



Hausarbeit OSS - Eclipse

Fenn Stefan SS04

FH Augsburg

Dr. Hubert Högl

1 Eclipse

Eclipse ist ein Framework, das als freie IDE (Entwicklungsumgebung) genutzt werden kann. **Eclipse** ist selbst nur die Kernanwendung, die verschieden Plugins lädt, die die eigentliche Funktionalität (z.B. die einer IDE) zur Verfügung stellen. Diese Funktionalität wird RCP¹ genannt. **Eclipse** an sich als auch die Plugins sind vollständig in Java implementiert² und damit auch sprach- und plattformunabhängig.

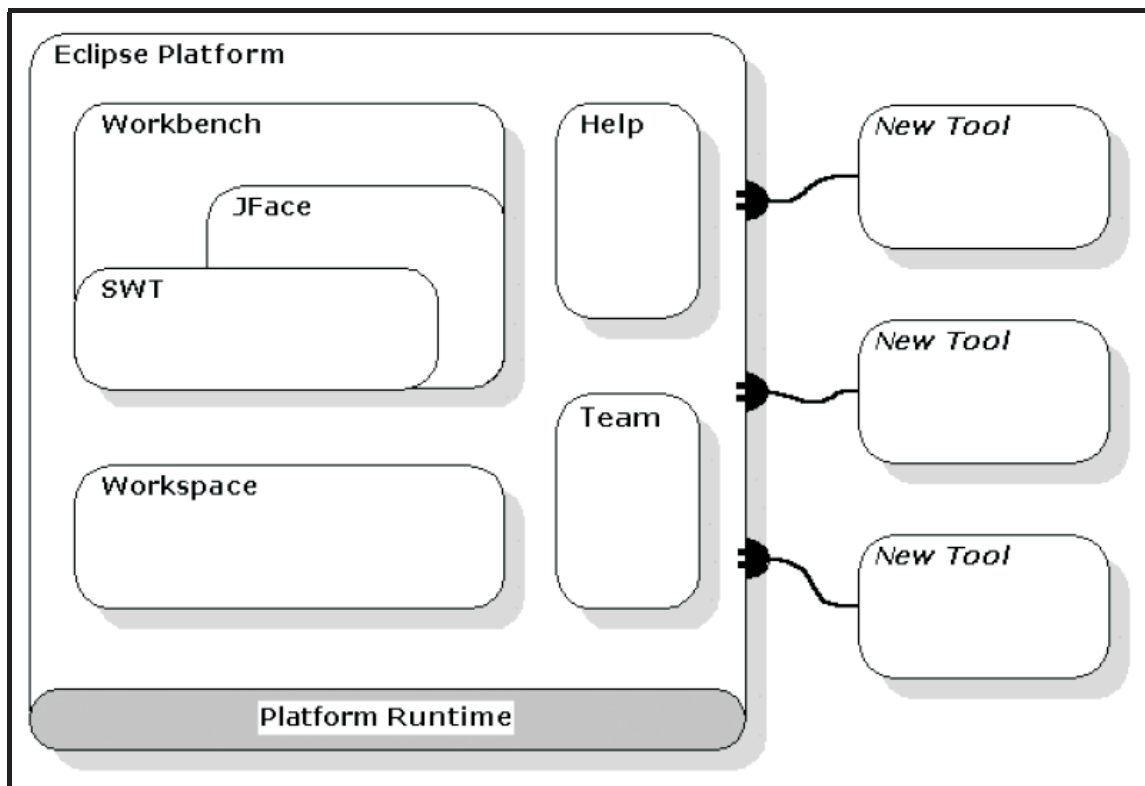


Abbildung 1: **Eclipse** Platform Architektur (aus www.eclipse.org)

Die Besonderheit ist hier, dass die Plugin-Struktur sehr dynamisch und versioniert abläuft. Das heißt vor allem, dass Plugin's auch Teile von anderen Plugin's über so genannte Extension Points benutzen können. Hierdurch wird der Kern sehr schlank und einfach gehalten. Das Prinzip ähnelt daher einer Mikrokern-Architektur bei Betriebssystemen, wo nur die wichtigsten Aufgaben z. B. Prozessumschaltung oder Nachrichtenverarbeitung im Kern läuft und die weiteren Aufgaben über Server-Prozesse dazu geschaltet werden. Dadurch entstehen sehr skalierbare Architekturen, die durch abgeschlossene Module verbunden werden.

Ein Plugin schreibt vor, welche anderen Plugins enthalten sein müssen. Auch die Version wird mit beachtet. Der Plugin-Manager verwaltet alle Plugins und ermöglicht auch das Zurückspringen, falls Probleme durch eine neue Pluginstruktur entstehen würden. Plugins können überall in **Eclipse** eingehängt werden, z. B. als Menüpunkt

¹Rich Client Platform

²bis auf die Anbindungsstellen von SWT

oder auch als eine neue Sicht. Eines der größten Plugins in **Eclipse** ist die Java-Projekt-Sicht. Hier wurden viele Ideen aus dem IBM-Vorgänger VisualAge übernommen und stellen heute eine echte Alternative zu anderen proprietären Entwicklungsumgebungen dar. Es wird auch an IDE's anderer Programmiersprachen gearbeitet wie C++ oder Cobol.

Die Installation eines Plugins verläuft in der Regel so, dass ein Ordner in das Plugin-Verzeichnis hineinkopiert wird. Wenn **Eclipse** neu gestartet wird, meldet der Plugin-Manager, die neu erkannten Plugins und man kann sich entscheiden, ob diese nun eingehängt werden sollen. Ein anderer Weg ist die Online-Installation. Hier wird von der betreffenden Firma, die ein Plugin zur Verfügung stellt, eine URL angegeben, die dann in **Eclipse** benutzt wird, um den Plugin vollständig automatisch herunterzuladen und anschließen zu installieren.

Es wurde ein eigenes Graphical User Interface Framework zur Erstellung der grafischen Oberfläche – das SWT – verwendet. SWT basiert ähnlich wie das AWT auf den nativen GUI-Komponenten und Schnittstellen des jeweiligen Betriebssystems. Der Unterschied zwischen AWT und SWT ist, dass das SWT wesentlich enger an die native Funktionalität gebunden ist und dadurch effizienter ablaufen kann. So muss man sich z. B. für die Entsorgung von Komponenten selber kümmern, dafür läuft aber auch die Applikation schneller und Ressourcen sparer ab.

Die Bezeichnung **Eclipse** ist der englische Begriff für eine Sonnenfinsternis (solar eclipse). Damals wurden in einem Brainstorm-Meeting nach einem Namen mit dem damals populären E-Präfix wie E-Business oder E-Commerce gesucht. Hier fiel der Name E-clipse, der darauf hinweisen soll, dass mit **Eclipse** proprietäre Entwicklungsumgebungen in den Schatten gestellt werden.

Eclipse stellt den Nachfolger für IBM Visual Age dar. Die Entwicklung kostete mehr als 40 Millionen Dollar. Der Quellcode für **Eclipse** wurde dann von IBM als Open-Source freigegeben. Die Verwaltung und Entwicklung von **Eclipse** wurde vom **Eclipse**-Projekt übernommen.

Zum **Eclipse**-Projekt gehören u.a. die folgenden Software-Firmen:

-Borland	-Erickson
-Fujitsu	-Hitachi
-HP	-IBM
-Rational Software	-Red Hat
-Serena	-SAP
-SuSE	-Sybase
-TogetherSoft	-WebGain
-Oracle	

1.1 SWT Grundlagen

Das **Standard Widget Toolkit** bündelt von den verschiedenen Betriebssystemen die graphische Schnittstelle. Anders wie bei Swing wird hier nicht nur das Zeichnen eines Punktes als Aufgabe an das Betriebssystem geschickt, sondern auch die Erzeugung und Verwaltung aller Standardkomponenten. Das SWT gibt es für viele Betriebssysteme und wird bei IBM weiterentwickelt.

Die Anforderungen an das SWT sind eine schnelle Reaktions- und Aufbaugeschwindigkeit und ein Betriebssystem spezifisches Verhalten bei den Standardkomponenten. Dies wird bei SWT dadurch erreicht, dass keine eigenen UI- und Event-Threads

erzeugt werden und man sich um die Zerstörung (dispose) von grafischen Objekten wie Kontrollen, Dialogen bis hin zu einzelnen Farben, selbst kümmern muß. Im Allgemeinen steigt dadurch der zu schreibende Code, aber man gewinnt eben dadurch auch Performance.

1.1.1 Vorbereitungen für SWT

Um ein SWT-Projekt zu erstellen müssen Sie zuvor eine SWT-Version für ihre Plattform herunterladen. Diese besteht aus den JNI-Stubs wie z. B. swt-win32-2134.dll und dem Jar-Archiv swt.jar. Des weiteren sollte man bei Veröffentlichung die Lizenzvereinbarung mit zu Verfügung stellen.

Kopieren Sie die zuvor genannten Komponenten unter Ihren Projekt in ein neues Verzeichnis „swt“. Hier nun ein kurzer Code, der ihnen eine MessageBox ausgeben sollte.

HelloWorld . java

```
import org.eclipse.swt.widgets.*;

public class HelloWorld {

    public static void main ( String [] args ) {
        MessageBox mb = new MessageBox ( new Shell ());
        mb . setMessage ( „Hello SWT-World!“ );
        mb . open ();
    } //main

} //HelloWorld
```

Gestartet wird das Programm mit

```
java -cp ./;./swt.jar; -Djava.library.path=./swt HelloWorld
```

Hier wird das SWT relativ zum Programm geladen. Wenn mehrere Programme auf eine SWT-Version zugreifen sollen, kann natürlich der Pfad auch absolut angegeben werden. Auch beim kompilieren muß das Archiv swt.jar im Classpath angegeben werden. Fall Sie das JVM Argument **-Djava.library.path= . . .** vergessen, so wird folgende Nachricht ausgegeben.

```
java.lang.UnsatisfiedLinkError: no swt-win32-2134 in java.library.path
...
```

Das sollten die größten Hürden für den Umstieg von Swing oder AWT auf SWT gewesen sein.

1.1.2 Eine einfache Applikation erstellen

Um eine Anwendung in SWT zu schreiben sind üblicherweise die folgenden Schritte zu machen. Falls Sie schnell eine kleine Anwendung schreiben möchten, kann ich nur **Jigloo GUI Builder von Cloudgarden** empfehlen. Auch um die neuen Layout-Manager und die Standardkomponenten kennen zu lernen ist dieses Tool eine schnelle Hilfe.

Zuerst wird ein Display erzeugt, welches die Schnittstelle zum Betriebssystem darstellt. Meistens wird in der gesamten Applikation nur ein Display benötigt. Danach wird das Shell erzeugt, was als Rahmenanwendung angesehen wird. Hier hat man auch die Möglichkeit die Titelleiste zu benennen. Wir wollen noch einen Button definieren, der beim Drücken auf den Standardoutput einen Text ausgibt.

Schließlich setzen wir noch die Größe, da wir keinen Layoutmanager benutzen und leiten die Nachrichtenschleife³ ein. Falls das Programm aus der Nachrichtenschleife springt, so wurde die Anwendung beendet und es werden die Komponenten wieder frei gegeben. Hier wird lediglich der Rahmen frei gegeben.

Der entstandene Code lautet dann wie folgt:

MyApp.java

```
import org.eclipse.swt.*;
import org.eclipse.swt.events.*;
import org.eclipse.swt.widgets.*;

public class MyApp {

    private Display display ;
    private Shell shell ;

    public MyApp ( String s ){
        Display display = new Display ();
        Shell shell = new Shell ( display );
        shell . setText ( s );

        Button button = new Button ( shell , SWT . PUSH );
        button . setText ( „Hello SWT-World„ );
        button . addSelectionListener ( new SelectionAdapter () {
            public void widgetSelected ( SelectionEvent se ) {
                System . out . println ( „Hello SWT-World„ );
            } //widgetSelected
        });

        button . setBounds ( 0 , 0 , 200 , 50 );
        shell . open ();

        while ( ! shell . isDisposed () ) {
            if ( ! display . readAndDispatch () ) {
                display . sleep ();
            } //if
        } //while

        display . dispose ();
    } //Konstruktor

    public static void main ( String [] args ) {
        MyApp app = new MyApp ( „MyApp„ );
    } //main
} //class MyApp
```

³in SWT wird dafür kein eigener Thread vorgesehen

Es gibt auch schon eine Reihe von Tutorials und Beispiele, die eine schnelle Einarbeitung ermöglichen. Bei meiner Arbeit mit SWT sind mir auch viele Ähnlichkeiten mit Swing aufgefallen wie z. B. , dass nur über einen Thread eine grafische Anweisung gemacht werden darf und als Ausweg wird bei beiden Systemen eine Methode zur Verfügung gestellt, die die Abarbeitung in den Grafik-Thread miteinschleust.

2 Plugins

Das JDT(Java Development Tooling) stellt die vollständige Integration einer Java-Entwicklungsumgebung zur Verfügung. Besonders durch die die **Refactor**-Methoden ist das JDT bekannt geworden.

Es existiert eine eigene Java-Sicht, die die Packet-Struktur eines Java-Projekts anzeigt. Bei Fehlern im Java-Code werden drei Fehlerklassen unterschieden, die man sich individuell einstellen kann. Zum Einem gibt es „Warnings“, mit denen man dennoch das Programm ohne Bedenken starten kann.

Bei der Standardeinstellungen fallen darunter:

- nicht benutze Imports
- Methoden, die „deprecated“ sind
- Überschriebene Methoden, die nicht mindestens Package-Sichtbarkeit haben
- Ein Zugriff auf ein statisches Attribut über ein Objekt
- Effektlose Zuweisung wie z. B. var = var
- Doppeltes auftreten von Java-Sourcen (z. B. aus jar-Library und eigenem Code)

Des weiteren gibt es auch „Errors“, bei dem das Programm nicht gestartet werden soll.

Hierzu gehören folgende Fälle:

- Code der unmöglich erreicht werden kann
- Ein Import, der nicht aufgelöst werden kann
- Unvollständiger Build-Pfad
- Zirkuläre Abhängigkeiten

Die letzte Fehlerklasse „Ignore“ werden von **Eclipse** nicht beachtet.

Wenn eine Klasse nicht gefunden wird, kann man mit „Organize Imports“ die fehlenden Imports einfügen lassen. Bei mehrdeutigen Klassen, wird eine Auswahl vorgeschlagen. Es werden alle Imports eindeutig aufgelöst, d. h. es gibt danach keine Wildcard-Imports, wie „**import java.util.*;**“.

Soll der Namen einer Klasse, Methode, Attribut, Parameter oder Variable geändert werden so kann die lästige Arbeit vom JDT übernommen werden. Dieser untersucht alle Abhängigkeit und benennt die Namen entsprechend um. Es können auch Textdateien oder Kommentare mit eingeschlossen werden. Früher änderte man den

Variablenamen an einer Stelle um und löste dann die Lawine von Fehlermeldungen einzeln auf, was unter Umständen ziemlich lange dauern kann. Die Anzahl der Fehlermeldungen steigt dort mit der Referenzzahl von anderen Klassen auf diesen Namen. Ebenso wird bei Fehlern versucht eine möglichst sinnvolle Lösung zu finden. So kann man sich bei nicht implementierten Interface-Schnittstellen die Methoden leer implementieren lassen, oder beim fehlen von try/catch-Klauseln die Klauseln automatisch erzeugen lassen.

Dies soll nur einen kurzen Ausflug auf die Features von **Eclipse** aufzeigen. Natürlich ist jede IDE eine Geschmackssache, jedoch spricht die enorm große Pluginzahl dafür, dass viele Entwickler mit **Eclipse** zufrieden sind.

3 Public Common License und Eclipse

ZITAT VON ERICH GAMMA AUS [2]

*Es braucht mehrere Anbieter, um das gesamte Spektrum an Entwicklungswerkzeugen abzudecken. Die Plattform ist dann erfolgreich, wenn viele verschiedene Anbieter Produkte darauf entwickeln. Anbieter, die in Produkte für **Eclipse** investieren, möchten aber unabhängig von einem einzelnen Anbieter sein und damit auch mehr Kontrolle über ihr eigenes Schicksal haben. Eine Opensource-Lizenz gibt den Anbietern genau diese Kontrolle. Zur Zeit wird ein großer Teil der **Eclipse**- Entwicklung noch von IBM getragen und vorangetrieben. Dies wird sich im Laufe der Zeit ändern. Ein gutes Beispiel dafür ist die C/C++-Entwicklungsumgebung, die vollständig außerhalb von IBM entwickelt wird.*

Eclipse steht unter der „Common Public License Version 1.0“. Jede Benutzung, Reproduktion oder Vertrieb von **Eclipse** steht ebenfalls unter dieser Lizenz.

Diese Lizenz wurde geschrieben um gemeinschaftliche Open Source Entwicklung zu unterstützen und bestärken. Sie ist so geschrieben, dass man diese auch in andere Softwarelizenzen integrieren kann. Die CPL wurde von der Open Source Initiative im Mai 2001 anerkannt.

Inhalt der Lizenz (siehe Original [3]):

3.1 Definitionen

Beitrag – im Falle eines Eigenbeitrages, ist hier der Code gemeint, der selbst geschrieben wurde, und die Dokumentation die unter dieser Lizenz verbreitet wurde. Im Falle einer Veränderung oder Erweiterung des Programms wo die Teile von einem Mitwirkenden verteilt wurden.

Veröffentlichungen enthalten nicht Erweiterungen, die entweder separate Softwaremodule darstellen, die unter ihren eigenen Lizenzbestimmungen vertrieben werden, oder die das Programm nicht erweitern.

Hier ist zu beachten, dass also in **Eclipse** Modulerweiterung kommerziell verarbeitet werden können. So ist z.B. das Produkt UML-Plugin Pro von Omondi kostenpflichtig und unter anderen Lizenzbedingungen festgelegt.

Mitwirkender – ist eine Person oder Einheit, die das Programm verteilt.

Lizenzierte Patente – sind Patentansprüche die für einen Mitwirkenden lizenzierbar sind und die notwendigerweise beim Gebrauch oder Vertrieb einer Erweiterung alleine oder kombiniert mit dem Programm verletzt werden.

Programm – sind Programmiererweiterungen, die in Übereinstimmung mit dieser Lizenz verteilt werden.

Bezieher – sind alle Personen, die das Programm unter dieser Lizenz erhalten und alle Erweiterungen enthält.

3.2 Gewährung der Rechte

Jeder Mitwirkende gewährt dem Bezieher einen nicht-exklusive, weltweite und gebührenfreie Copyright-Lizenz für Reproduktion, Veränderung, öffentlicher Aushang, öffentliche Arbeit, Verteilung und Sublizenzierung.

Des Weiteren werden hier noch die lizenzierten Patentrechte festgelegt.

3.3 Anforderungen

Wird eine vom Ersteller eigene Lizenzbedingung genutzt, so muss auf folgende Faktoren geachtet werden:

- Die Lizenz darf nicht der der CPL Lizenzklärung widersprechen.
- Die Lizenz darf sich nicht auf die Haftbarkeit der Mitwirkenden durch Zerstörung jeglicher Art wie z. B. Profitverlust berufen.

Falls das Programm mit Source-Code erhältlich ist:

- Muss es unter der CPL Lizenz stehen.
- Muss eine Kopie der CPL Lizenz in jeder Kopie des Programms enthalten sein.

Weiter dürfen die Lizenzen nicht verändert werden und der Urheber muss stets ersichtlich sein.

3.4 Kommerzielle Verbreitung

Hier wird im Groben geklärt, dass ein Vertreiber Verantwortung übernehmen kann, so dass ein Endnutzer mehr Sicherheit zum Produkt hat – d. h. der Vertreiber kann haftbar gemacht werden. Hier werden die zuverigen Mitwirkenden nicht in Verantwortung gezogen.

3.5 Keine Garantie

Gilt nur, wenn keine kommerzielle Verbreitung vorliegt:

Das Produkt wird so wie es ist bereitgestellt. Das heißt Garantie, Bedingungen, Limitationen, Fehlerfreiheit usw. sind nicht enthalten.

3.6 Ablehnung der Haftung

Gilt nur, wenn keine kommerzielle Verbreitung vorliegt:

Keiner der Mitwirkenden haftet an einen irgendwie gearteten Schaden. sogar, wenn dieser eine mögliche Schadensentwicklung gewusst hat.

3.7 Allgemein

Wird ein Teil dieser Lizenz ungültig, so ist der restliche Teil dennoch gültig. Es folgen weitere patentrechtliche Festlegungen und wie – bzw. von wem – eine Überarbeitung dieser Lizenz erfolgt.

4 Fazit

Das Projekt **Eclipse** wurde als Allgemeingut für Entwickler herausgegeben. Der Urheber IBM hat mit der CPL alle Rechte so festgelegt, damit dieses Projekt als Open Source angesehen werden darf. Selbst eine Umbenennung der Gesamtheit des Programms ist zulässig. IBM übergab das Projekt an ein unabhängiges Konsortium, dem mittlerweile mehrere Dutzend Mitglieder angehören.

Ein Nutzer dieser Software kann so aus dem Source lernen oder ihn erweitern. Er kann Bugfixes oder Erweiterungen schreiben und dies wieder der Allgemeinheit zukommen lassen. Wenn der Entwickler keine zusätzliche Lizenz angibt, gilt die CPL Lizenz. Wenn der Entwickler eine eigene Lizenz benutzt und sie der CPL nicht widerspricht, so kann er sein Werk unter eine andere Lizenz stellen. Wenn ein Schaden entsteht oder Entschädigungsansprüche gefordert werden, ist der Entwickler durch die CPL geschützt. Das heißt der Benutzer hat keine Garantie auf Fehlerfreiheit oder Support.

Wenn eine Firma aus dem **Eclipse**-Projekt ein neues Produkt erschafft oder neue Module für das **Eclipse**-Projekt zur Verfügung stellt, dann hat die Firma die Möglichkeit dafür Geld zu verlangen, indem die Institution eine eigene Lizenz angibt. Hier können auch Garantien und Haftbarkeit firmenspezifisch festgelegt werden. Daher ist die CPL ähnlich wie die GNU Lesser General Public License.

Hier wird ersichtlich, dass die CPL sowohl Open Source-Entwickler schützt als auch Absatz orientierte Firmen ein festes Fundament bietet. Und auch die hunderten von Plugin´s (siehe [5]) die es schon für **Eclipse** gibt scheinen zu zeigen, dass dies die richtige Herangehensweise ist.

Literatur

- [1] Ramin Assisi *Einführung und Referenz*, Hanser-Verlag, 2004, ISBN 3-446-22620-6
- [2] Erich Gamma *Codename E-clipse - Interview mit Erich Gamma*, Java'SPEKTRUM CEBIT, 2003 [hier als Pdf](#)
- [3] *Common Public License - v 1.0* [hier als Html](#)
- [4] *Eclipse Home*
- [5] *Eclipse Plugins*