



20. Eclipse and Framework Extension Languages

Prof. Uwe Aßmann

TU Dresden

Institut für Software- und Multimediatechnik

Lehrstuhl Softwaretechnologie

Version 11-1.0, 12/17/11



References

- ▶ Frank Gerhardt, Christian Wege. Neuer Reichtum – Eclipse als Basis für Rich-Client-Anwendungen. IX 7/2004, Heise-Verlag.
- ▶ Ed Burnett. RCP tutorial.
<http://www.eclipse.org/articles/Article-RCP-1/tutorial1.html>
- ▶ S. Shavor, J. D'Anjou, S. Fairbrother, D. Kehn, J. Kellerman, P. MacCarthy. The Java Developer's Guide to Eclipse. Addison-Wesley, 2003

Overview of the Course

Eclipse

SAP

Concrete Frameworks

Tools & Materials

Pattern Languages

Layered Frameworks

Patterns and Frameworks

Metapatterns
and Framework patterns

Composite Patterns

Role Models

Employment and Usage

Basic Patterns

Variability Patterns

Extensibility Patterns

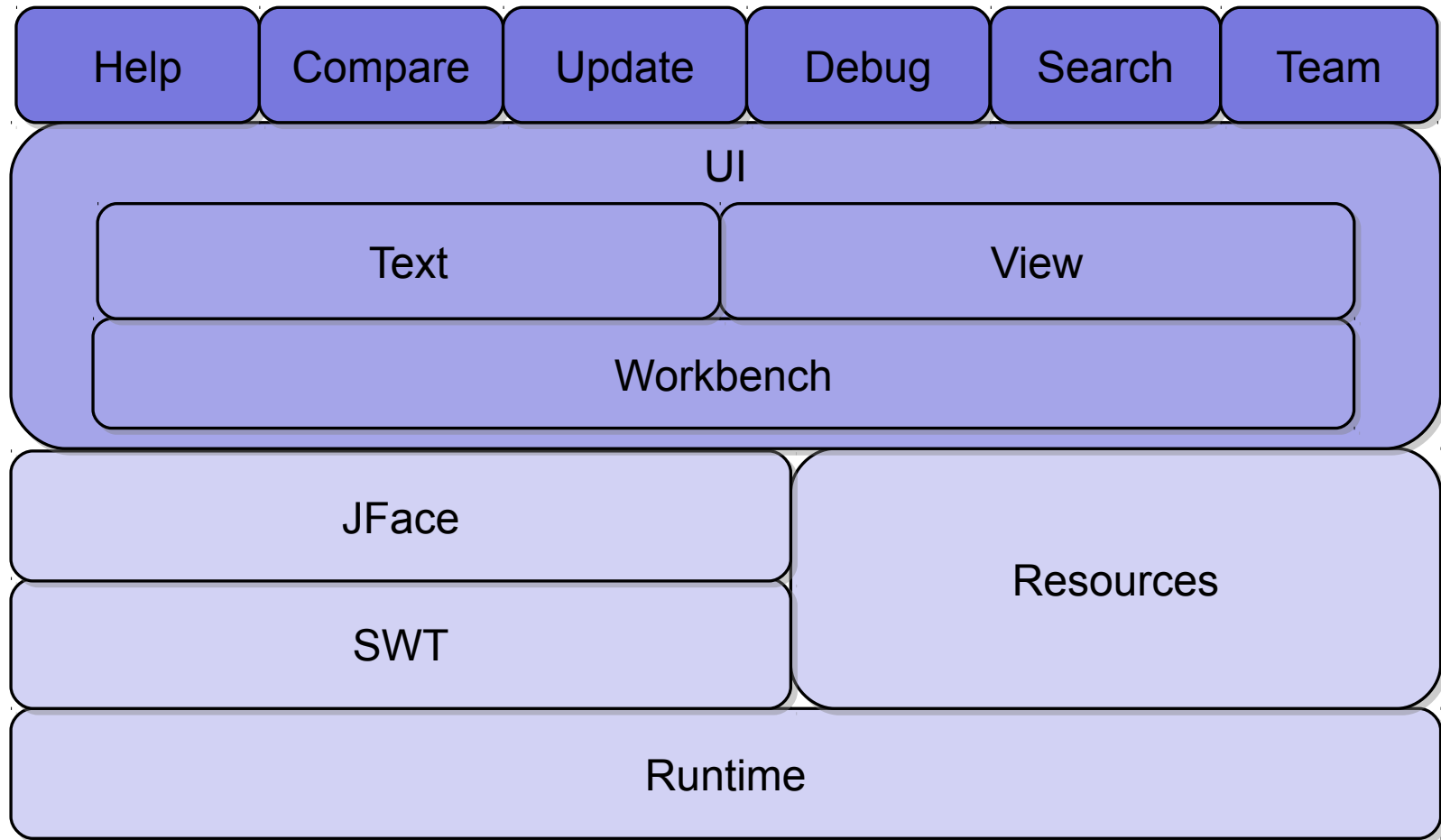
Connection Patterns

Intro

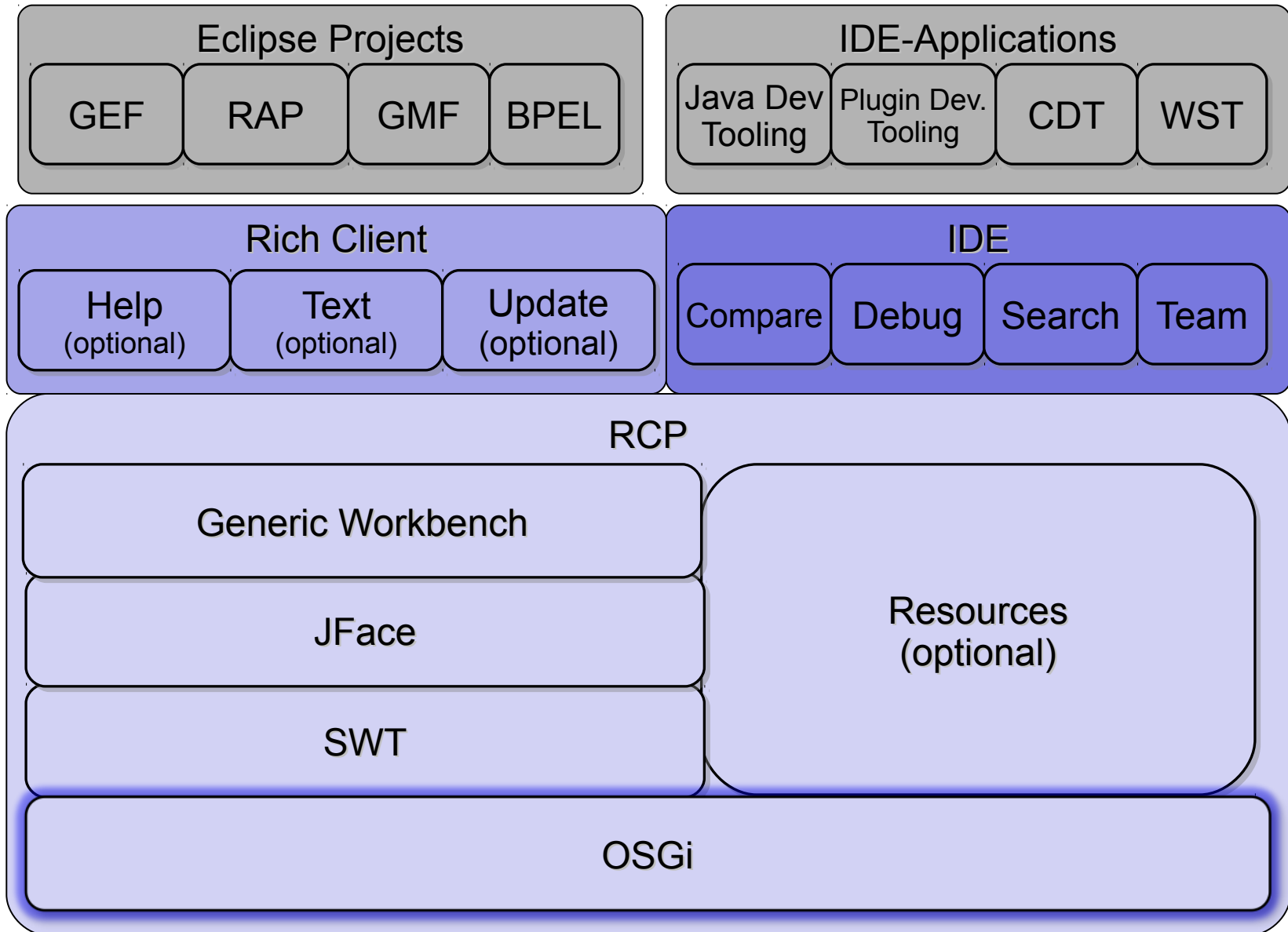
Eclipse Structure

- ▶ Eclipse is a set of frameworks for development of
 - IDE applications
 - IDE (not only for Java)
 - GUI applications
 - Rich thin clients
- ▶ To this end, it stacks several frameworks

Eclipse Framework 2.x

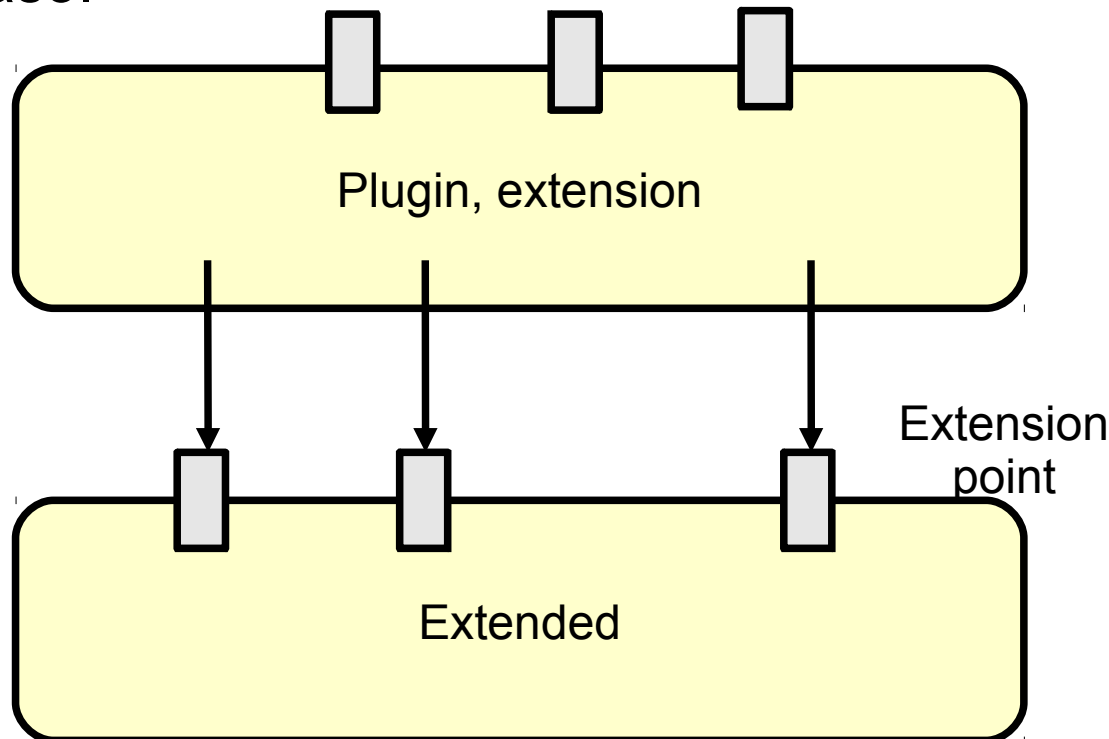


Eclipse Framework 3.x



Plugins and Extensions Points

- ▶ Eclipse frameworks carry framework extension hooks, *extension points*.
 - No concept for variability, only extension
- ▶ An upper-level framework (or the rest of the application), which is fed into a lower-level framework, is called *plugin* or *extension*
- ▶ Extension points can be classes, menus, properties, class path entries, aso.



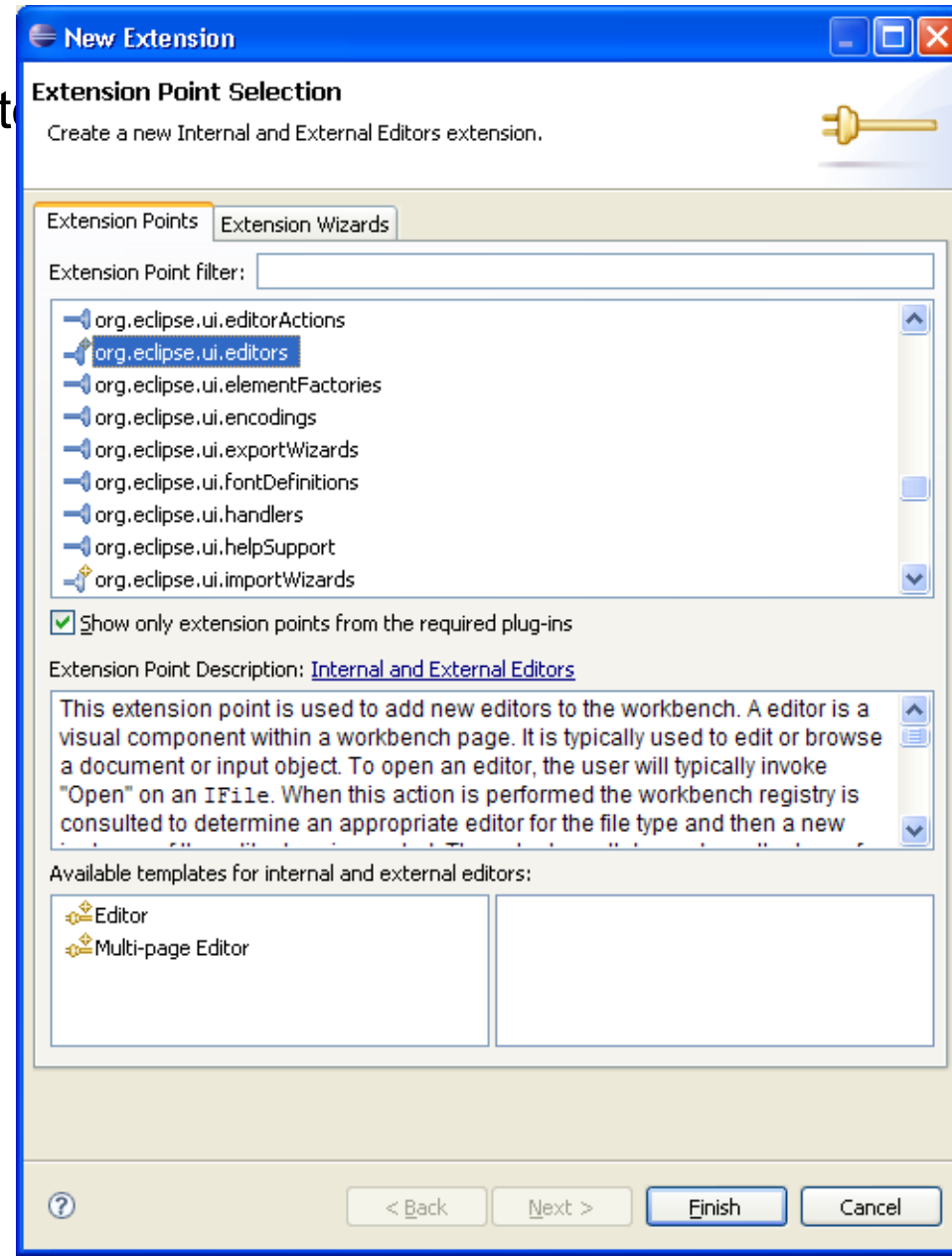
Plugins (Extensions)

- ▶ Are classes that are dynamically loaded from a special directory `eclipse/plugins`
- ▶ Every plugin is represented by a *plugin class*,
- ▶ Specifies a **manifest** file (runtime properties)
- ▶ And the `plugin.xml` (usage of extension points)

```
Manifest-Version: 1.0
Bundle-SymbolicName: org.eclipse.ui; singleton:=true
Bundle-Activator: org.eclipse.ui.internal.UIPlugin
Bundle-ManifestVersion: 2
Bundle-Version: 3.4.0.I20080610-1200
Require-Bundle: org.eclipse.core.runtime;bundle-version="[3.2.0,4.0.0)",
org.eclipse.swt;bundle-version="[3.3.0,4.0.0)";visibility:=reexport,
org.eclipse.jface;bundle-version="[3.4.0,4.0.0)";visibility:=reexport,
org.eclipse.ui.workbench;bundle-version="[3.4.0,4.0.0)";visibility:=reexport,
org.eclipse.core.expressions;bundle-version="[3.4.0,4.0.0)"
Bundle-Name: %Plugin.name
Bundle-Localization: plugin
Bundle-ClassPath: .
Bundle-ActivationPolicy: lazy
Export-Package: org.eclipse.ui.internal;x-internal:=true
```


Some Extension Points

- ▶ Actions
 - Menu bar, toolbar to views and editors
 - Menu choices
 - Object context menu
- ▶ Creation wizard for File
 - >New
- ▶ Preference page to Window
 - >Preferences
- ▶ Views for Window
 - >ShowView
- ▶ Perspectives for Window
 - >OpenPerspectives
- ▶ Help manual for Help
 - >HelpContents



Using Extension Points and Extensions in plugin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>
  <extension-point id="org.tud.ospp.ProcessState" name="ProcessState"
    schema="schema/org.tud.ospp.ProcessState.exsd"/>
  <extension
    point="org.eclipse.ui.perspectives">
    <perspective
      name="Modeller Perspective"
      class="org.tud.ospp.graph.ModellerPerspective"
      id="ospp.modeller">
    </perspective>
  </extension>
  <extension
    point="org.eclipse.ui.views">
    <view
      allowMultiple="false"
      icon="icons/repo1.png"
      name="Process Repository View"
      class="org.tud.ospp.graph.view.RepositoryView"
      id="ospp.repository">
    </view>
  </extension>
</plugin>
```

Declare a new
ExtensionPoint

Register a
perspective

Register a view

The Plugin Class

- ▶ Represents the plugin
- ▶ Extends class `Plugin` or `AbstractUIPlugin`
- ▶ Has functions to handle directories for persistent state and intermediate data
- ▶ Handles input streams
- ▶ Treats plugin preferences

```
public class LocalityPlugin extends AbstractUIPlugin
{
    /**
     * This method is called upon plug-in activation
     */
    public void start(BundleContext context) throws Exception
    {
        super.start(context);
    }

    /**
     * This method is called when the plug-in is stopped
     */
    public void stop(BundleContext context) throws Exception
    {
        super.stop(context);
    }
}
```

Extension Points are Ubiquitous

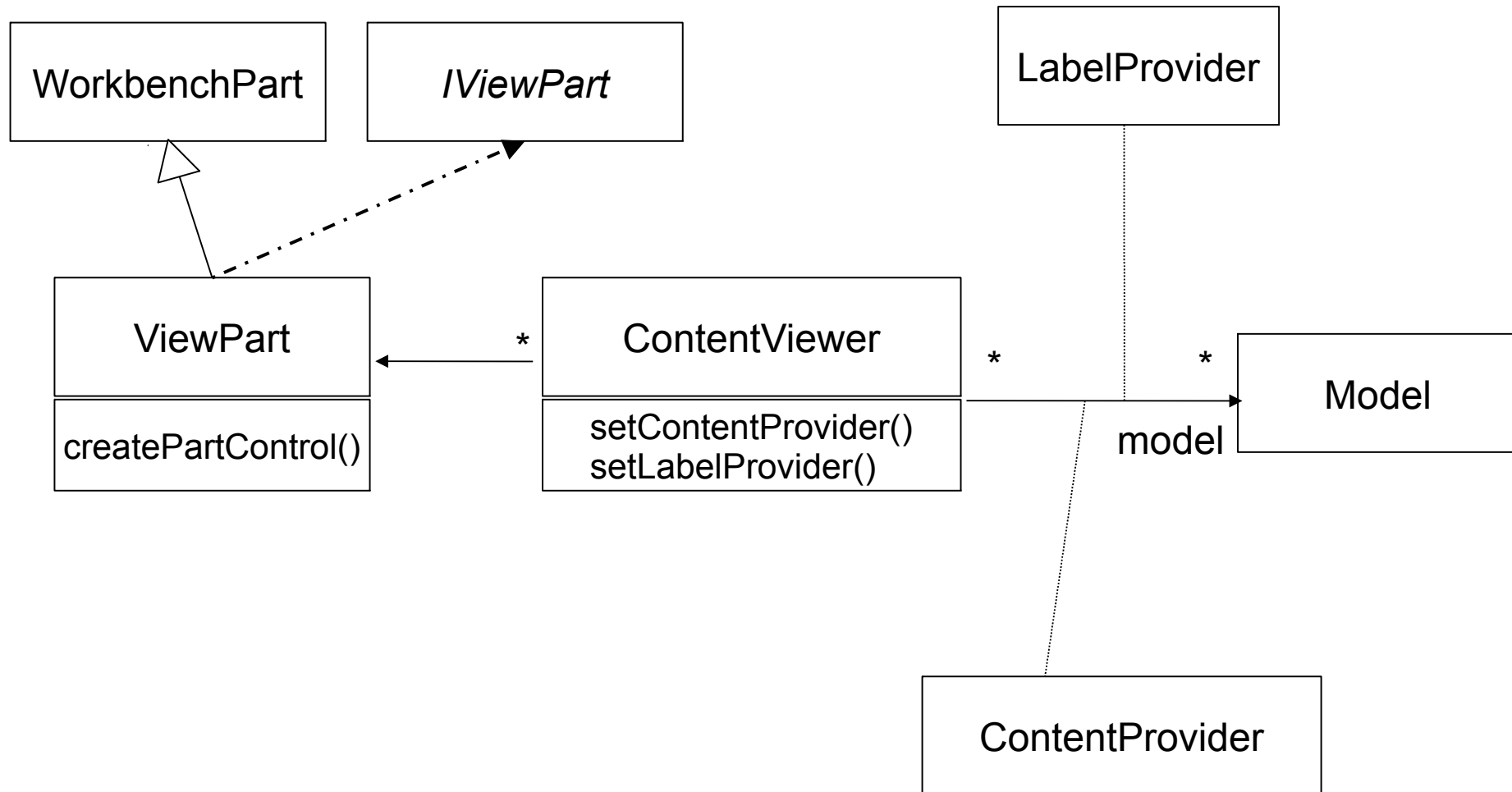
- ▶ Eclipse generalizes the hook concept from framework hooks to extension points of
 - Resources
 - Pages for page tabs
 - Menu entries and their underlying commands, e.g., creation wizards
 - Views
 - Editors
 - Perspectives
 - Help
- ▶ i.e., to other conceptual entities of the Eclipse RCP

To make a good application GUI framework, hooks need to be defined on all tools, materials, and environments of the framework

The Generic Workbench (part of the RCP)

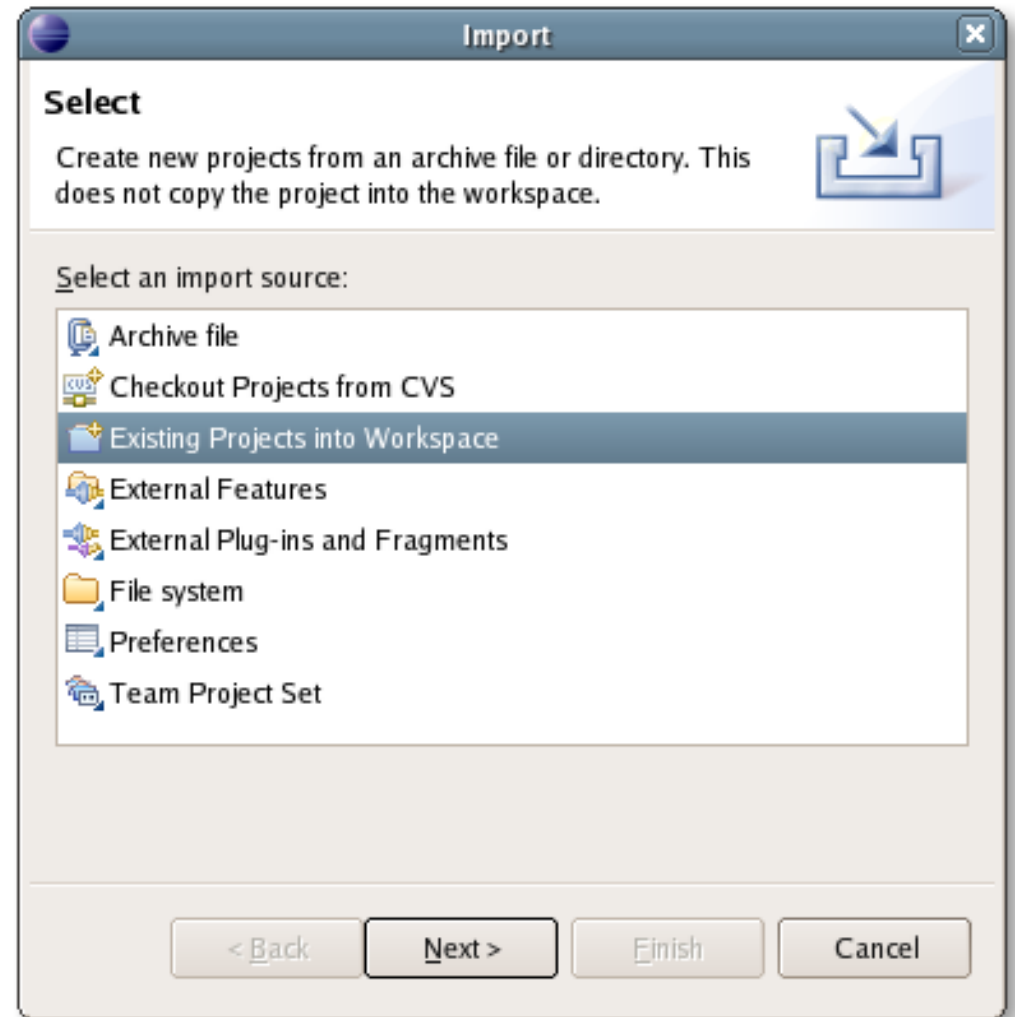
- ▶ The Generic Workbench structures and organizes the GUI of an RCP application
 - File, Edit, Resources, Run, Navigate, Help menu entries
 - Uses one or several Perspectives with Editors and Views
- ▶ **Perspective**: A collection of editors and views, bundled together in a specific GUI configuration
- ▶ **Editors**: tool to edit an artifact
- ▶ **View**: view onto an artifact
 - Outline views
 - Structural views
 - Property views
 - Graphic views
- ▶ The **workbench** can be extended on all three levels (new perspectives, new editors, new views)

Views Use Viewers to Display Models



JFace Predefined Viewers

- ▶ JFace on top of SWT
- ▶ Predefined Dialogs, Actions, Wizards and Viewer:
 - ▶ TableView
 - ▶ TextViewer
 - ▶ TreeViewer
 - ▶ **ListViewer**
 - ▶ PropertySheetViewer
 - ▶ CheckboxTreeViewer
 - ▶ ...



The Plugin Development Environment PDE

- ▶ PDE has a *registry* for plugins
 - Different views and editors for plugins (e.g., Tree-based view)
- ▶ PDE New Extension wizard for creating extensions
 - Template-controlled wizards
 - User-written wizards
- ▶ The Extension Wizard selects a project code generation wizard
 - A wizard generating the initial plugin code
 - Creating a standalone version of the RCP application, without the development environment (if the application should run standalone)

Insight: Language-Controlled Framework Extension

- ▶ Framework extension points (framework hooks) are *interpreted* in Eclipse.
 - Instead of specifying them as a framework hook pattern, the core interpreter interprets XML files to know how to extend extension points
 - Hence, Eclipse has a little domain-specific language (DSL) for extension points and bindings of them (language-controlled extension)
- ▶ This goes beyond the framework hook patterns, because they only use polymorphism and design patterns.

Eclipse' main feature is an *extension language interpreter*.

The Future of Eclipse

**Eclipse will stay, because it has the first
*framework extension language***

- ▶ There might be a market for about 3-5 framework extension languages, in which the product families of the world will be made
- ▶ *Can you define other framework extension languages?*

The Nature of Framework Hooks

Framework hook patterns provide a very simple framework extension language.

- ▶ The framework hook patterns can all be written down in logic (see exercises).
- ▶ Hence, they provide a little constraint language for variability and extensibility of frameworks.
- ▶ Variability and Extensibility are distinguished by
 - 1 or n multiplicity constraint (see description logic)
 - Object recursion or non-recursion (recursive logic or non-recursive)

The End

▶ www.eclipse.org