
		return            (ICompletionProposal[])            proposals.toArray(new
ICompletionProposal[proposals.size()]);
	}

	public ICompletionProposal[] computeKeysProposals(ITextViewer viewer,    int
offset) {
		String[] keys =  Activator.getDefault().getLanguagePreference();
		List<ICompletionProposal> proposals;

		if(keys.length > 0){
			proposals = createProposals(keys, viewer, offset);
		}else{
			keys =  Activator.getDefault().getDefaulLanguagePreference();
			proposals = createProposals(keys, viewer, offset);
		}
		return (ICompletionProposal[]) proposals.toArray(
				new ICompletionProposal[proposals.size()]);
	}
	protected List<ICompletionProposal> createProposals(String[] candidates,
			ITextViewer viewer, int offset) {
		Arrays.sort(candidates);
		List<ICompletionProposal> proposals =
					new ArrayList<ICompletionProposal>();
		String prefix = getPrefix(viewer.getDocument(), offset - 1);
		for (String kw : candidates) {
			if (prefix.length() <= kw.length()) {
				if (kw.substring(0,
				prefix.length()).equalsIgnoreCase(prefix)) {
					proposals.add(createProposal(kw,
						offset - prefix.length(),
							prefix.length()));
```

```
                        }
                }
        }
        return proposals;
}
protected String getPrefix(IDocument doc, int offset) {
        int pos = offset;
        try {
                while (pos >= 0 && Character.isLetter(doc.getChar(pos))) {
                        pos--;
                }
                return doc.get(pos + 1, offset - pos);
        } catch (BadLocationException e) {
                e.printStackTrace();
        }
        return "";
}
protected ICompletionProposal createProposal(String string, int offset,
                int prefix) {
        CompletionProposal p = new CompletionProposal(string, offset, prefix,
                        string.length());
        return p;
}
private TemplateContextType getContextType(Programm programm, int offset) {
        if (programm.getVariablesSection() != null &&
                programm.getVariablesSection().getOffset() < offset)
                return Activator.getDefault().
                getContextTypeRegistry().getContextType(PREPARATION_TEMPLATE_CTX);
        else
                return Activator.getDefault().
                getContextTypeRegistry().getContextType(SUBROUTINES_TEMPLATE_CTX);
}
/**
 * Creates a concrete template context for the given region in the document. This
involves finding out which
 * context type is valid at the given location, and then creating a context of
this type. The default implementation
 * returns a <code>DocumentTemplateContext</code> for the context type at the
given location.
 *
 * @param viewer the viewer for which the context is created
 * @param region the region into <code>document</code> for which the context is
created
 * @return a template context that can handle template insertion at the given
location, or <code>null</code>
 */
private TemplateContext createContext(ITextViewer viewer, IRegion region ,
Programm programm) {
        TemplateContextType contextType= getContextType(programm,
                        region.getOffset());
        if (contextType != null) {
                IDocument document= viewer.getDocument();
                return new DocumentTemplateContext(contextType,
                        document, region.getOffset(), region.getLength());
        }
        return null;
}
@SuppressWarnings("unchecked")
private ICompletionProposal[] computeTemplateProposals(ITextViewer viewer,
                IRegion region, Programm programm, String prefix) {
        TemplateContext context= createContext(viewer, region, programm);
        if (context == null)
                return new ICompletionProposal[0];
        ITextSelection selection= (ITextSelection)
                viewer.getSelectionProvider().getSelection();
        context.setVariable("selection", selection.getText());
        // name of the selection variables {line, word}_selection
        String id= context.getContextType().getId();
        Template[]                                              templates=
Activator.getDefault().getTemplateStore().getTemplates(id);
```

```java
		List<TemplateProposal> matches= new ArrayList<TemplateProposal>();
		for (int i= 0; i < templates.length; i++) {
			Template template= templates[i];
			try {
				context.getContextType().validate(template.getPattern());
			} catch (TemplateException e) {
				continue;
			}
			int relevance= getRelevance(template, prefix);
			if (relevance > 0) {
				matches.add(new TemplateProposal(template, context, region,
getImage(template), relevance));
			}
		}
		Collections.sort(matches, fgProposalComparator);
		return		(ICompletionProposal[])		matches.toArray(new
ICompletionProposal[matches.size()]);
	}
	/**
	 * Returns the relevance of a template given a prefix. The default
	 * implementation returns a number greater than zero if the template name
	 * starts with the prefix, and zero otherwise.
	 *
	 * @param template the template to compute the relevance for
	 * @param prefix the prefix after which content assist was requested
	 * @return the relevance of <code>template</code>
	 * @see #getPrefix(ITextViewer, int)
	 */
	private int getRelevance(Template template, String prefix) {
		if (template.getName().startsWith(prefix))
			return 90;
		return 0;
	}
	private String getPrefix(ITextViewer viewer, int offset) {
		int i= offset;
		IDocument document= viewer.getDocument();
		if (i > document.getLength())
			return "";
		try {
			while (i > 0) {
				char ch= document.getChar(i - 1);
				if (! Character.isLetterOrDigit(ch))
					break;
				i--;
			}
			return document.get(i, offset - i);
		} catch (BadLocationException e) {
			return "";
		}
	}
	/**
	 * Always return the default image.
	 */
	private Image getImage(Template template) {
		ImageRegistry registry= Activator.getDefault().getImageRegistry();
		Image image= registry.get(TEMPLATE_ICON);
		if (image == null) {
			ImageDescriptor desc= Activator.imageDescriptorFromPlugin(
	"com.starfinanz.smob.ide.editors.SmobEditor", TEMPLATE_ICON);
			registry.put(TEMPLATE_ICON, desc);
			image= registry.get(TEMPLATE_ICON);
		}
		return image;
	}
	public IContextInformation[] computeContextInformation(
					ITextViewer viewer, int offset) {
		return null;
	}
	public char[] getCompletionProposalAutoActivationCharacters() {
		return null;
```

```
	}
	public char[] getContextInformationAutoActivationCharacters() {
		return null;
	}
	public String getErrorMessage() {
		return null;
	}
	public IContextInformationValidator getContextInformationValidator() {
		return null;
	}
}
```

Anhang 4

A.

SmobDataLoader.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package org.vln.smobeditor.filetype;

import java.io.IOException;
import org.openide.filesystems.FileObject;
import org.openide.loaders.DataObjectExistsException;
import org.openide.loaders.MultiDataObject;
import org.openide.loaders.UniFileLoader;
import org.openide.util.NbBundle;

public class SmobDataLoader extends UniFileLoader {

    public static final String REQUIRED_MIME = "text/x-smob";
    private static final long serialVersionUID = 1L;
    public SmobDataLoader() {
        super("org.vln.smobeditor.filetype.SmobDataObject");
    }
    @Override
    protected String defaultDisplayName() {
        return NbBundle.getMessage(SmobDataLoader.class, "LBL_Smob_loader_name");
    }
    @Override
    protected void initialize() {
        super.initialize();
        getExtensions().addMimeType(REQUIRED_MIME);
    }
    protected    MultiDataObject    createMultiObject(FileObject    primaryFile)    throws
DataObjectExistsException, IOException {
        return new SmobDataObject(primaryFile, this);
    }
    @Override
    protected String actionsContext() {
        return "Loaders/" + REQUIRED_MIME + "/Actions";
    }
}
```

B.

SmobResolver.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
To change this template, choose Tools | Templates
and open the template in the editor.
-->
<!DOCTYPE    MIME-resolver    PUBLIC    "-//NetBeans//DTD    MIME    Resolver    1.0//EN"
"http://www.netbeans.org/dtds/mime-resolver-1_0.dtd">
<MIME-resolver>
    <file>
        <ext name="bas"/>
        <resolver mime="text/x-smob"/>
    </file>
</MIME-resolver>
```

C.

SmobDataLoaderBeanInfo.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package org.vln.smobeditor.filetype;

import java.awt.Image;
import java.beans.BeanInfo;
import java.beans.IntrospectionException;
import java.beans.Introspector;
import java.beans.SimpleBeanInfo;
import org.openide.loaders.UniFileLoader;
import org.openide.util.Utilities;

public class SmobDataLoaderBeanInfo extends SimpleBeanInfo {
    @Override
    public BeanInfo[] getAdditionalBeanInfo() {
        try {
            return new BeanInfo[]{Introspector.
getBeanInfo(UniFileLoader.class)};
        } catch (IntrospectionException e) {
            throw new AssertionError(e);
        }
    }
    @Override
    public Image getIcon(int type) {
        return super.getIcon(type);
    }}
```

D.

SmobDataObject.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package org.vln.smobeditor.filetype;
import java.io.IOException;
import org.openide.filesystems.FileObject;
import org.openide.loaders.DataObjectExistsException;
import org.openide.loaders.MultiDataObject;
import org.openide.nodes.CookieSet;
import org.openide.nodes.Node;
import org.openide.util.Lookup;
import org.openide.text.DataEditorSupport;
public class SmobDataObject extends MultiDataObject {

    public SmobDataObject(FileObject pf, SmobDataLoader loader) throws
DataObjectExistsException, IOException {
        super(pf, loader);
        CookieSet cookies = getCookieSet();
        cookies.add((Node.Cookie) DataEditorSupport.create(this, getPrimaryEntry(),
cookies));
    }
    @Override
    protected Node createNodeDelegate() {
        return new SmobDataNode(this, getLookup());
    }
    @Override
    public Lookup getLookup() {
        return getCookieSet().getLookup();
    }
}
```

E.

SmobDataNode.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package org.vln.smobeditor.filetype;

import org.openide.loaders.DataNode;
import org.openide.nodes.Children;
import org.openide.util.Lookup;

public class SmobDataNode extends DataNode {

    private static final String IMAGE_ICON_BASE = "SET/PATH/TO/ICON/HERE";

    public SmobDataNode(SmobDataObject obj) {
        super(obj, Children.LEAF);
    }
    SmobDataNode(SmobDataObject obj, Lookup lookup) {
        super(obj, Children.LEAF, lookup);
    }

}
```

Anhang 5

A.

SmobEditorKit.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package org.vln.smobeditor;

import javax.swing.text.Document;
import org.netbeans.editor.Syntax;
import org.netbeans.editor.ext.plain.PlainSyntax;
import org.netbeans.modules.editor.NbEditorKit;
import org.vln.smobeditor.syntax.SmobSyntax;


/**
 *
 * @author vln
 */
public class SmobEditorKit extends NbEditorKit{

    public SmobEditorKit(){
        super();
    }

    @Override
    public String getContentType(){
        return "text/x-smob";
    }

    @Override
    public Syntax createSyntax(Document document)
    {
      return new SmobSyntax();
    }
}
```

B.

Installer.java

```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package org.vln.smobeditor;

import org.openide.modules.ModuleInstall;
import org.netbeans.editor.LocaleSupport;
import org.netbeans.editor.Settings;
import org.openide.modules.ModuleInstall;
import org.openide.util.NbBundle;
import org.vln.smobeditor.syntax.SmobSettingsInitializer;
/**
 * Manages a module's lifecycle. Remember that an installer is optional and
 * often not needed at all.
 */
public class Installer extends ModuleInstall {

    /**
     * Localizer passed to editor.
     */
    private static LocaleSupport.Localizer localizer;

    /**
     * Registers properties editor, installs options and copies settings.
     * Overrides superclass method.
     */
    @Override
    public void restored() {
        addInitializer();
        installOptions();
    }
    /**
     * Uninstalls properties options.
     * And cleans up editor settings copy.
     * Overrides superclass method.
     */
    public void uninstalled() {
        uninstallOptions();
    }
    /**
     * Adds initializer and registers editor kit.
     */
    public void addInitializer() {
        Settings.addInitializer(new SmobSettingsInitializer());
    }
    /**
     * Installs properties editor and print options.
     */
    public void installOptions() {
        // Adds localizer.
        LocaleSupport.addLocalizer(localizer = new LocaleSupport.Localizer() {
            public String getString(String key) {
                return NbBundle.getMessage(Installer.class, key);
            }
        });
    }
    /** Uninstalls properties editor and print options. */
    public void uninstallOptions() {
        // remove localizer
        LocaleSupport.removeLocalizer(localizer);
    }
}
```

Anhang 6

A.

SmobEditorOptions.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package org.vln.smobeditor.options;

import java.util.MissingResourceException;
import org.netbeans.modules.editor.options.BaseOptions;
import org.openide.util.NbBundle;
import org.vln.smobeditor.SmobEditorKit;

/**
 *
 * @author vln
 */
public class SmobEditorOptions extends BaseOptions {

      /** A name for these options */
  public static String SMOB_PREFIX= "smob"; // NOI18N 1
  /** A default constructor */
  public SmobEditorOptions()
  {
    super( SmobEditorKit.class, SMOB_PREFIX );
    // Just for fun: modify a property in the base class ;-)
    setCaretBlinkRate( 450 );
    setStatusBarVisible(true);
  }
  /** A sample property */
  private int mySampleProperty;
  public void setMySampleProperty( int aValue )
  {
    mySampleProperty = aValue;
  }
  public int getMySampleProperty()
  {
    return mySampleProperty;
  }
  /**
   * This is used to retrieve display names to
   *   easy internationalization of options.
   */
  protected String getString(String key)
  {
    try
    {
      return NbBundle.getMessage(SmobEditorOptions.class, key);
    }
    catch (MissingResourceException e)
    {
      return super.getString(key);
    }
  }
}
```

B.

SmobEditorOptions.settings

```
<?xml version="1.0"?>
<!DOCTYPE settings PUBLIC "-//NetBeans//DTD Session settings 1.0//EN"
"http://www.netbeans.org/dtds/sessionsettings-1_0.dtd">
<settings version="1.0">
  <module name="org.vln.smobeditor"/>
  <instanceof class="java.io.Externalizable"/>
  <instanceof class="org.openide.util.SharedClassObject"/>
  <instanceof class="java.beans.beancontext.BeanContextProxy"/>
  <instanceof class="java.io.Serializable"/>
  <instanceof class="org.openide.options.SystemOption"/>
  <instanceof class="org.netbeans.modules.editor.options.OptionSupport"/>
  <instanceof class="org.netbeans.modules.editor.options.BaseOptions"/>
  <instance class="org.vln.smobeditor.options.SmobEditorOptions"/>
</settings>
```

Anhang 7

A.

SmobTokenContext.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package org.vln.smobeditor.syntax;

/**
 *
 * @author vln
 */

import org.netbeans.editor.Utilities;
import org.netbeans.editor.BaseTokenID;
import org.netbeans.editor.TokenContext;
import org.netbeans.editor.TokenContextPath;

public class SmobTokenContext extends TokenContext {

    // Numeric-ids for token categories
    public static final int PROG_ID = 1;
    public static final int STR_ID = 2;
    public static final int VAR_ID = 3;
    public static final int END_OF_LINE_ID = 4;

    // Token-ids
    public static final BaseTokenID PROGRAMM = new BaseTokenID("name", PROG_ID);
    public static final BaseTokenID STRING = new BaseTokenID("colon", STR_ID);
    public static final BaseTokenID VARIABLE = new BaseTokenID("value", VAR_ID);
    public static final BaseTokenID END_OF_LINE =
            new BaseTokenID("end-of-line", END_OF_LINE_ID);

    // Context instance declaration
    public static final SmobTokenContext context = new SmobTokenContext();
    public static final TokenContextPath contextPath = context.getContextPath();

    /**
     * Construct a new ManifestTokenContext
     */
    private SmobTokenContext() {
        super("smob-");

        try {
            addDeclaredTokenIDs();
        } catch (Exception e) {
            Utilities.annotateLoggable(e);
        }
    }
}
```

B.

SmobSyntax.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package org.vln.smobeditor.syntax;
/**
 *
 * @author vln
 */
import org.netbeans.editor.Syntax;
import org.netbeans.editor.TokenID;
import org.openide.ErrorManager;

public class SmobSyntax extends Syntax {

    /**
     * The logger for this class. It can be used for tracing the class activity,
     * logging debug messages, etc.
     */
    private static final ErrorManager LOGGER =
            ErrorManager.getDefault().getInstance("org.vln.smobeditor." +
            "SmobEditor.SmobSyntax");
    /**
     * Used to avoing calling the log() or notify() method if the message
     * wouldn't be loggable anyway.
     */
    private static final boolean LOG =
            LOGGER.isLoggable(ErrorManager.INFORMATIONAL);

    // The states for the lexical analyzer
    private static final int ISI_NAME = 1; // irgendein Wort
    private static final int ISI_NAME_VAR = ISI_NAME + 1; // Variable
    private static final int ISI_ANF_STR = ISI_NAME_VAR + 1; // Anfang des Strings
    private static final int ISI_STR = ISI_ANF_STR + 1; // String-Körper
    private static final int ISI_END_STR = ISI_STR + 1; // Ende des Strings
    @Override
    protected TokenID parseToken() {
        TokenID result = doParseToken();
        if (LOG) {
            LOGGER.log(ErrorManager.INFORMATIONAL, "parseToken: " + result);
        }
        return result;
    }
    private TokenID doParseToken() {
        char actChar;
        while (offset < stopOffset) {
            actChar = buffer[offset];
            switch (state) {
                case INIT:
                    switch (actChar) {
                        case '\"':
                            state = ISI_ANF_STR;
                            break;
                        case ' ':
                            state = INIT;
                            break;
                        case '=':
                            state = INIT;
                            break;
                        case '(':
                            state = INIT;
                            break;
                        case ')':
```

```
                state = INIT;
                break;
            case ',':
                state = INIT;
                break;
            case '\n':
                state = INIT;
                offset++;
                return SmobTokenContext.END_OF_LINE;
            default:
                state = ISI_NAME;
        }
        break;
    case ISI_NAME:
        switch (actChar) {
            case '\"':
                state = ISI_ANF_STR;
            case '\n':
                offset++;
                return SmobTokenContext.END_OF_LINE;
            case '$':
                state = ISI_NAME_VAR;
                break;
            case '%':
                state = ISI_NAME_VAR;
                break;
            default:
                state = ISI_NAME;
        }
        break;
    case ISI_ANF_STR:
        switch (actChar) {
            case '\"':
                state = ISI_END_STR;
                offset++;
                return SmobTokenContext.STRING;
            default:
                state = ISI_STR;
        }
        break;
    case ISI_STR:
        switch (actChar) {
            case '\"':
                state = ISI_END_STR;
                offset++;
                return SmobTokenContext.STRING;
            default:
                state = ISI_STR;
        }
        break;
    case ISI_END_STR:
        switch (actChar) {
            case '\n':
                state = INIT;
                return SmobTokenContext.STRING;
            case '\"':
                state = ISI_ANF_STR;
            default:
                state = INIT;
        }
        break;
    case ISI_NAME_VAR:
        switch (actChar) {
            case '\n':
                state = INIT;
                return SmobTokenContext.VARIABLE;
            case ' ':
                state = INIT;
                return SmobTokenContext.VARIABLE;
            default:
```

```
                        state = ISI_NAME;
                    }
                    break;
            }
            offset++;
        }

        /*
         * At this stage there's no more text in the scanned buffer.
         * It is valid to return a token here only if this is the last
         * buffer (otherwise the token could continue in the next buffer).
         */
        if (lastBuffer) {
            switch (state) {
                case ISI_NAME:
                    state = INIT;
                    return SmobTokenContext.END_OF_LINE;
                case INIT:
                    state = INIT;
                    return SmobTokenContext.VARIABLE;
            }
        }
        return null;
    }
}
```

C.

SmobSettingsInitializer java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package org.vln.smobeditor.syntax;
/**
 *
 * @author vln
 */
import java.awt.Color;
import java.awt.Font;
import java.util.Map;
import org.netbeans.editor.BaseKit;
import org.netbeans.editor.Coloring;
import org.netbeans.editor.Settings;
import org.netbeans.editor.SettingsDefaults;
import org.netbeans.editor.SettingsNames;
import org.netbeans.editor.SettingsUtil;
import org.netbeans.editor.TokenCategory;
import org.netbeans.editor.TokenContext;
import org.netbeans.editor.TokenContextPath;
import org.vln.smobeditor.SmobEditorKit;

public class SmobSettingsInitializer
        extends Settings.AbstractInitializer {

    public static final String NAME =
            "Smob-settings-initializer"; // NOI18N
    /**
     * Constructor
     */
    public SmobSettingsInitializer() {
        super(NAME);
    }
    /**
     * Update map filled with the settings.
     * @param kitClass kit class for which the settings are being updated.
     *   It is always non-null value.
     * @param settingsMap map holding [setting-name, setting-value] pairs.
     *   The map can be empty if this is the first initializer
     *   that updates it or if no previous initializers updated it.
     */
    public void updateSettingsMap(Class kitClass, Map settingsMap) {
        if (kitClass == BaseKit.class) {
            new SmobTokenColoringInitializer().
                    updateSettingsMap(kitClass, settingsMap);
        }

        if (kitClass == SmobEditorKit.class) {
            SettingsUtil.updateListSetting(
                    settingsMap,
                    SettingsNames.TOKEN_CONTEXT_LIST,
                    new TokenContext[]
            { SmobTokenContext.context }
            );
        }
    }

    /**
     * Class for adding syntax coloring to the editor
     */
    /** Properties token coloring initializer. */
```

```java
    private static class SmobTokenColoringInitializer
            extends SettingsUtil.TokenColoringInitializer {

        /** Bold font. */
        private static final Font boldFont =
                SettingsDefaults.defaultFont.deriveFont(Font.BOLD);
        /** Italic font. */
        private static final Font italicFont =
                SettingsDefaults.defaultFont.deriveFont(Font.ITALIC);

        /** Value coloring. */
        private static final Coloring valueColoring =
                new Coloring(null, Color.magenta, null);
        /** Colon coloring. */
        private static final Coloring colonColoring =
                new Coloring(null, Color.green, null);
        /** Empty coloring. */
        private static final Coloring emptyColoring =
                new Coloring(null, null, null);


        /** Constructs PropertiesTokenColoringInitializer. */
        public SmobTokenColoringInitializer() {
            super(SmobTokenContext.context);
        }


        /** Gets token coloring. */
        public Object getTokenColoring(TokenContextPath tokenContextPath,
                TokenCategory tokenIDOrCategory, boolean printingSet) {

            if(!printingSet) {
                int tokenID = tokenIDOrCategory.getNumericID();
                  if(tokenID == SmobTokenContext.VAR_ID) {
                    return valueColoring;
                } else if(tokenID == SmobTokenContext.STR_ID) {
                    return colonColoring;

                }
            } else { // printing set
                return SettingsUtil.defaultPrintColoringEvaluator;
            }

            return null;
        }

    } // End of class SmobTokenColoringInitializer.

}
```

D.

SmobSyntaxTest.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author vln
 */
import java.util.Arrays;
import java.util.Iterator;
import junit.framework.TestCase;
import org.netbeans.editor.Syntax;
import org.netbeans.editor.TokenID;
import junit.framework.TestCase;
import org.vln.smobeditor.syntax.SmobSyntax;
import org.vln.smobeditor.syntax.SmobTokenContext;

public class SmobSyntaxTest extends TestCase {

    public SmobSyntaxTest(String testName) {
        super(testName);
    }

    public void testNextToken() {
        doParse("ChainParameterString$  ,  DebugMode%  frmName$  =  \"frmHello\"",  new
TokenID[] {
            SmobTokenContext.VARIABLE,
            SmobTokenContext.VARIABLE,
            SmobTokenContext.VARIABLE,
            SmobTokenContext.STRING,
        });
    }

    public void doParse(String m, TokenID[] expected) {
        Syntax s = new SmobSyntax();
        s.load(null, m.toCharArray(), 0, m.length(), true, m.length());

        TokenID token = null;
        Iterator i = Arrays.asList(expected).iterator();
        do {
            token = s.nextToken();
            if (token != null) {
                if (!i.hasNext()) {
                    fail("More tokens returned than expected.");
                } else {
                    assertSame("Tokens differ", i.next(), token);
                }
            } else {
                assertFalse("More tokens expected than returned.", i.hasNext());
            }
            System.out.println(token);
        } while (token != null);
    }
}
```

Versicherung über die Selbständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(4) bzw. §25(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort, Datum Unterschrift