it-wps
Die Software-Architekten

"Building Non-IDE Applications with the Eclipse Plug-in Component Model

Petra Becker-Pechau, Niels Kausche, Martin Lippert
Holger Breitling, Matthias Lübken, Sebastian Sanitz

Tutorial at NetObjectDays 2003, Erfurt
22 September 2003

Tutorial Outline

it-wps
Die Software-Architekten

- Beyond Eclipse as an IDE
- Plug-in Concepts in Detail
- Examples
- Construction of Plug-ins
- A Hands-On Example: JHotDraw
- Open Questions and Conclusion



Timetable

it-wps Die Software-Architekten

13:30	Welcome
13:45	Introduction to Eclipse and its Plug-In Component Model
14:30	Hands-On Example
15:00	Break
15:30	Excercise
16:45	Concluding Discussion
17:00	End of the Tutorial

What is Eclipse?

it-wps Die Software-Architekten

Eclipse is an Integrated Developed Environment (IDE) for Java

Eclipse is also a language independent IDE

- Java IDE on top of the platform
- C/C++ IDE
- COBOL IDE
- many more languages ...

Eclipse is more than an IDE, a Platform

➔Providing

- a plug-in component model, widget toolkit, workbench, etc.

The Eclipse Project

it-wps Die Software-Architekten

Eclipse is more than a Platform, a Collection of Projects

- **Eclipse:** Platform and IDE(s)
- **Tools:** Additional tools for developers (GEF, Hyades, ...)
- **Technology:** research-oriented projects focussing on parts of Eclipse (AspectJ, AJDT, Equinox, ...)
- **Web Tools:** planned J2EE development platform

The Eclipse Platform

it-wps Die Software-Architekten

The Eclipse Platform:

- ➔ Plug-in Component Model
  - allows to develop Eclipse itself as a set of small parts called plug-ins
- ➔ SWT
  - a stand-alone widget toolkit for Java
- ➔ Resources, Workbench
  - Providing a general vocabulary for resources, editors, views, perspectives, etc.

Goal for Eclipse 3.0:

- ➔ Improve Eclipse as Rich-Client Platform:
  - Workbench independent of any IDE-specific features or plug-ins

Eclipse Plug-in Model

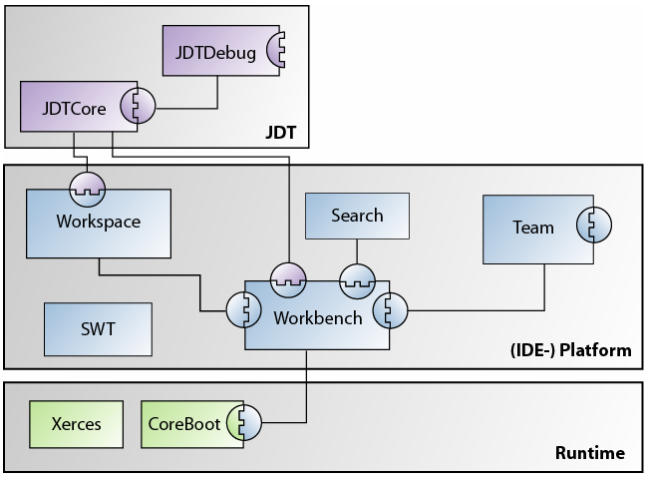
it-wps Die Software-Architekten

What is a "Plug-in"?

- ➔ A plug-in is a small component that has ...
  - ... a name, id, vendor, version, etc.
  - ... some implementations and/or resources
  - ... a well-defined import interface
  - ... a well-defined export interface
- ➔ It is quite similar to a module

In Eclipse, Everything is a Plug-in

it-wps Die Software-Architekten



Other Examples

it-wps Die Software-Architekten

Banking application:

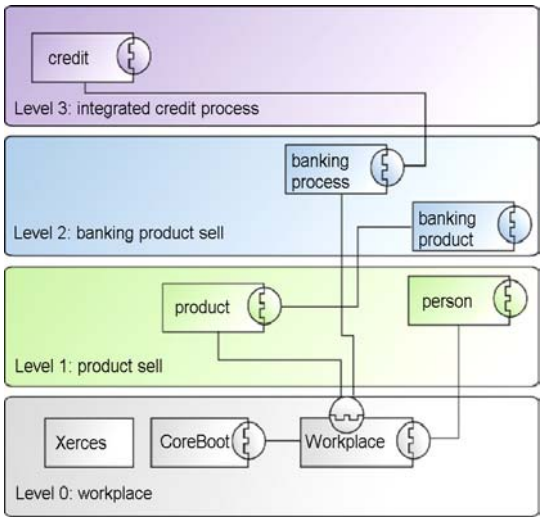
- built on top of plug-ins
- allows third-party companies to integrate products into the banking workplace

JWAM:

- is a framework for the tools and materials approach

JLense:

- JLense is a plug-in-based application framework for building Java applications

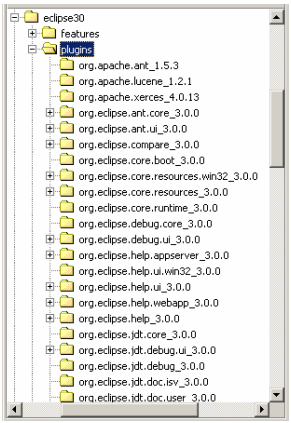


The Difference

it-wps Die Software-Architekten

Plug-ins are not Plug-ins

- Plug-ins are often used to allow extensions by third parties
  - Adobe Photoshop for example
  - Normally only specific extensions possible
- The idea behind Eclipse plug-ins is different:
  - Build the complete application as a combination of small plug-ins



it-wps Die Software-Architekten

Construction of Plug-ins



it-wps Die Software-Architekten

Separated Namespaces

Plug-ins define separate namespaces

- ➔ only explicitly exported types (interfaces or classes) are known to other plug-ins
- ➔ different plug-ins can use different versions of the same class (or library) internally
- ➔ implemented with an own classloader per plug-in (classes are referenced using name, package *and* classloader)

Plug-ins introduce modularisation to Java.

Sample plugin.xml – each plug-in has one

it-wps Die Software-Architekten

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin
   id="de.unihamburg.informatik.swt.jhotdraw.platform"
   name="JHotDrawPlatform"
   version="1.0.0">

   <runtime>
      <library name="JHotDrawPlatform.jar">
         <export name="*"/>
      </library>
   </runtime>

   <requires>
      <import plugin="org.eclipse.core.boot"/>
      <import plugin="org.eclipse.core.runtime"/>
   </requires>

</plugin>
```

Extensions and Extension-Points (1/3)

it-wps Die Software-Architekten

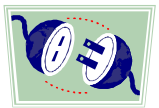






Extension Points

- ➔ a plug-in can define a number of extension points
- ➔ an extension point consists of
  - an id
  - a (display) name
  - an optional xml schema file
  - normally some exported types (part of xml schema)
- ➔ meta information is stored in the manifest file (plugin.xml)



```
<extension-point
        id="palettefigures"
        name="JHotDraw Palette Figure"/>
```



Extensions and Extension-Points (2/3)

it-wps Die Software-Architekten

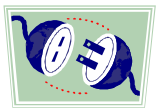






Extensions

- a plug-in can define a number of extensions
- extend a specific extension point (referenced via its id)
- an extension consists of
  - a name and an id
  - specific data in the the manifest file (plugin.xml) matching the extension point’s xml schema
  - normally some subtypes of expected interfaces or classes (attribute “class”)





Extensions and Extension-Points (3/3)

it-wps Die Software-Architekten

```
<extension
        id="swtfigure"
        name="SWT Figure"
        point="de.unihamburg.informatik.swt.
                jhotdraw.platform.palettefigures">
    <figure
        iconname="de/unihamburg/informatik/swt/
                jhotdraw/figures/images/uhh"
        class="de.unihamburg.informatik.swt.
                jhotdraw.figures.SWTFigure"
        toolname="SWT Power Figure">
    </figure>
 </extension>
```

Extension point „applications“

it-wps Die Software-Architekten

- If everything is a plug-in what is the „first“ plug-in?
  - org.eclipse.core.runtime
- This plug-in is started by „startup.jar“ and establishes the plug-in component model.
- It provides an extension point „applications“ for applications.
- Extensions of „applications“ have to implement the interface
  `org.eclipse.core.boot.IPlatform`

Sample plugin.xml

it-wps Die Software-Architekten

```
<?xml version="1.0" encoding="UTF-8"?>
<plug-in
   id="de.unihamburg.informatik.swt.jhotdraw.platform"
   name="JHotDrawPlatform"
   version="1.0.0">
[...]
   <extension-point id="palettefigures"
                    name="JHotDraw Palette Figure"/>
   <extension id="JHotDrawPlatformRunnable"
              name="JHotDrawPlatformRunnable"
              point="org.eclipse.core.runtime.applications">
      <application>
         <run class="de.unihamburg.informatik.swt.
                    jhotdraw.platform.JHotDrawPlatformRunnable">
         </run>
      </application>
   </extension>
</plug-in>
```

Eclipse Plug-ins in Detail

it-wps Die Software-Architekten

Mandatory Plug-ins

- org.eclipse.core.boot
  - application extension point
- org.eclipse.core.runtime
  - plug-in registry
  - access to extension points object model
- org.apache.xerces
  - XML processing

Plug-in Development Environment (PDE)

it-wps Die Software-Architekten

The Eclipse Java IDE contains an extension for building plug-ins: PDE

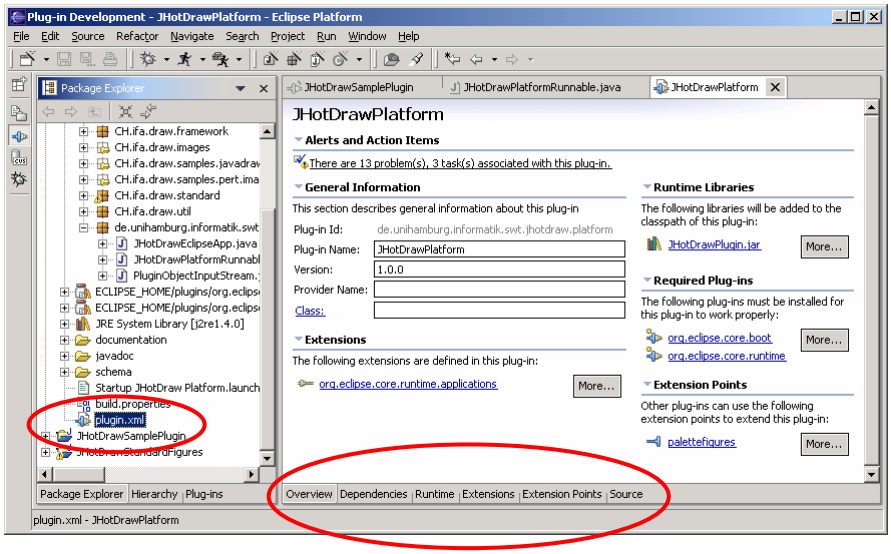
- ease plug-in development, mainly to enhance the Eclipse IDE
- can also be used for any other kind of plug-in development

Features

- enhance built classpath with exported types from imported plug-ins
- allows workspace and external plug-ins
- provides runtime-workspace
- find and define dependencies between plug-ins
- view extension points and extension

PDE in detail

it-wps Die Software-Architekten



Our example application: JHotDraw

it-wps Die Software-Architekten

JHotDraw:

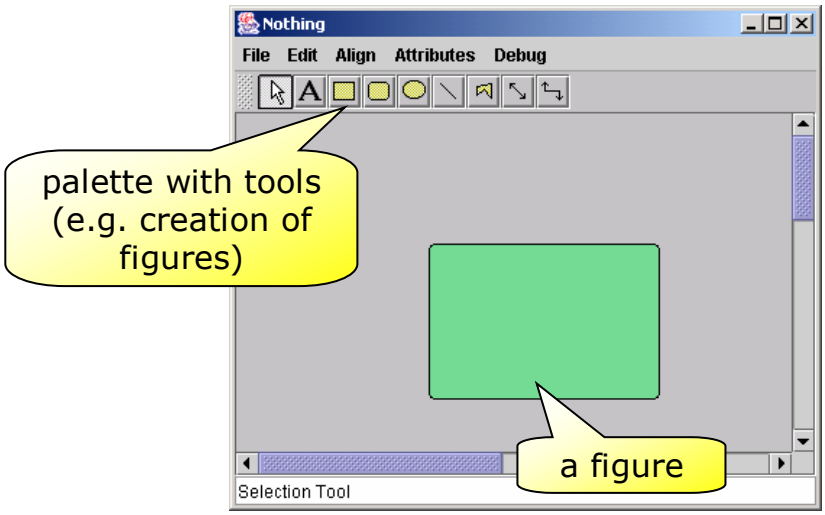
- ➔ is a famous drawing application framework based on the work by Erich Gamma
- ➔ is a Java version of the original HotDraw, written in Smalltalk (by Ward Cunningham and Kent Beck)
- ➔ http://jhotdraw.sourceforge.net/

A pluggable JHotDraw:

- ➔ our modified version is based on the Eclipse plug-in model
- ➔ provides a set of extension points to be extensible via plug-ins

JHotDraw in a nutshell

it-wps Die Software-Architekten



JHotDraw Applications without Plug-ins

it-wps Die Software-Architekten

```
public class NothingApp extends DrawApplication {
  public NothingApp() {
    super("Nothing");
  }

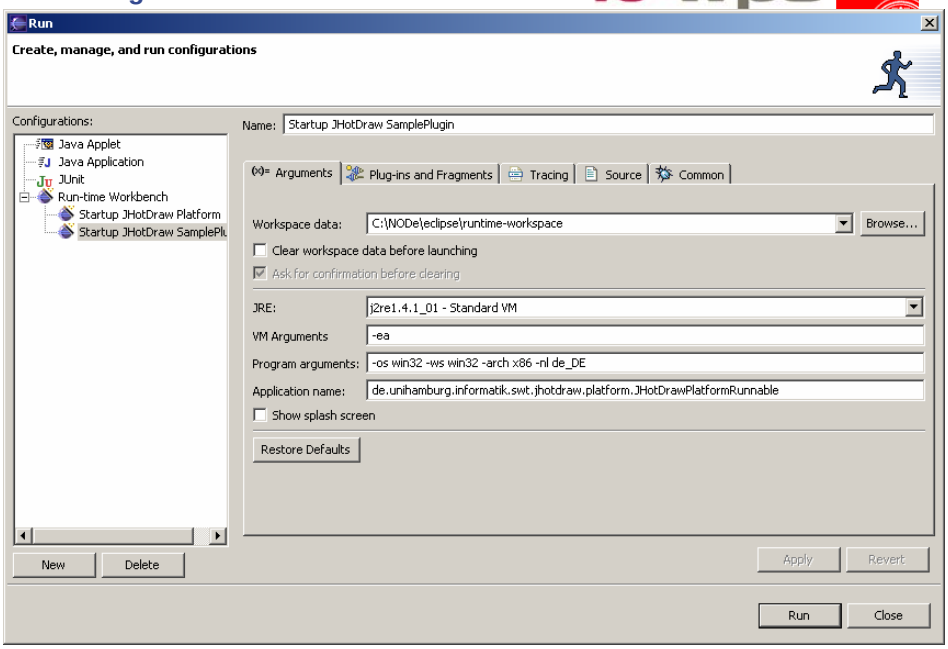
  protected void createTools(JToolBar palette) {
    super.createTools(palette);

    Tool tool = new TextTool(this, new TextFigure());
    palette.add(createToolButton(IMAGES+"TEXT", "Text Tool", tool));

    tool = new CreationTool(this, new RectangleFigure());
    palette.add(createToolButton(IMAGES+"RRECT","Rectangle Tool",tool));
    [...]
  }

  // start application
  public static void main(String[] args) {
    DrawApplication window = new NothingApp();
    window.open();
  }
}
```

Starting JHotDraw from PDE



Pluggable JHotDraw Extension Point

Die Software-Architekten

Adding figures (CreationTools with figures) to the tool palette.

Extension point id:

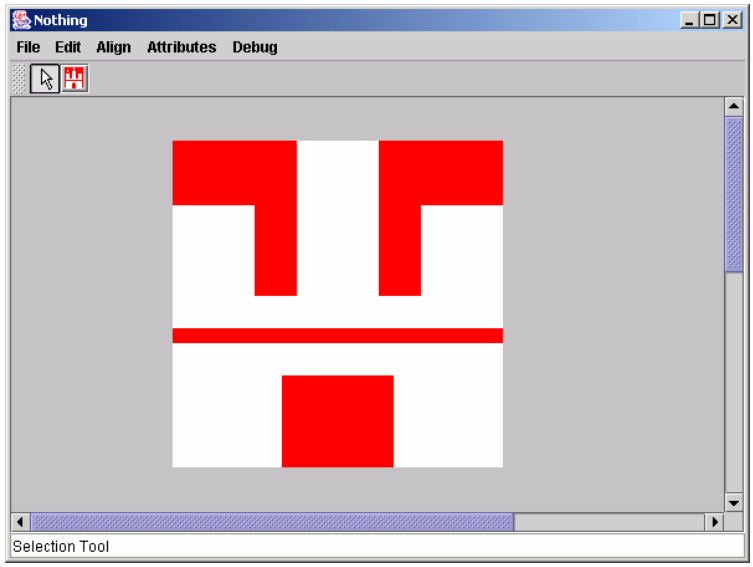
- `de.unihamburg.informatik.swt.jhotdraw.platform.palettefigures`

Needs: <figure> element with attributes

- class – full referenced classname of class that implements `(Abstract)Figure`
- iconname – file path and name prefix of icon images
- iconplugin – id of the plug-in which contains the images (optional)
- toolname - text for tool tip

Pluggable JHotDraw with Sample Plugin

it-wps Die Software-Architekten



Pluggable JHotDraw: The Plug-Ins

it-wps Die Software-Architekten

JHotDrawPlatform

- ➔ the basic application
- ➔ defines `palettefigures` extension point

JHotDrawStandardFigures

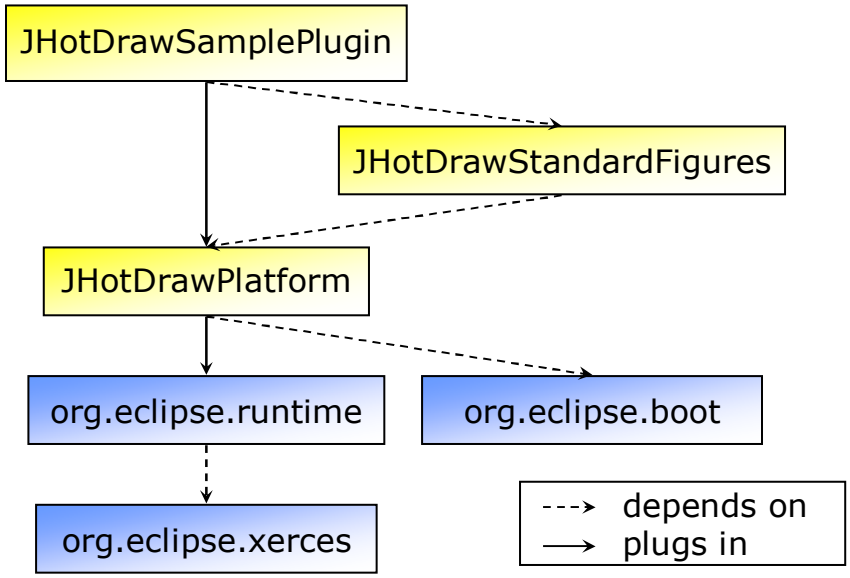
- ➔ contains all standard figures and icon images

JHotDrawSamplePlugin

- ➔ introduces and adds new figure to tool palette (SWT Figure)

The JHotDraw Plug-Ins

it-wps Die Software-Architekten



it-wps Die Software-Architekten

Hands-on



JHotDraw Platform and Sample Plug-in

it-wps Die Software-Architekten

- import the JHotDrawPlatform plug-in
- start the base application (`de.unihamburg.informatik.swt.jhotdraw.platform.JHotDrawPlatformRunnable`)
- import the JHotDrawSamplePlugin
- configure the runtime-workspace to include this plug-in
- start the base application again (with the sample plug-in)

Pluggable JHotDraw: Our Modifications

it-wps Die Software-Architekten

`JHotDrawPlatformRunnable`

- extension-class for `applications` extension point
- creates our JHotDraw application

`JHotDrawEclipseApp`

- JHotDraw application that reads plugged palette figures from an extension point

`PluginObjectInputStream`

- used for cloning prototype figures (in `AbstractFigure`)
- reading the class definition from a temporary byte stream now uses explicitly specified class loader

`IconKit / ToolButton`

- loading icon images now uses explicitly specified class loader

Exercise

it-wps Die Software-Architekten

1. **add standard figure**
   - ➔ import the JHotDrawStandardFigures plug-in
   - ➔ change the plugin.xml file of the sample plugin to add a new standard figure from this plug-in
2. **implement and add new figure**
   - ➔ implement a new figure in the JHotDrawSamplePlugin project (e.g. `Triangle`)
   - ➔ change the plugin.xml file of the sample plugin to add this figure to the tool palette

Loading an Extension Point's Extensions

it-wps Die Software-Architekten

1. Get the extension point from the platform
2. Get the registered extensions (IExtension)
3. For each extension, get the XML elements inside it (IConfigurationElement)
4. For each configuration element, create an object whose class will be based on the contents of the class attribute. Validate that the defined attributes are complete.
5. Save the newly created extensions in a collection

it-wps Die Software-Architekten

Open Questions and Conclusions



it-wps Die Software-Architekten

Open Questions

There are still some open issues with the eclipse plug-in model

- ugly libraries (using a non-standard way of class-loading)
- server-side plug-ins (within an application server)
- communication across VMs (serialization difficult because of the advanced class-loading)
- References to resources across plug-ins

Conclusions
1/2

it-wps Die Software-Architekten

The Eclipse Plug-in Model is a nice mechanism to ...

- ➔ ... simulate a module mechanism in Java (on top of packages)
  - improved modularization
- ➔ ... structure large applications into small parts
  - team development profits
- ➔ ... make the architecture of the system explicit
  - when using plug-ins in a well-organized way
- ➔ …make the system more extendable
  - extension points define how and where extensions are possible
  - plug-ins for extension points introduce new functionality without changing the base system

Conclusions
2/2

it-wps Die Software-Architekten

Rich IDE-Support

- ➔ ... Eclipse Java IDE provides rich tools to do plug-in development
- ➔ ... but it does not depend on the Eclipse IDE
  - we did plug-in development with IntelliJ IDEA as well, for example

it-wps
Die Software-Architekten

Thank you for your attention !!!

Petra Becker-Pechau, Niels Kausche, Martin Lippert
University of Hamburg, it-wps GmbH

email:
becker@informatik.uni-hamburg.de, niels.kausche@it-wps.de, lippert@acm.org