

Die Eclipse Rich-Client-Plattform

Johannes Göhr

Fachbereich Informatik
Fachhochschule München

Entwicklung eines Computerspiels 08.05.2007

Inhalt

- 1 Einleitung**
 - Was ist eine Rich-Client-Plattform
 - Eclipse
- 2 Eclipse Produkt**
 - Allgemein
 - Beispiel FH-Experiment
- 3 Plugins**
 - Meta-Daten
 - Extension
 - Extension-Point

Index

1

Einleitung

- Was ist eine Rich-Client-Plattform
- Eclipse

2

Eclipse Produkt

- Allgemein
- Beispiel FH-Experiment

3

Plugins

- Meta-Daten
- Extension
- Extension-Point

Definition:

Rich-Client-Plattform

Der Rich-Client ist ein:

- neuer Ableger des Fat-Client
- meist handelt es sich um ein Framework
- durch Module und Plug-ins erweiterbar
- für unterschiedlichste Probleme geeignet
- er einfacher verteilbar und aktualisierbar

Eigenschaften einer Rich-Client-Plattform

Eigenschaften:

- Anpassungsfähigkeit an verschiedene Benutzer
- Anpassungsfähigkeit an verschiedene Geräte
- Einfache Verteilung an Endbenutzer
- Einfache Aktualisierung des Clients
- Komplexe Benutzeroberfläche möglich

Example

- Eclipse RCP

Eclipse Aufbau

Eclipse Core

- Eclipse Runtime
- SWT
- JFace
- Workbench
- Support für XML, Hilfe und Commands

Dazu gehört nicht:

- Ressourcen Verwaltung
- Java Editor
- Update Funktionen
- usw.

Eclipse Aufbau

Eclipse Core

- Eclipse Runtime
- SWT
- JFace
- Workbench
- Support für XML, Hilfe und Commands

Dazu gehört nicht:

- Ressourcen Verwaltung
- Java Editor
- Update Funktionen
- usw.

Index

- 1 Einleitung
 - Was ist eine Rich-Client-Plattform
 - Eclipse
- 2 **Eclipse Produkt**
 - Allgemein
 - Beispiel FH-Experiment
- 3 Plugins
 - Meta-Daten
 - Extension
 - Extension-Point

Das Produkt

Wie entsteht ein Produkt?

- 1 Definition einer Extension
org.eclipse.core.runtime.applications
- 2 Definition einer Extension
org.eclipse.core.runtime.products
- 3 Anlegen einer *.product* Datei

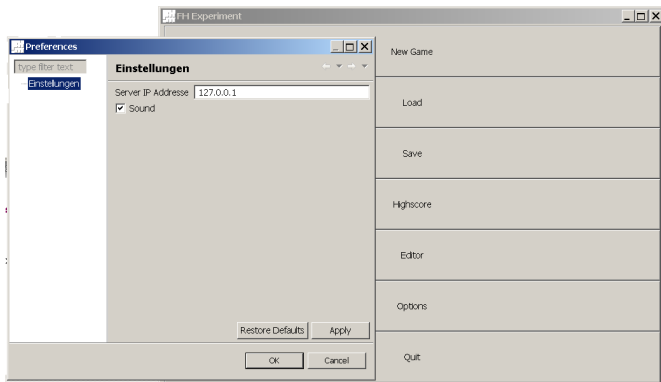
Inhalt der *.product* Datei

- Produktname und Produkt ID
- Plugins und Features die zu diesem Product gehören
- Icon und Dateiname der ausführbaren Datei für jede Plattform

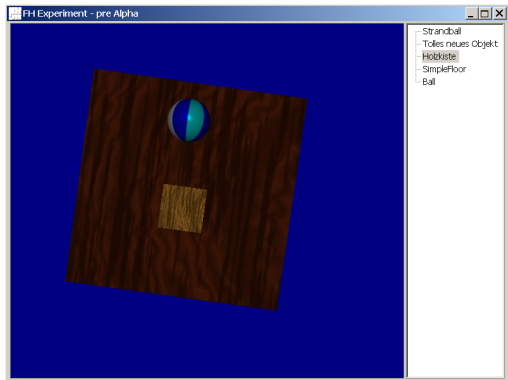
Klassen zu einem Produkt

```
public class Application implements IPlatformRunnable
{
    /*
     * (non-Javadoc)
     *
     * @see org.eclipse.core.runtime.IPlatformRunnable#run(java.lang.Object)
     */
    public Object run(Object args) throws Exception
    {
        Display display = PlatformUI.createDisplay();
        try
        {
            int returnCode = PlatformUI.createAndRunWorkbench(display,
                new ApplicationWorkbenchAdvisor());
            if (returnCode == PlatformUI.RETURN_RESTART)
            {
                return IPlatformRunnable.EXIT_RESTART;
            }
            return IPlatformRunnable.EXIT_OK;
        }
        finally
        {
            display.dispose();
        }
    }
}
```

Screenshots



Screenshots



Index

- 1 Einleitung
 - Was ist eine Rich-Client-Plattform
 - Eclipse
- 2 Eclipse Produkt
 - Allgemein
 - Beispiel FH-Experiment
- 3 **Plugins**
 - Meta-Daten
 - Extension
 - Extension-Point

Definition:

Plugin

- Ein Plugin ist eine in sich abgeschlossene Einheit eines Produktes.
- Es kann Abhängigkeiten zu anderen Plugins haben.
- Es kann Schnittstellen zu anderen Plugins zur Verfügung stellen.
- Es ist austauschbar.
- Ein Plugin wird über Meta-Daten beschrieben.

MANIFEST.MF

Inhalt der MANIFEST.MF:

- Plugin Name
- Version
- Classpath
- Native Binärdateien
- Exportierte Packages

MANIFEST.MF

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: FH Experiment Ressourcen
Bundle-SymbolicName: edu.fhm.cs.fhexperiment.ressourcen
Bundle-Version: 1.0.0
Bundle-Localization: plugin
Eclipse-LazyStart: true
Bundle-ClassPath: lib/commons-logging-1.0.4.jar,
    lib/jogl.jar,
    lib/junit-4.3.1.jar,
    lib/log4j-1.2.14.jar,
    lib/vecmath-1.5.jar,
    ..
    lib/oscilloscope-0.1.jar
Bundle-NativeCode:
    /lib/natives/win32/jogl.dll ; osname=win32; processor=x86,
    /lib/natives/win32/jogl.dll ; osname=Windows Vista; processor=x86,
    /lib/natives/linux/libjogl.so ; osname =Linux ;processor = x86,
    /lib/natives/mac/libjogl.jnilib ; osname = Mac OS X; processor = ppc,
    /lib/natives/mac/libjogl.jnilib ; osname = Mac OS X; processor = x86
Export-Package: com.sun.gluegen.runtime,
    com.sun.opengl.cg,
    com.sun.opengl.impl,
    com.sun.opengl.impl.error,
    com.sun.opengl.impl.glue,
    com.sun.opengl.impl.macosx,
    com.sun.opengl.impl.mipmap,
    com.sun.opengl.impl.registry,
```

Definition: Extension

Eine Extension hat

- Ein beschreibendes XML Schema
- Ein Attribut *point*
- XML Elemente

Example

```
<extension
  point="edu.fhm.cs.fhexperiment.Texture">
  <Texture
    textureDatatype="png"
    textureFilename="texture/wood.png"
    textureName="wood"/>
```

Xml Schema

Example

```

<element name="Texture">
  <annotation>
    <appInfo>
      <meta.element labelAttribute="textureName"/>
    </appInfo>
  </annotation>
  <complexType>
    <attribute name="textureName" type="string" use="required">
      <annotation>
        <documentation>
        </documentation>
      </annotation>
    </attribute>
    <attribute name="textureFilename" type="string" use="required">
      <annotation>
        <documentation>

        </documentation>
      <appInfo>
        <meta.attribute kind="resource"/>
      </appInfo>
    </annotation>
  </attribute>
  <attribute name="textureDatatype" type="string">
    <annotation>
      <documentation>
      </documentation>
    </annotation>
  </attribute>
</complexType>
</element>

```

Wichtige Extensions

Vorteile

- Man kann Funktionen aus anderen Plugins verwenden.
- Andere Plugins können eigene Funktionen verwenden.

Example

- *org.eclipse.ui.views* **view**
- *org.eclipse.ui.perspectives* **perspective**
- *org.eclipse.ui.actionSets* **action**
- *org.eclipse.ui.commands* **command**
- *org.eclipse.ui.bindings* **key**
- *org.eclipse.ui.preferencePages* **page**

Definition: Extension-Point

Extension-Point

Ein Extension Point ist eine Verbindung zwischen einer Extension und dem Plugin.

Dadurch wird es möglich Funktionen zu benutzen, die zum Kompilierzeitpunkt noch nicht bekannt sind.

Sie können nachträglich über Extensions bekannt gemacht werden.

Example

```
<extension-point
  id="edu.fhm.cs.fhexperiment.Items"
  name="Teile die man verwenden kann"
  schema="schema/edu.fhm.cs.fhexperiment.Items.exsd"
/>
```

Benutzen einer Extension

Example

```
public URL getTexture(String name) throws IOException{
    IExtensionPoint extension = Platform.getExtensionRegistry()
        .getExtensionPoint("edu.fhn.cs.fhexperiment.Texture");
    if (extension != null) {
        IConfigurationElement[] configurationElements = extension
            .getConfigurationElements();
        for (IConfigurationElement element : configurationElements) {
            if ("Texture".equals(element.getName())) {
                if (element.getAttribute("textureName").equals(
                    name)) {

                    String filename = element.getAttribute("textureFilename");
                    Bundle bundle = Platform.getBundle(element.getContributor()
                        .getName());
                    URL url = bundle.getResource(filename);

                    return url;
                }
            }
        }
    }
    throw new IOException("Texture not found");
}
```

Extension-Point

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
  <extension
    id="application"
    point="org.eclipse.core.runtime.applications">
    <application>
      <run
        class="edu.fhm.cs.fhexperiment.application.Application">
      </run>
    </application>
  </extension>
  <extension-point
    id="edu.fhm.cs.fhexperiment.Items"
    name="Teile die man verwenden kann"
    schema="schema/edu.fhm.cs.fhexperiment.Items.exsd"
  />
  <extension
    point="edu.fhm.cs.fhexperiment.Items">
    <Part partClass="edu.fhm.cs.fhexperiment.model.items.SimpleFloor"/>
    <Part partClass="edu.fhm.cs.fhexperiment.model.items.Ball"/>
  </extension>
  <extension
    point="org.eclipse.ui.views">
    <view
      class="edu.fhm.cs.fhexperiment.control.plugin.view.GLView"
      id="edu.fhm.cs.fhexperiment.control.plugin.view.GLView"
      name="GLView"/>
    <view
      class="edu.fhm.cs.fhexperiment.control.plugin.view.ListView"
      id="edu.fhm.cs.fhexperiment.control.plugin.view.ListView"
      name="ListView"/>
    <view
      class="edu.fhm.cs.fhexperiment.control.plugin.view.MenuView"
      id="edu.fhm.cs.fhexperiment.control.plugin.view.MenuView"
      name="MenuView"/>
  </extension>
</plugin>
```

Summary

- Es gibt viele Aufgaben die an die Rich-Client-Plattform delegiert werden können.
- Um die RCP sinnvoll einzusetzen muss man ihre Funktionen auch kennen.
- Man muss nicht jedes Rad neu erfinden, wenn man einige Plugins kennt.
- Die Entscheidung für eine RCP hat Auswirkungen auf das Design einer Anwendung

Abschluss

Vielen Dank für eure Aufmerksamkeit!