Modellbasierte Softwareentwicklung (MODSOFT)

Part II

Domain Specific Languages

Eclipse / Plug-ins

Prof. Joachim Fischer /
Dr. Markus Scheidgen / Dipl.-Inf. Andreas Blunk

{fischer,scheidge,blunk}@informatik.hu-berlin.de

LFE Systemanalyse, III.310

Agenda

prolog (1 VL)	**Introduction:** languages and their aspects, modeling vs. programming, meta-modeling and the 4 layer model
➡ 0. (2 VL)	**Eclipse/Plug-ins:** eclipse, plug-in model and plug-in description, features, *p2*-repositories, *RCP*s
1. (2 VL)	**Structure:** *Ecore*, *genmodel*, working with generated code, constraints with *Java* and *OCL*, *XML/XMI*
2. (3 VL)	**Notation:** Customizing the tree-editor, textural with *XText*, graphical with *GEF* and *GMF*
3. (4 VL)	**Semantics:** interpreters with Java, code-generation with *Java* and *XTend*, model-transformations with *Java* and *ATL*
epilog (2 VL)	**Tools:** persisting large models, model versioning and comparison, model evolution and co-adaption, modular languages with *XBase*, *Meta Programming System* (MPS)

Eclipse/Eclipse-Plug-ins – Agenda

- Eclipse
- OSGi and Equinox
- Plug-in Architecture
- Eclipse Platform (a.k.a Existing Plug-ins)
- Plug-in Development Environment (PDE)
- Plug-in Distribution Options

Eclipse

Eclipse – History

- Eclipse started as a proprietary IBM product (IBM Visual age for Smalltalk/Java).
- Eclipse is open source - it is a general purpose open platform that facilitates and encourages the development of third party plug-ins.
- Eclipse is best known as an *Integrated Development Environment* (IDE).
- Eclipse was originally designed for Java, now supports many other languages.
  - C, C++, Python, PHP, Ruby
  - XML, HTML, CSS
  - ant, maven, and many more

Original Eclipse Project Aims

- Provide open platform for application development tools
  - Run on a wide range of operating systems
  - GUI and non-GUI
- Language-neutral
  - Permit unrestricted content types
  - HTML, Java, C, JSP, EJB, XML, GIF, …
- Facilitate seamless tool integration
  - At UI and deeper
  - Add new tools to existing installed products
- Attract community of tool developers
  - Including independent software vendors (ISVs)
  - Capitalize on popularity of Java for writing tools

What is Eclipse, what is an IDE

- In this lecture we manly see eclipse as an IDE.
- Programming requires the use of many tools:
  - editors (vim, emacs)
  - compilers (gcc, javac)
  - code analyzers (lyn)
  - debuggers (gdb, jdb)
  - build-tools (make, ant, maven)
  - version control (cvs, svn, git, ClearCase)
- IDEs integrate those tools into a single coherent environment.
  - one rich graphical user interface
  - one configuration scheme
  - The different tools are *integrated* with each other.

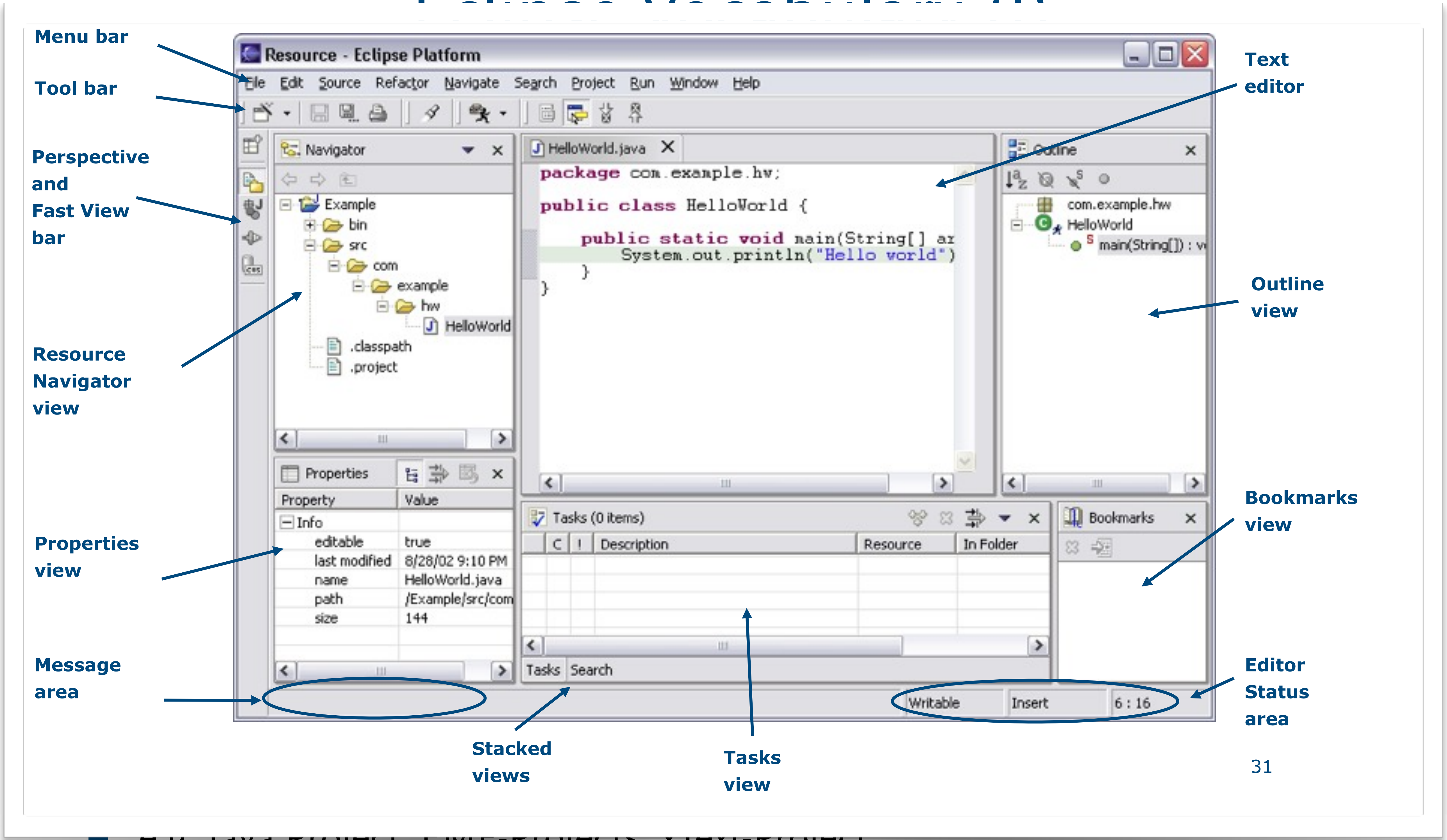
Eclipse Versions

- download: http://www.eclipse.org/downloads/
- Eclipse 3.x releases are: Callisto, Europa, Ganymede, Galileo, Helios, Indigo (3.7, latest)
- Eclipse 4.x releases are: Juno (4.2), Kepler (4.3), Luna (4.4, current)
- There is a 32- and 64-bit version for Windows, MacOS, and Linux/Unix.
- Eclipse is Java-based but uses SWT, a GUI-toolkit with platform specific versions.
- There are different packages (different collections of plug-ins) for different use-case. For this lecture *Eclipse Modeling Tools* has most needed plug-ins pre-installed.

Eclipse Vocabulary (I)

- Workbench, Perspective, Editor, View
- Project
  - organizational unit for your work
  - corresponds to a folder on your hard-drive, by default in the workspace directory
  - is a resource
- Project Properties
  - project specific configuration
    allows to create project specific settings for large parts of the preferences
- Project Nature
  - e.g. Java Project, EMF-Projects, xText-Project
  - determines project properties, build-process, specific sub-folder types (e.g. source-folder)

Eclipse Vocabulary (I)



```
package com.example.hw;

public class HelloWorld {

    public static void main(String[] ar
        System.out.println("Hello world")
    }
}
```

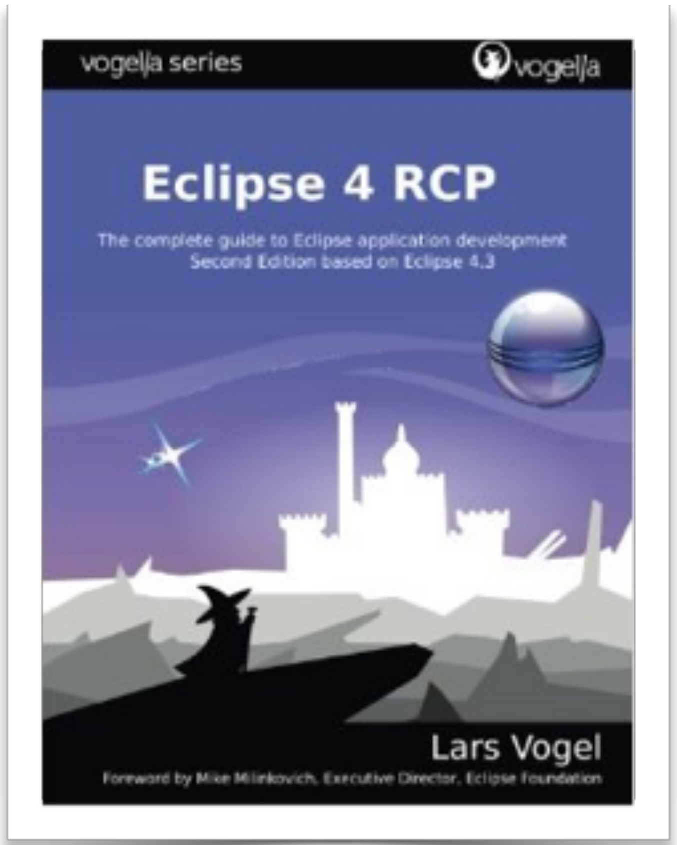
Property	Value
Info	
editable	true
last modified	8/28/02 9:10 PM
name	HelloWorld.java
path	/Example/src/com
size	144

- e.g. Java Project, EMF-Projects, Xtext-Project
- determines project properties, build-process, specific sub-folder types (e.g. source-folder)

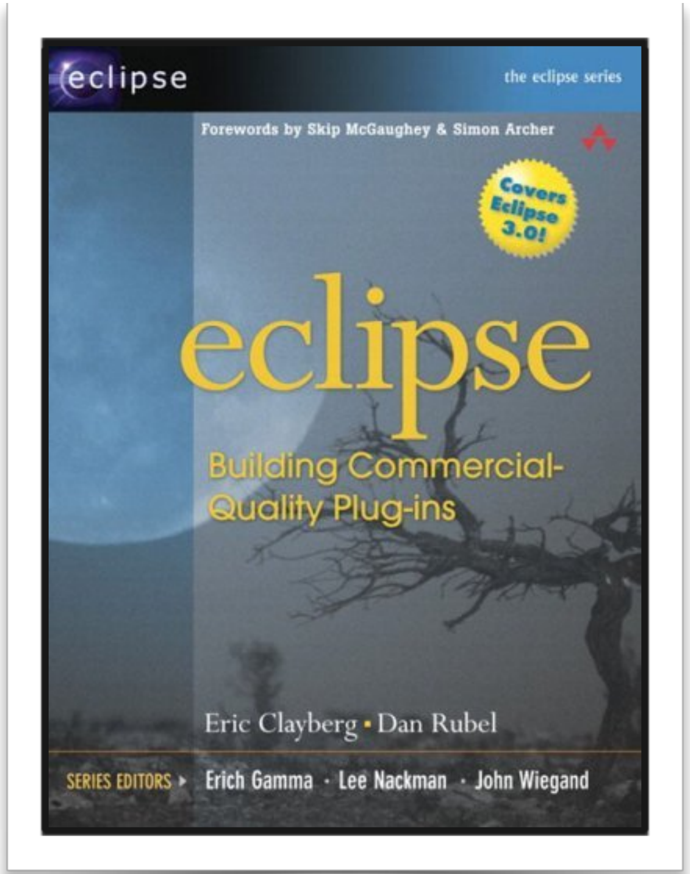
Eclipse Vocabulary (II)

- Resource
  - generic term for folders, files, and sometimes file-like (virtual resources) entities
- Preferences
  - eclipse wide configuration organized by plug-ins
- Launch-configuration
  - e.g. Java Application, RCP/Eclipse Application
  - used to Run, Debug, Profile

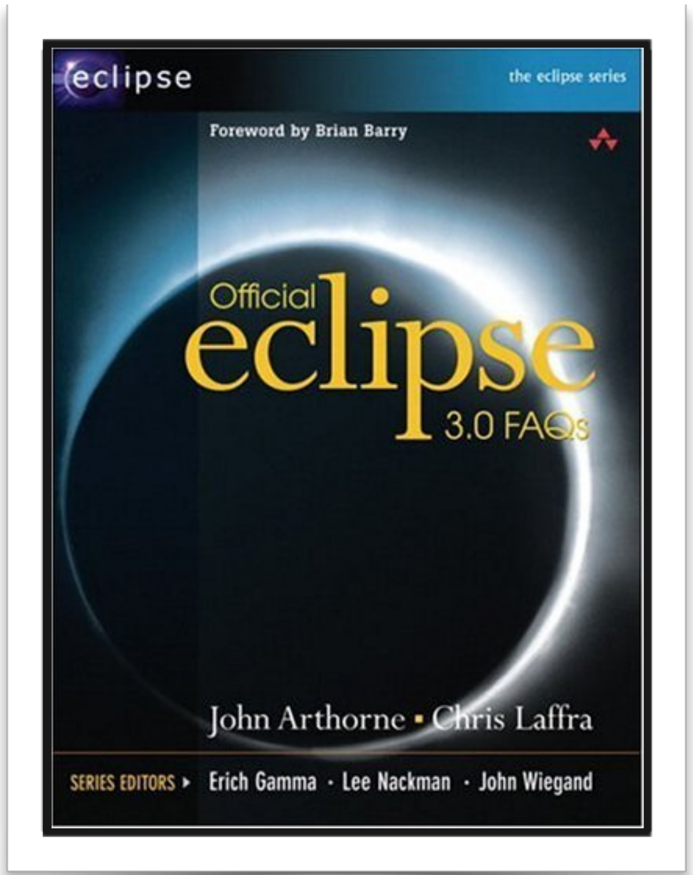
Eclipse Books/Resources


vogella series
vogella
Eclipse 4 RCP
The complete guide to Eclipse application development
Second Edition based on Eclipse 4.3
Lars Vogel
Foreword by Mike Milinkovich, Executive Director, Eclipse Foundation

Lars Vogel: http://www.vogella.com/tutorials/eclipse.html
Eclipse: http://help.eclipse.org/ (PDE Dev. Guide)


eclipse
the eclipse series
Forewords by Skip McGaughey & Simon Archer
Covers Eclipse 3.0!
eclipse
Building Commercial-Quality Plug-ins
Eric Clayberg • Dan Rubel
SERIES EDITORS ▸ Erich Gamma · Lee Nackman · John Wiegand

eclipse
the eclipse series
eclipse
Rich Client Platform
Designing, Coding, and Packaging Java Applications
Jeff McAffer • Jean-Michel Lemieux
SERIES EDITORS ▸ Erich Gamma · Lee Nackman · John Wiegand


eclipse
the eclipse series
Foreword by Brian Barry
Official eclipse 3.0 FAQs
John Arthorne • Chris Laffra
SERIES EDITORS ▸ Erich Gamma · Lee Nackman · John Wiegand

Eclipse Plug-ins Vocabulary

- OSGi, Equinox
  - Open Service Gateway initiative (OSGi) specification
  - modular system and service platform
  - dynamic component model
  - Equinox is one implementation of OSGi
- Bundle, Plug-in, Feature, Application
- Dependency, Extension, Extension-point
- Plug-in Development Environment (PDE)
- PDE-project
  - special Java-project nature
  - contains *manifest* and *plugin.xml*

Eclipse Architecture (I)

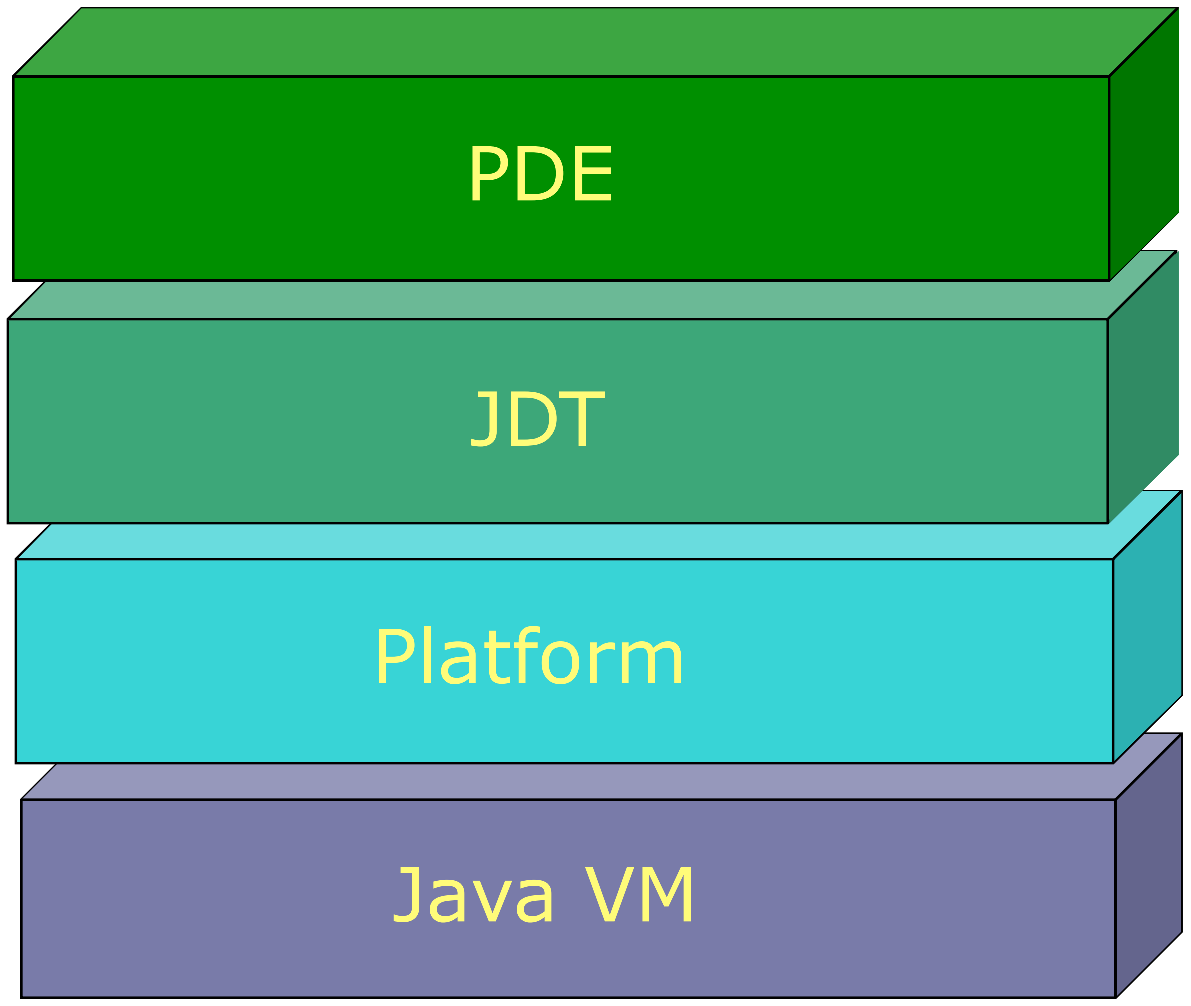
- Eclipse is a universal platform for integrating development tools
- Open, extensible architecture based on plug-ins

Plug-in development environment (PDE)

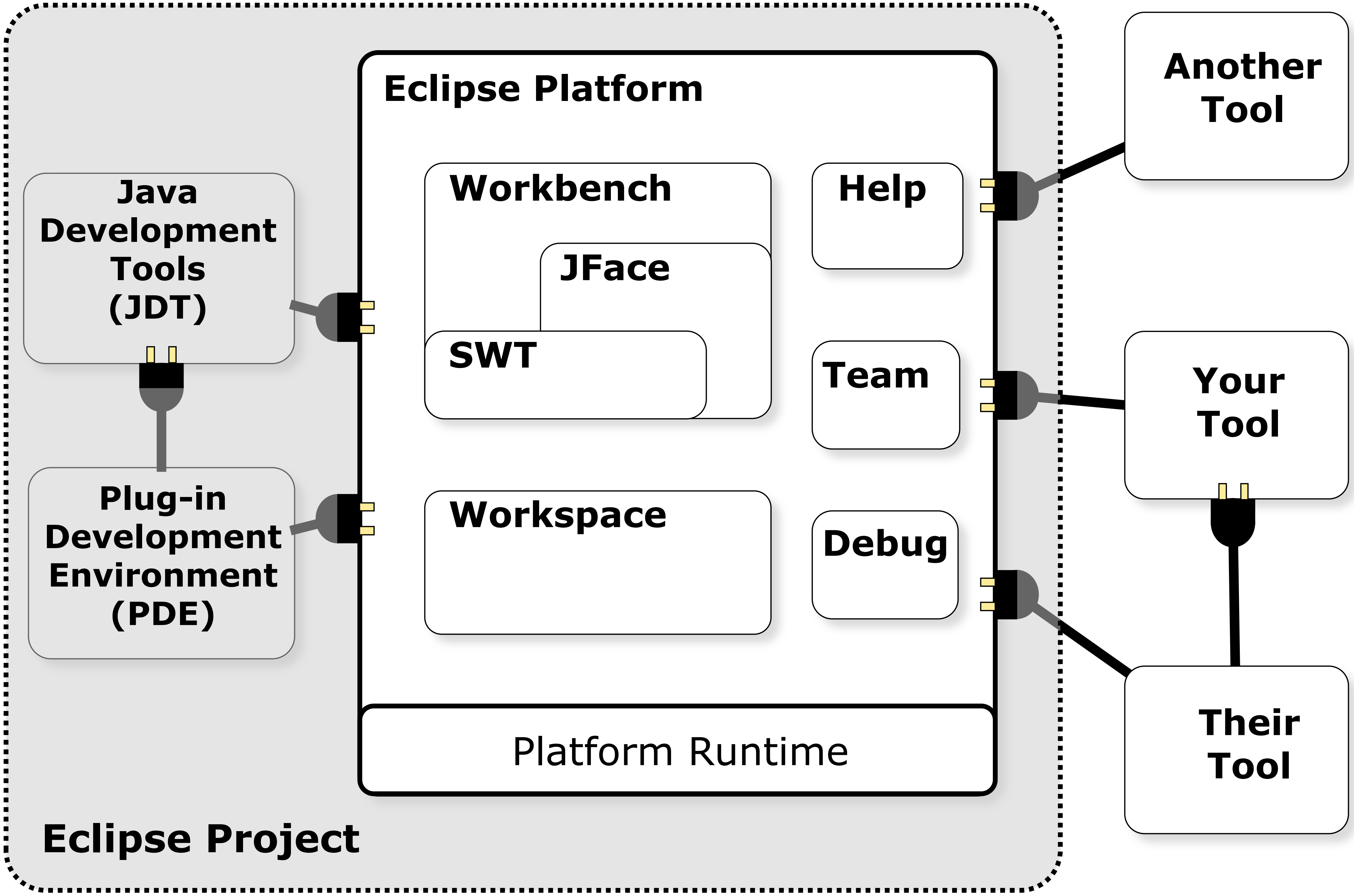
Java development tools (JDT)

Eclipse Platform

Standard Java2 Virtual Machine

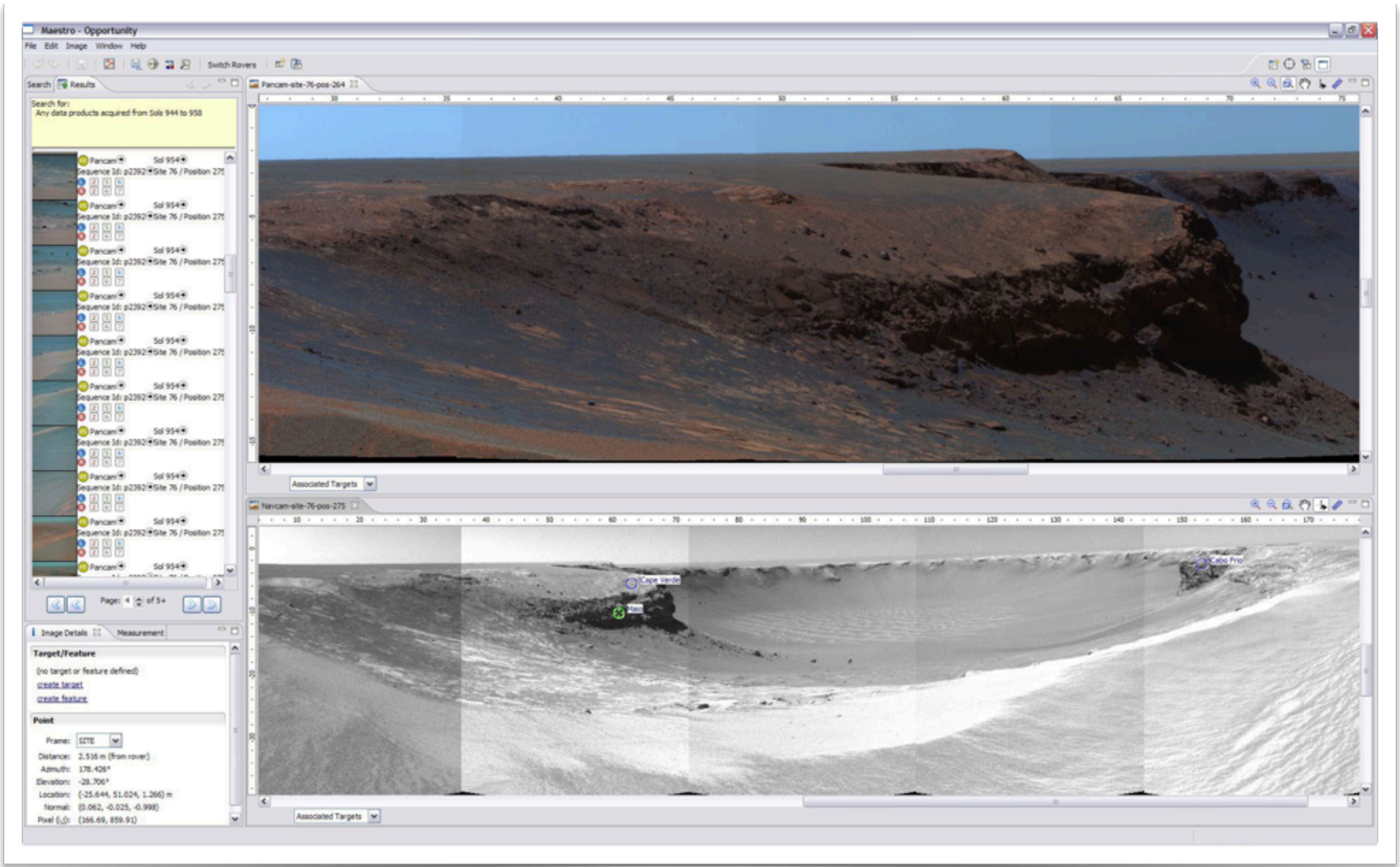


Eclipse Architecture (II)



Eclipse RCPs – Examples

Eclipse RCPs – Examples



Search for:
Any data products acquired from Sols 944 to 958

Pancam Sol 954
Sequence Id: p2392 Site 76 / Position 275

Pancam Sol 954
Sequence Id: p2392 Site 76 / Position 275

Pancam Sol 954
Sequence Id: p2392 Site 76 / Position 275

Pancam Sol 954
Sequence Id: p2392 Site 76 / Position 275

Pancam Sol 954
Sequence Id: p2392 Site 76 / Position 275

Pancam Sol 954
Sequence Id: p2392 Site 76 / Position 275

Pancam Sol 954
Sequence Id: p2392 Site 76 / Position 275

Pancam Sol 954
Sequence Id: p2392 Site 76 / Position 275

Pancam Sol 954
Sequence Id: p2392 Site 76 / Position 275

Pancam Sol 954

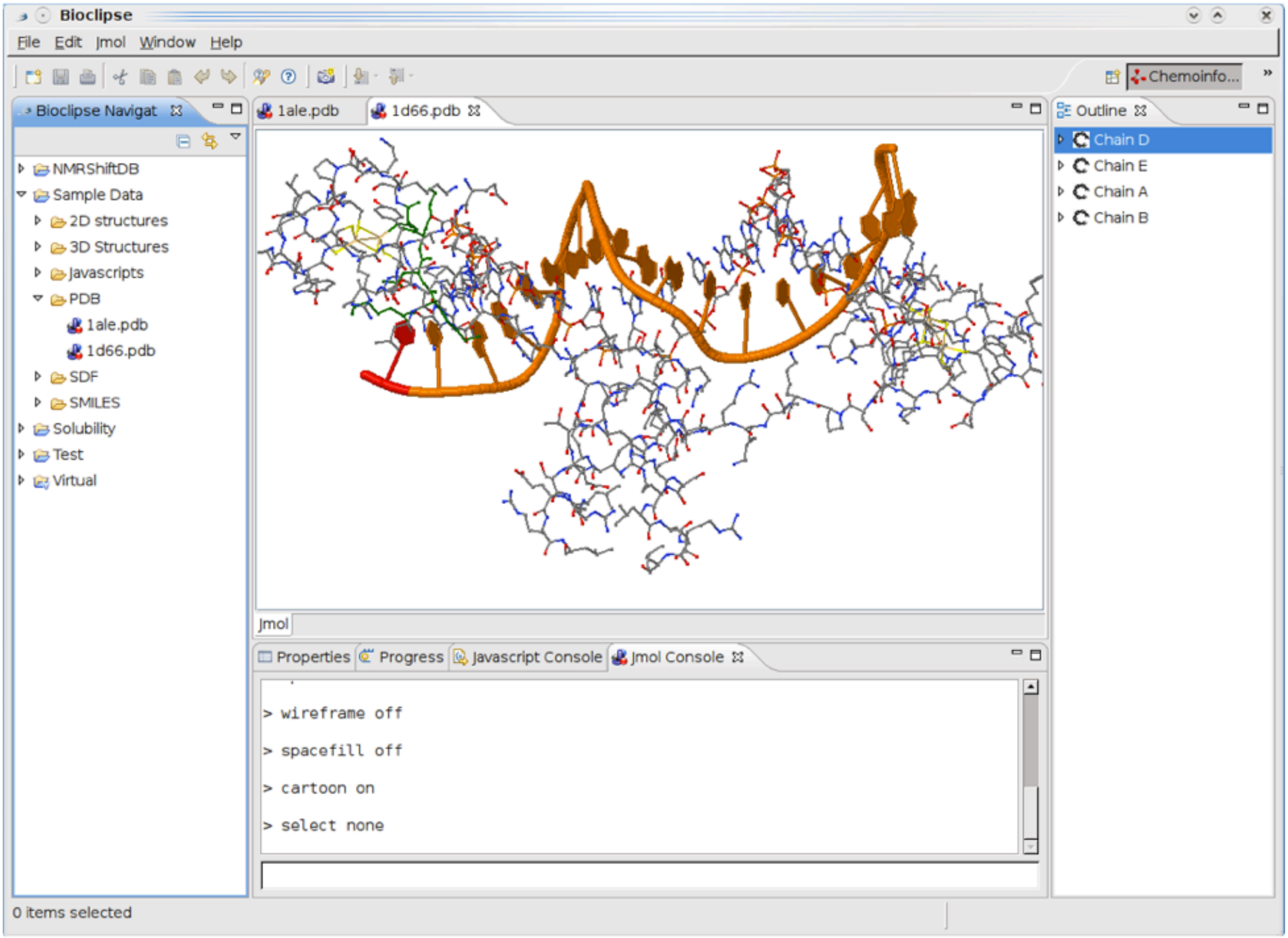
Page: 4 of 5+

Target/Feature

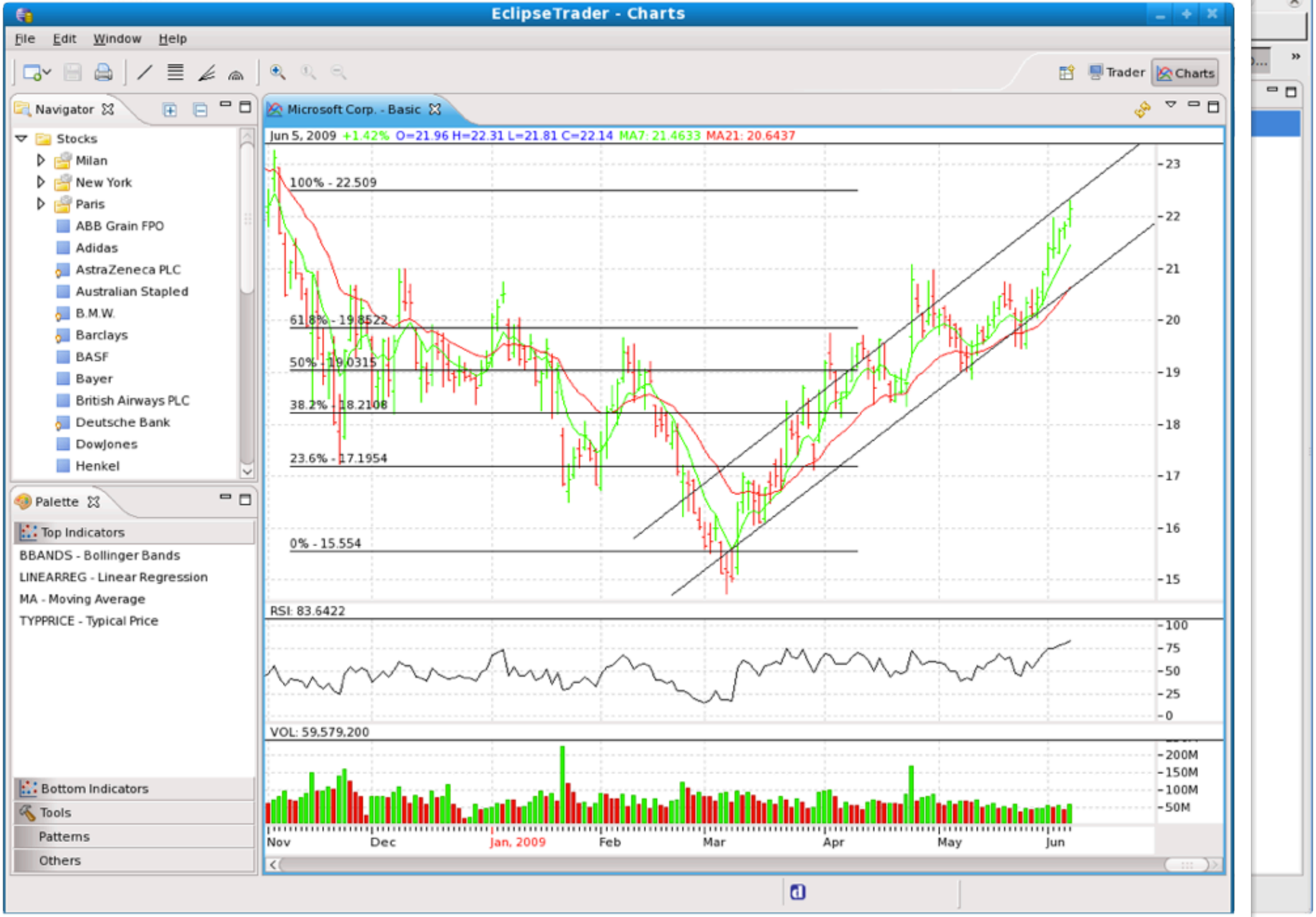
(no target or feature defined)
create target
create feature

Point

Frame: SITE
Distance: 2.516 m (from rover)
Azimuth: 178.426°
Elevation: -28.706°
Location: (-25.644, 51.024, 1.266) m
Normal: (0.062, -0.025, -0.998)
Pixel (i,j): (366.69, 859.91)



```
> wireframe off

> spacefill off

> cartoon on

> select none
```



Jun 5, 2009 +1.42% O=21.96 H=22.31 L=21.81 C=22.14 MA7: 21.4633 MA21: 20.6437

RSI: 83.6422

VOL: 59,579,200

Summary

- Eclipse is an extendable IDE
- Eclipse is a collection of Eclipse Plug-Ins (and Features, and Applications, etc.)
- Eclipse is a Platform to build Rich Clients (RCP)

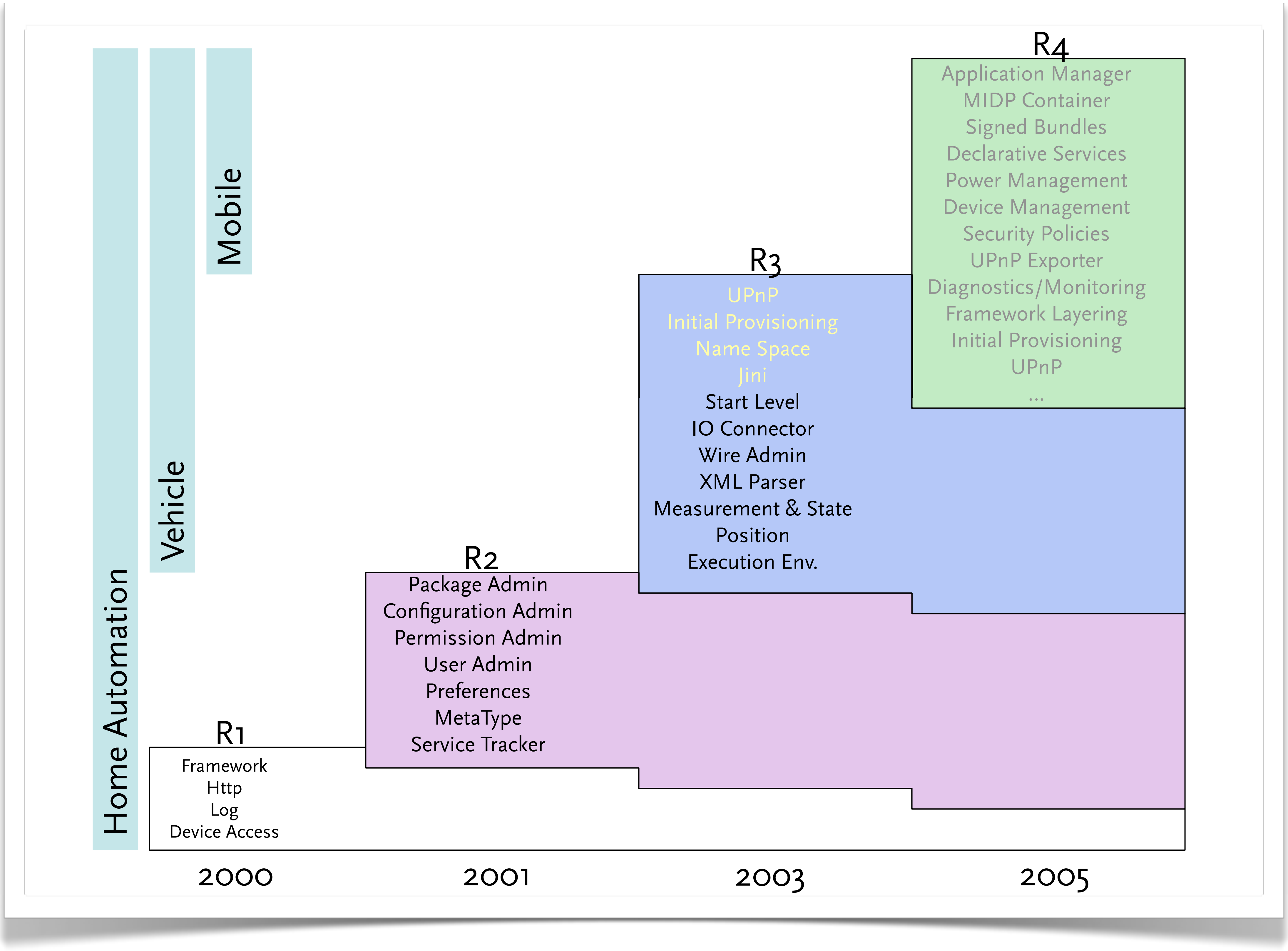
OSGi / Equinox

- Thomas Watson, Peter Kriens: *OSGi™ Component Programming* (EclipseCon, 2006)

What is the OSGi service platform?

- A Java™ framework for developing remotely deployed service applications, that require:
  - Reliability
  - Large scale distribution
  - Wide range of devices
  - Collaborative
- Created through collaboration of industry leaders
- Specifications publicly available at www.osgi.org

Evolution (up to 2006)



Complexity of Software

- A DVD player can contain 1 Million lines of code
  - Comparison: Space Shuttle ~ 0.5 Million
- A BMW car can contain up to 50 networked computerized devices
- Eclipse contains 2.5 million lines of code
- An average programmer writes an average of 10 lines a day

What problems does the OSGi Service Platform address?

in general

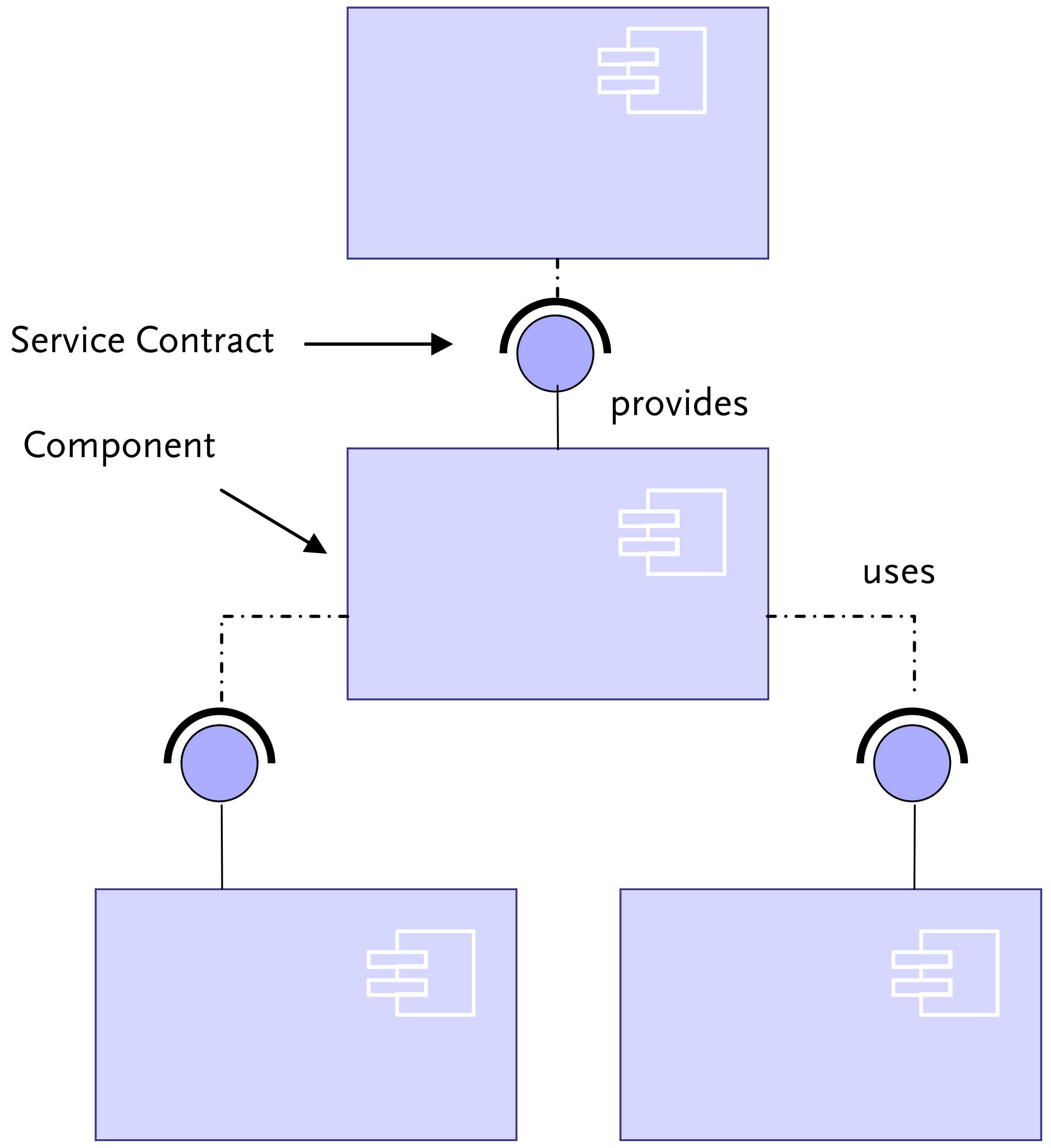
- The limited (binary) software portability problem
- The complexity of building heterogeneous software systems
  - Supporting the myriad of configuration, variations, and customizations required by today's devices
- Managing the software life-cycle on the device

for eclipse

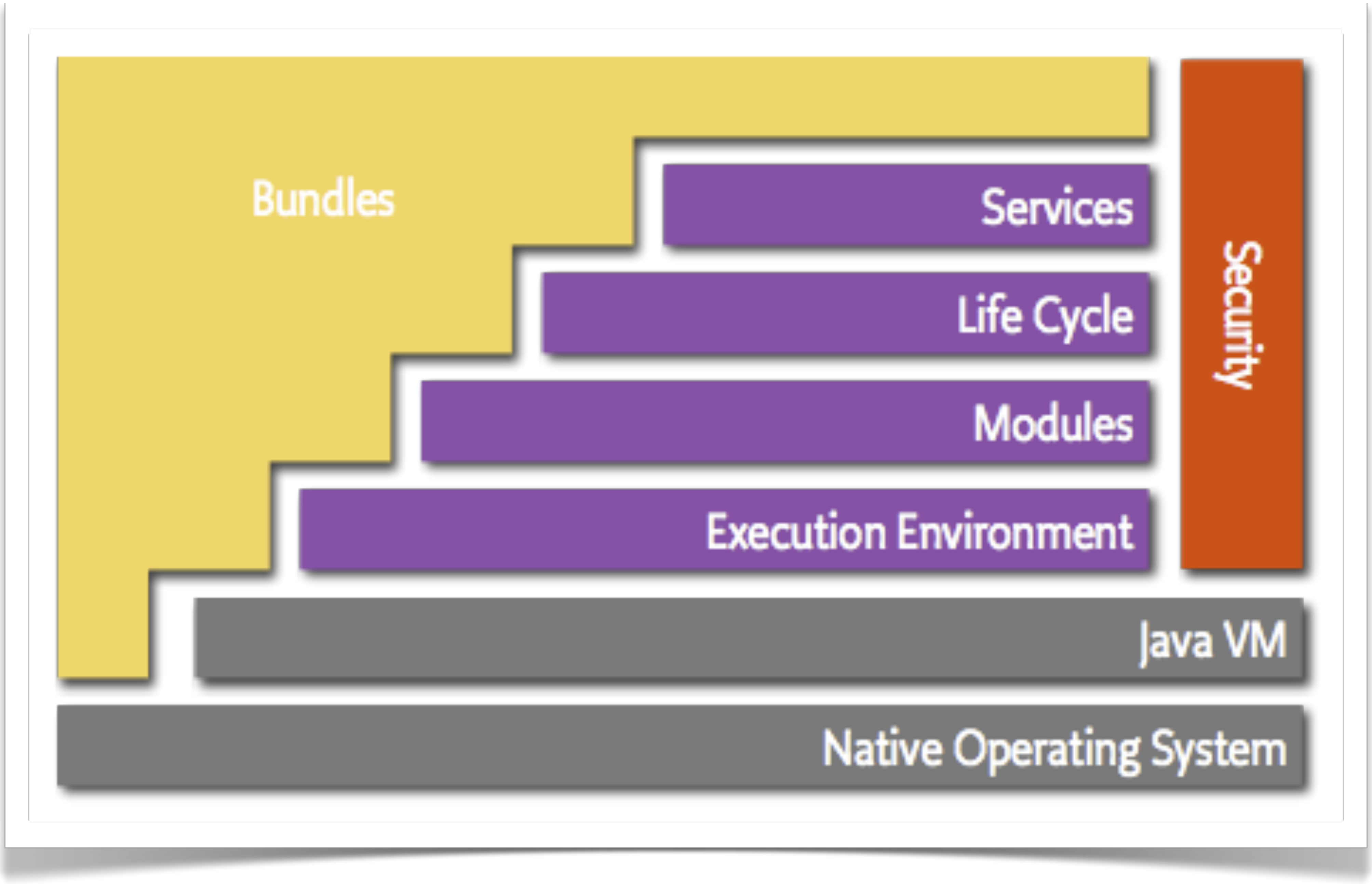
- Eclipse runs on Windows, Linux, MacOS, Unix derivates, 32/64-bit, etc.
- Plug-ins, Plug-ins, Plug-ins
  - different package solutions
  - different plug-in versions in different features
  - 3rd-party plug-ins
  - backward compatibility
- Lazy loading: not all plug-ins need to be started

Service Oriented Architectures

- Separate the contract from the implementation
- Allows alternate implementations
- Dynamically discover and bind available implementations
- Based on contract (interface)
- Components are reusable
- Not coupled to implementation details

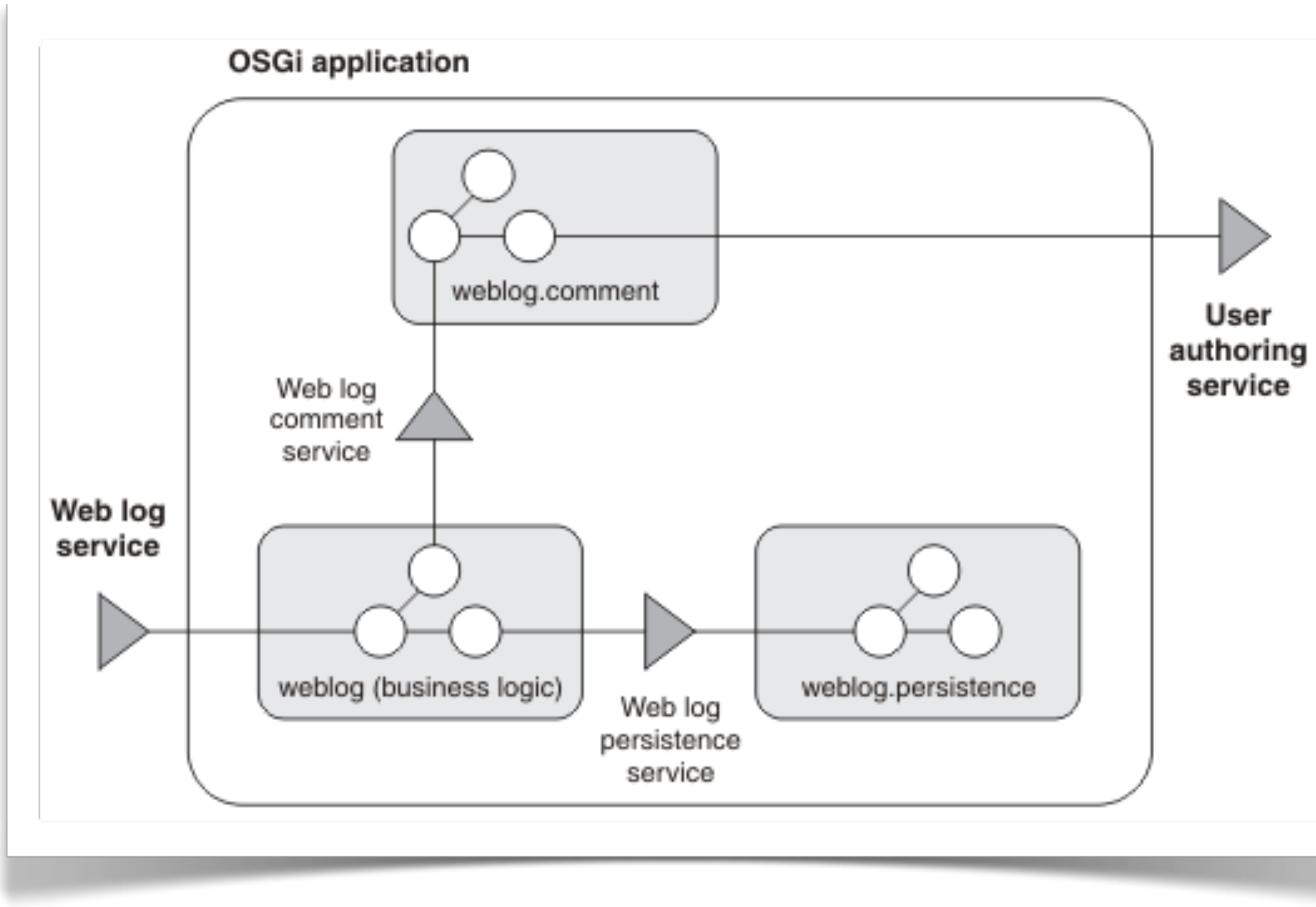


OSGi Feature Layering



Module Layer

- Packaging of applications and libraries in Bundles
  - Raw Java has significant deployment issues
- Class Loading modularization
  - Raw Java provides the Class Path as an ordered search list, which makes it hard to control multiple applications
- Protection
  - Raw Java can not protect certain packages and classes
- Versioning
  - Raw Java can not handle multiple versions of the same package



Module Layer – What is in a Bundle?

- A Bundle contains:
  - Manifest (META-INF/MANIFEST.MF)
  - Code
  - Resources
  - build.properties
- The Framework:
  - Reads the bundle’s manifest
  - Installs the code and resources
  - Resolves dependencies
- During Runtime:
  - Calls the Bundle Activator to start the bundle
  - Manages java classpath
  - Handles the service dependencies
  - Calls the Bundle Activator to stop the bundle

Module Layer – Wha

- A Bundle contains:
  - Manifest (META-INF/MANIFEST.MF)
  - Code
  - Resources
  - build.properties
- The Framework:
  - Reads the bundle's manifest
  - Installs the code and resource
  - Resolves dependencies
- During Runtime:
  - Calls the Bundle Activator to start the bundl
  - Manages java classpath
  - Handles the service dependencies
  - Calls the Bundle Activator to stop the bundle

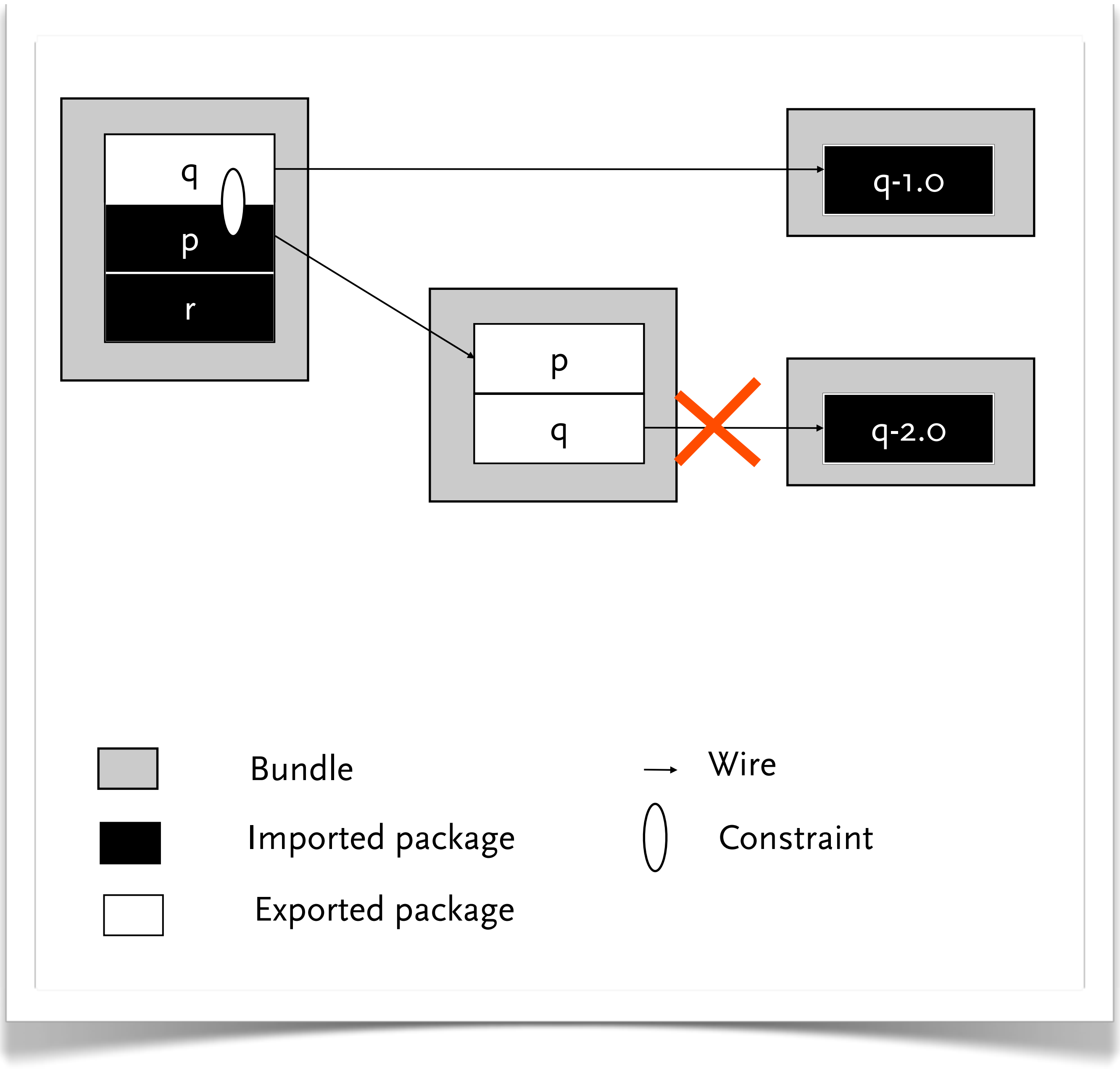
```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Helloworld Plug-in
Bundle-SymbolicName: helloworld
Bundle-Version: 1.0.0
Bundle-Localization: plugin
Bundle-Activator: helloworld.Activator
Import-Package:
 org.osgi.framework;version="1.3.0"
```

```
package helloworld;

public class HelloWorld implements BundleActivator {

   public void start(BundleContext context)
            throws Exception{
      System.out.println("Hello world!!");
   }

   public void stop(BundleContext context)
            throws Exception {
      System.out.println("Goodbye world!!");
      }
}
```

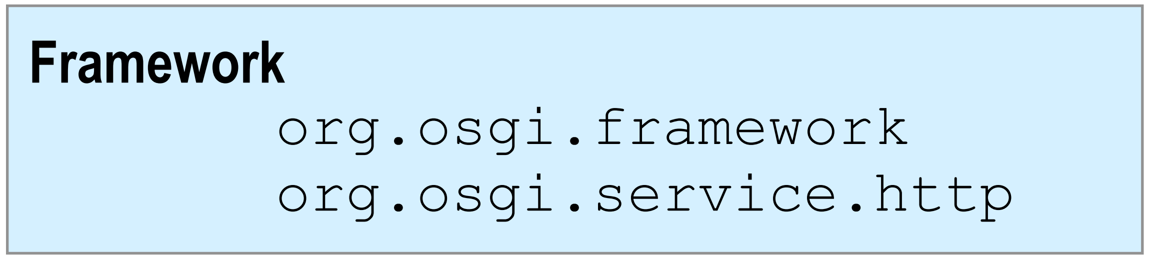
```
source.. = src/
output.. = bin/
bin.includes = META-INF/,\
               .
```

Module layer – Classpath issues

- Java applications consists of classes placed in packages
- Java searches for a package or class in different jar files and directories
  - These are usually specified in the CLASSPATH environment variable
- An OSGi Framework is a network of class loaders.
  - Parameterized by the Manifest headers
- Any dependencies between bundles are resolved by the Framework
- It is possible to fetch bundles on demand
- Complicated – But an OSGi Framework makes it painless to use



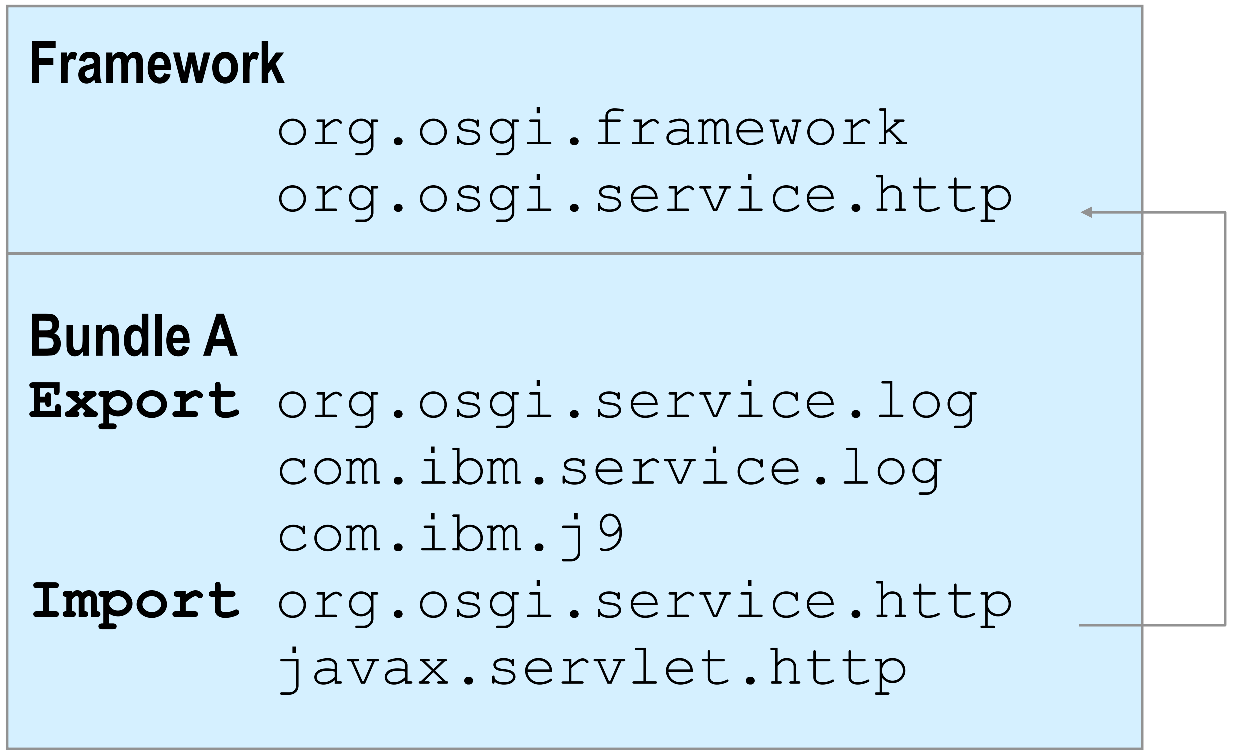
Module layer – OSGi dependency resolution



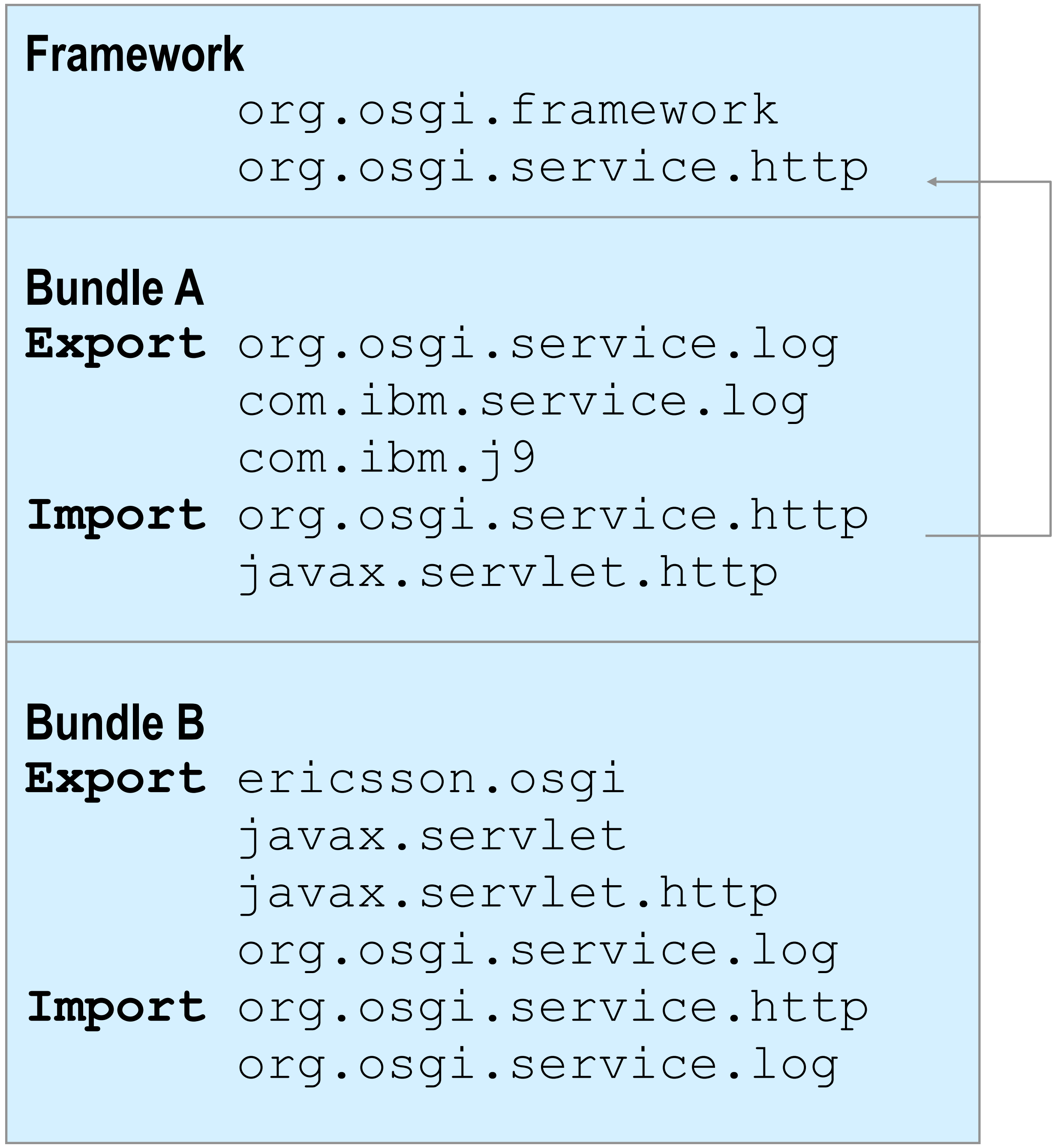
Module layer – OSGi dependency resolution

```
Framework
       org.osgi.framework
       org.osgi.service.http

Bundle A
Export org.osgi.service.log
       com.ibm.service.log
       com.ibm.j9
Import org.osgi.service.http
       javax.servlet.http
```

Module layer – OSGi dependency resolution



Module layer – OSGi dependency resolution

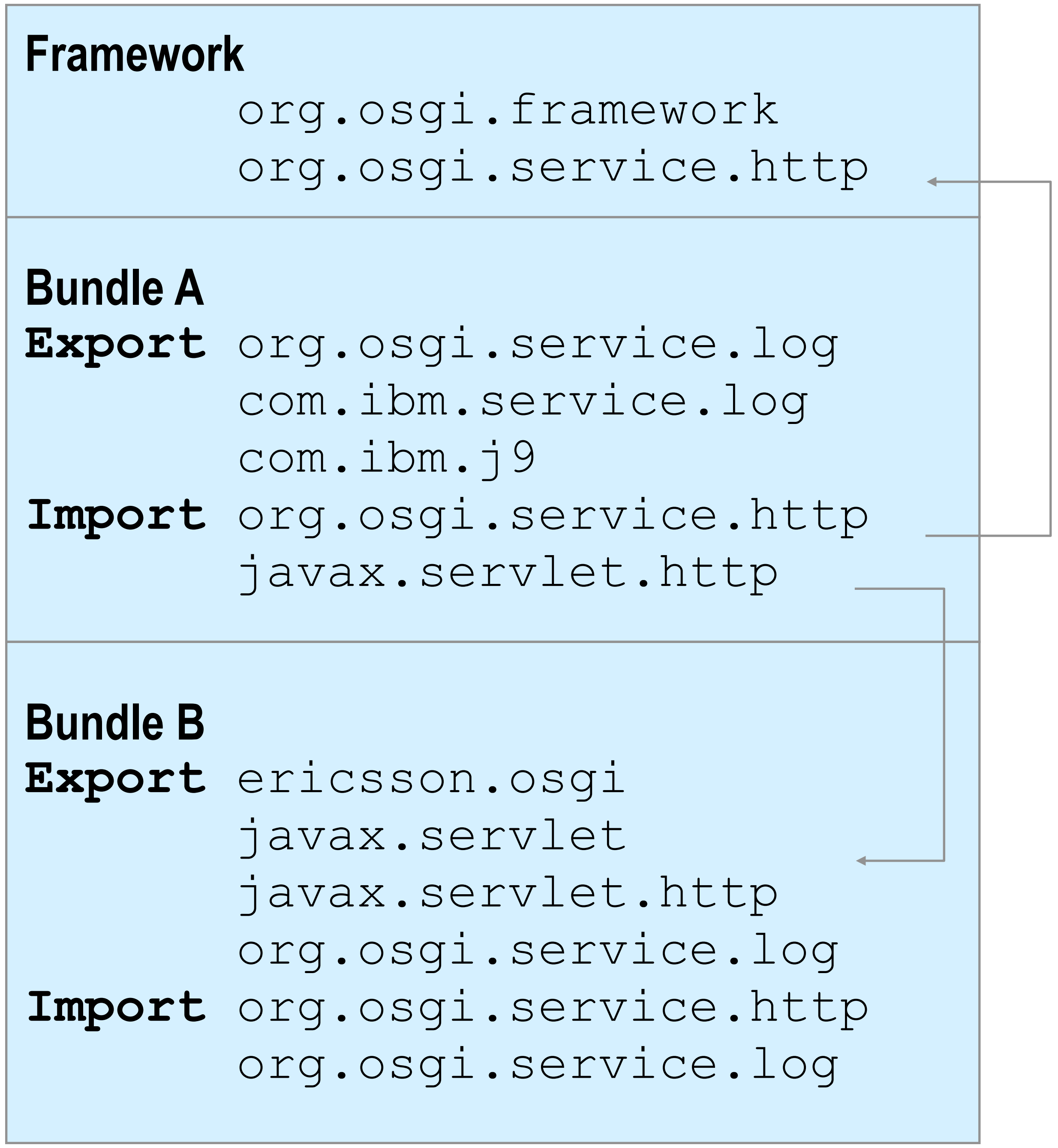


Framework
```
       org.osgi.framework
       org.osgi.service.http
```

Bundle A
```
Export org.osgi.service.log
       com.ibm.service.log
       com.ibm.j9
Import org.osgi.service.http
       javax.servlet.http
```

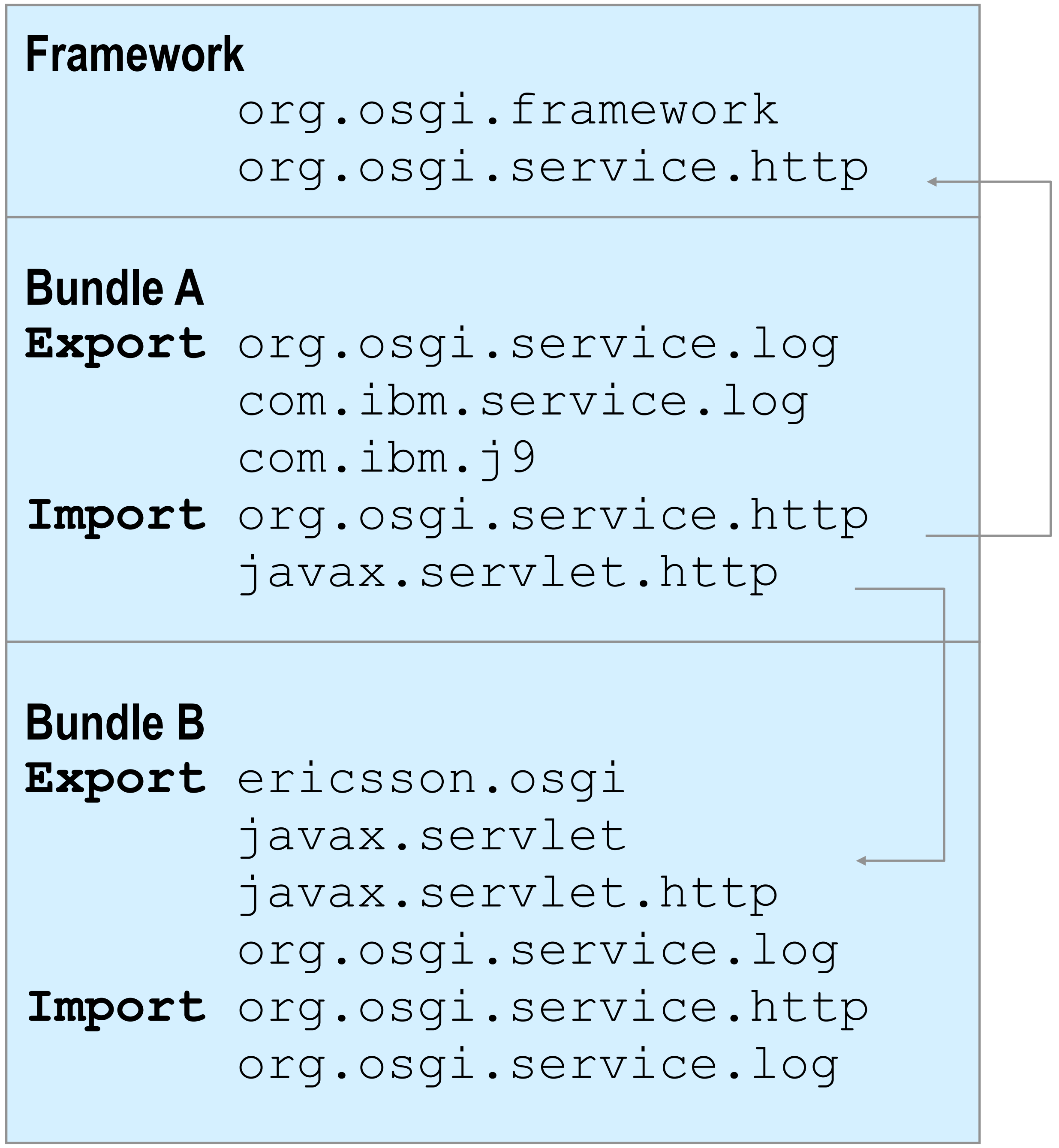
Bundle B
```
Export ericsson.osgi
       javax.servlet
       javax.servlet.http
       org.osgi.service.log
Import org.osgi.service.http
       org.osgi.service.log
```

Module layer – OSGi dependency resolution



```
Framework
        org.osgi.framework
        org.osgi.service.http

Bundle A
Export  org.osgi.service.log
        com.ibm.service.log
        com.ibm.j9
Import  org.osgi.service.http
        javax.servlet.http

Bundle B
Export  ericsson.osgi
        javax.servlet
        javax.servlet.http
        org.osgi.service.log
Import  org.osgi.service.http
        org.osgi.service.log
```

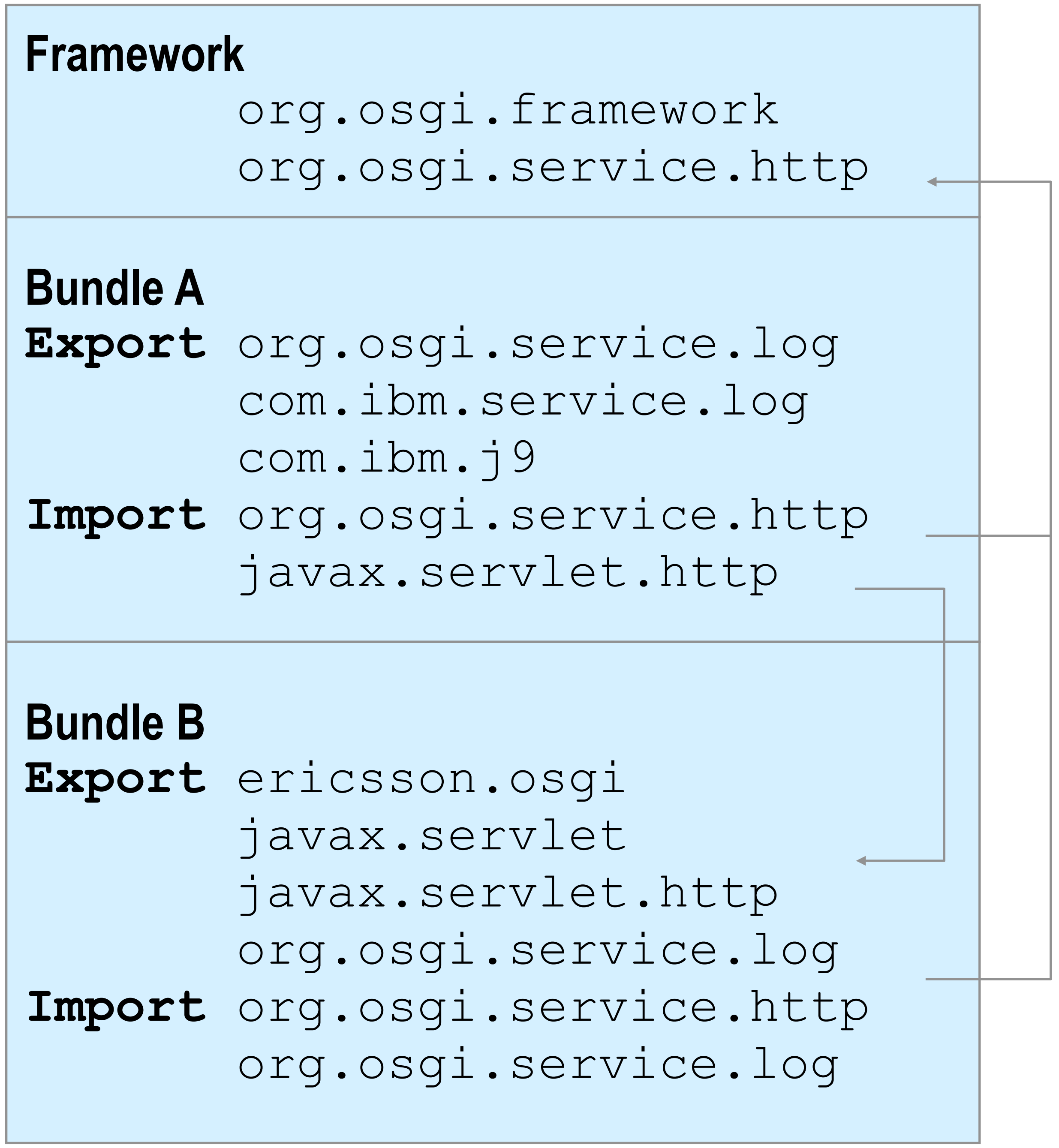
Module layer – OSGi dependency resolution



```
Framework
        org.osgi.framework
        org.osgi.service.http

Bundle A
Export  org.osgi.service.log
        com.ibm.service.log
        com.ibm.j9
Import  org.osgi.service.http
        javax.servlet.http

Bundle B
Export  ericsson.osgi
        javax.servlet
        javax.servlet.http
        org.osgi.service.log
Import  org.osgi.service.http
        org.osgi.service.log
```

A resolved

Module layer – OSGi dependency resolution



Framework

```
        org.osgi.framework
        org.osgi.service.http
```

Bundle A

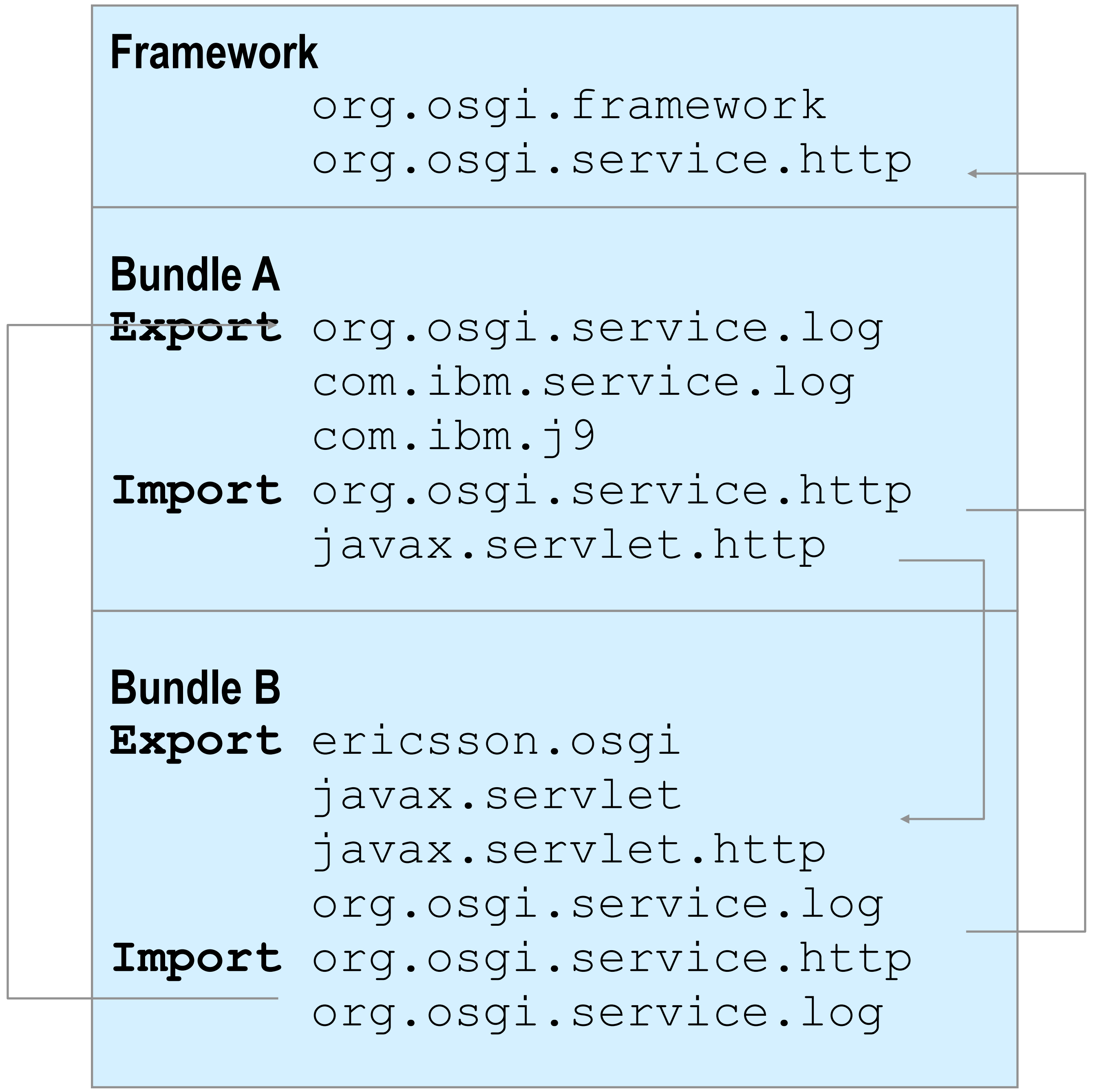
```
Export  org.osgi.service.log
        com.ibm.service.log
        com.ibm.j9
Import  org.osgi.service.http
        javax.servlet.http
```

Bundle B

```
Export  ericsson.osgi
        javax.servlet
        javax.servlet.http
        org.osgi.service.log
Import  org.osgi.service.http
        org.osgi.service.log
```

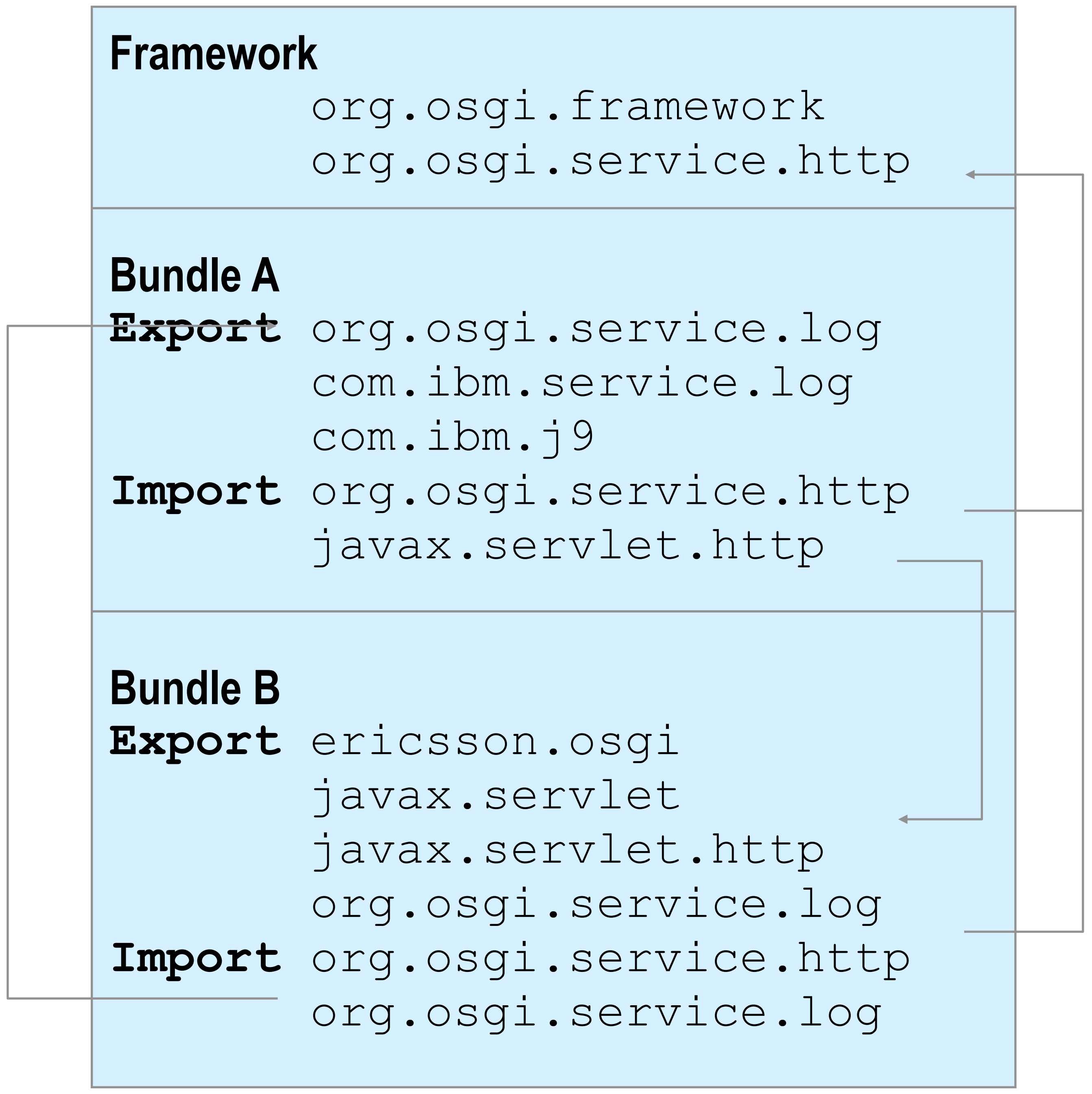
A resolved

Module layer – OSGi dependency resolution



A resolved

Module layer – OSGi dependency resolution

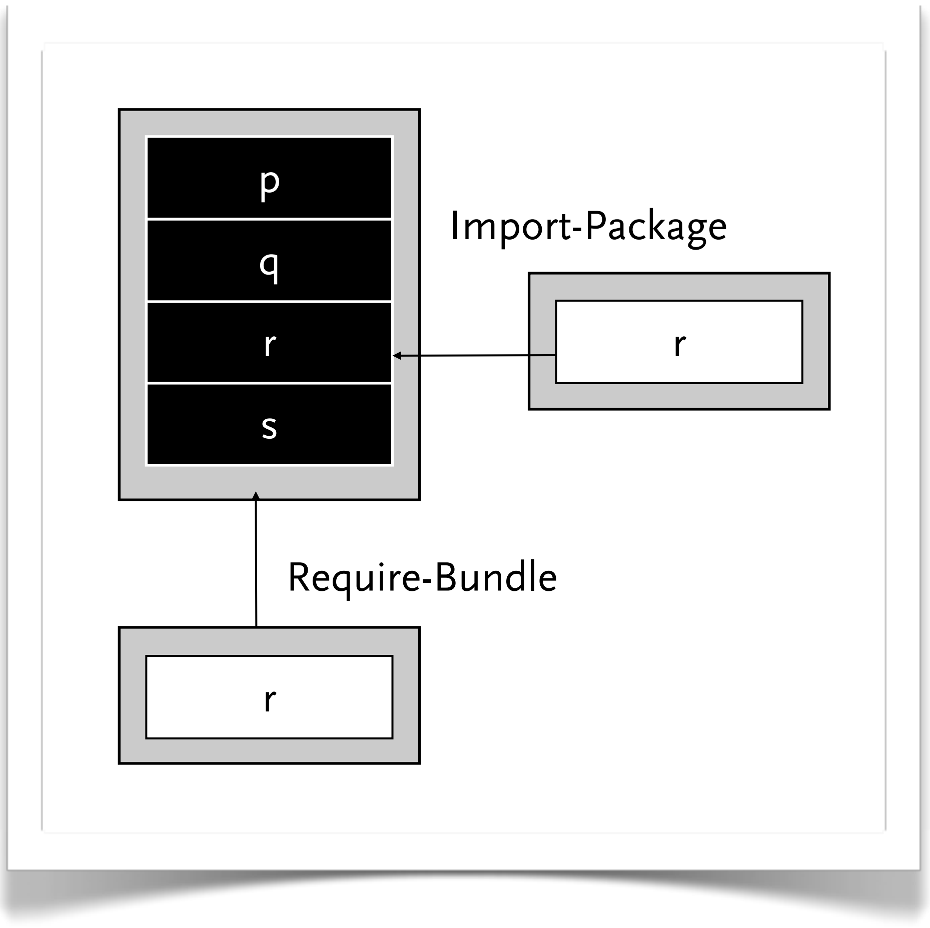


A resolved

B resolved

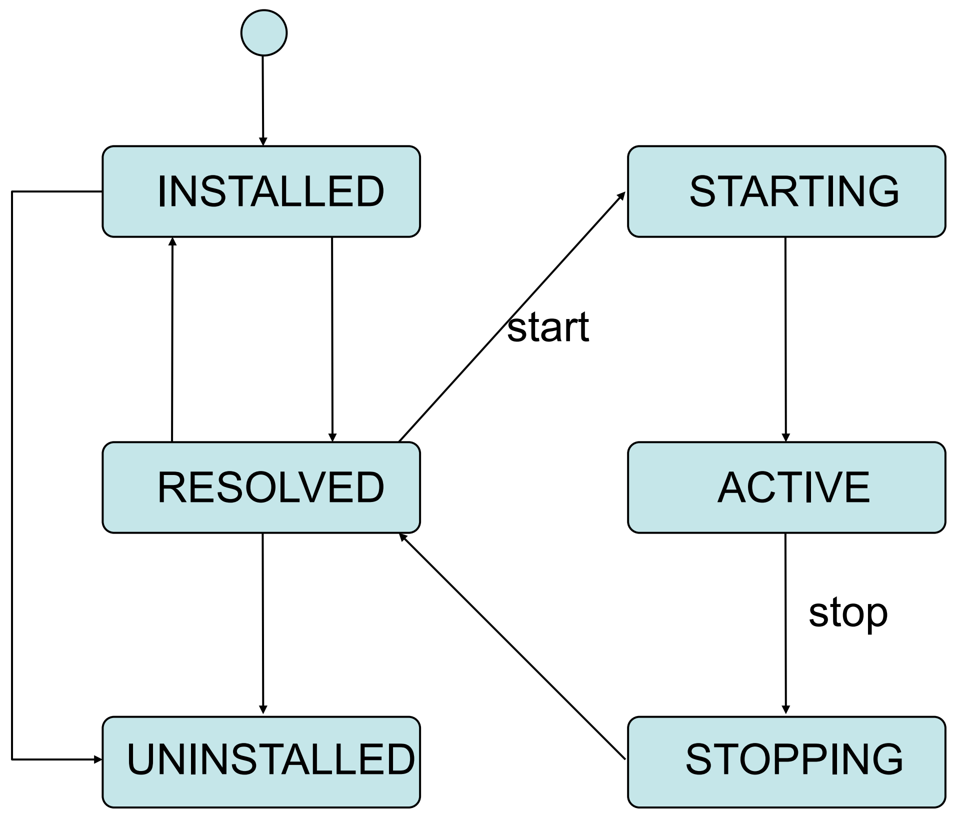
Module layer – Package or Bundle Dependencies?

- The OSGi Specifications supports both Require-Bundle and Import-Package
- Require-Bundle creates a dependency on a complete bundle
  - Simple to use
  - Imports packages that are not used
- Import-Package creates a dependency on just a package
  - Creates less brittle bundles because of substitutability
  - More cumbersome to use (Tools!)
- In almost all cases, Import-Package is recommended because it eases deployment and version migration
- The specifications detail a number of additional problems with Require-Bundle



Life Cycle Layer

- System Bundle represents the OSGi Framework
- Provides an API for managing bundles
  - Install
  - Resolve
  - Start
  - Stop
  - Refresh
  - Update
  - Uninstall
- Based on the module layer



Service Layer

- Provides an in-VM service model
  - Discover (and get notified about) services based on their interface or properties
  - Bind to one or more services by
    - program control,
    - default rules, or
    - deployment configuration
- SOA Confusion
  - Web services bind and discover over the net
  - The OSGi Service Platform binds and discovers inside a Java VM
- The OSGi Alliance provides many standardized services

What is Equinox ?

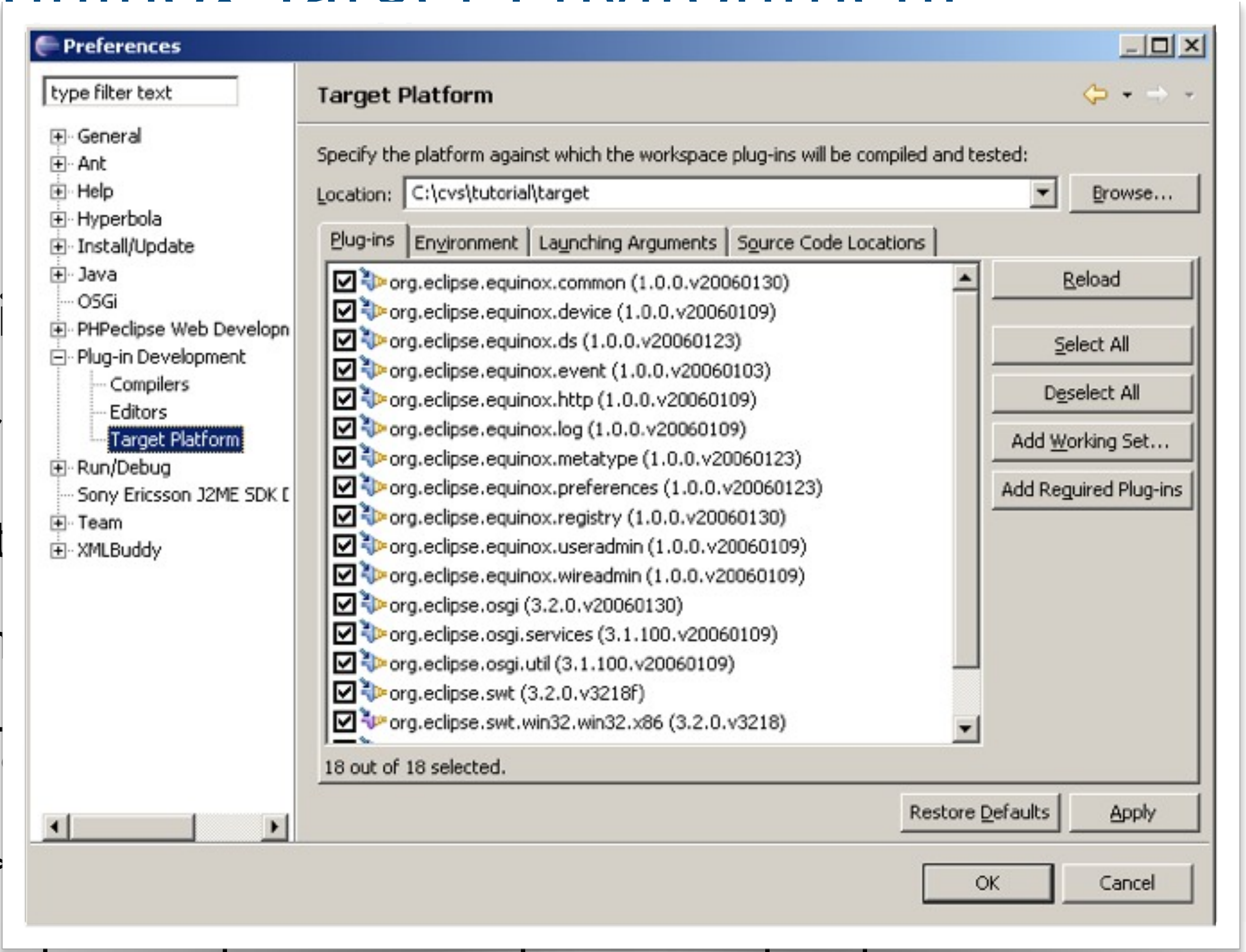
- An open source community focused on OSGi Technology
  - Develop OSGi specification implementations
  - Prototype ideas related to OSGi
- An OSGi Framework implementation
  - Core of the Eclipse runtime
  - Provides the base for Eclipse plug-in collaboration
  - Fully compatible with the OSGi R4 specification

The Equinox Target Environment

- Eclipse makes it easy to develop for OSGi Service Platforms
- A target platform
  - Contains a set of bundles
  - Defines runtime parameters
- To Define the Target Platform, goto:
  - Preferences ->Plug-in Development ->Target Platform
  - Select the target project in your workspace as location

The Equinox Target Environment

- Eclipse makes i
- A target platfor
  - Contains a set
  - Defines runtin
- To Define the T
  - Preferences ->
  - Select the target project in your workspace as location



Target Platform

Specify the platform against which the workspace plug-ins will be compiled and tested:

Location: C:\cvs\tutorial\target

Plug-ins:

- [✓] org.eclipse.equinox.common (1.0.0.v20060130)
- [✓] org.eclipse.equinox.device (1.0.0.v20060109)
- [✓] org.eclipse.equinox.ds (1.0.0.v20060123)
- [✓] org.eclipse.equinox.event (1.0.0.v20060103)
- [✓] org.eclipse.equinox.http (1.0.0.v20060109)
- [✓] org.eclipse.equinox.log (1.0.0.v20060109)
- [✓] org.eclipse.equinox.metatype (1.0.0.v20060123)
- [✓] org.eclipse.equinox.preferences (1.0.0.v20060123)
- [✓] org.eclipse.equinox.registry (1.0.0.v20060130)
- [✓] org.eclipse.equinox.useradmin (1.0.0.v20060109)
- [✓] org.eclipse.equinox.wireadmin (1.0.0.v20060109)
- [✓] org.eclipse.osgi (3.2.0.v20060130)
- [✓] org.eclipse.osgi.services (3.1.100.v20060109)
- [✓] org.eclipse.osgi.util (3.1.100.v20060109)
- [✓] org.eclipse.swt (3.2.0.v3218f)
- [✓] org.eclipse.swt.win32.win32.x86 (3.2.0.v3218)

18 out of 18 selected.

Summary

- The OSGi Service Platform is kind of a Java Operating System
- It simplifies:
  - Deployment Problems
  - Software composition
  - Software management
- Eclipse provides a development environment for OSGi Bundles
- Eclipse provides open source implementations of the OSGi specifications in the Equinox project

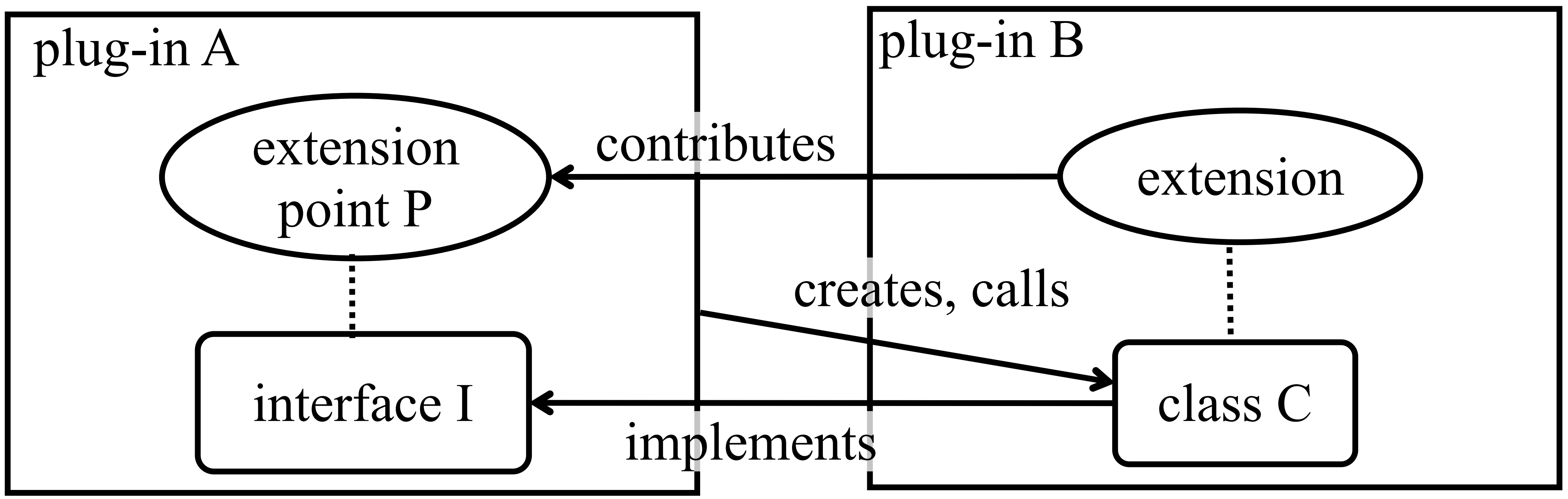
Eclipse Plug-in Architecture

- North Western University, Boston, MA: *Introduction to Eclipse plugin development*

Eclipse Plug-in Architecture

- Plug-in - smallest unit of Eclipse function
  - A special type of OSGi Bundle
  - Large example: Java Editor
  - Small example: Action to create zip files
- Extension point: named entity for collecting “contributions”
  - Example: extension point for workbench text editor.
- Extension: a contribution
  - Example: extending the text editor for domain specific language with syntax highlighting and semantics checking.

Extension and Extension-Points (I)



- Plug-in A
  - Declares extension point P
  - Declares interface I to go with P
- Plug-in B
  - Implements interface I with its own class C
  - Contributes class C to extension point P
- Plug-in A instantiates C and calls its I methods

Extension and Extension-Points (II)

- Each plug-in
  - Contributes to 1 or more extension points
  - Optionally declares new extension points
  - Depends on a set of other plug-ins
  - Contains Java code libraries and other files
  - May export Java-based APIs for downstream plug-ins
  - Lives in its own plug-in subdirectory
- Details spelled out in the plug-in manifest
  - Manifest declares contributions
  - Code implements contributions and provides API
  - plugin.xml file in root of plug-in subdirectory

Extension and Extension-Points (III)

The extension-point defines the contract (markup and code) for the extensions

Extension point declaration – plugin.xml

```
<extension-point id="views"
              name="Views"
              schema="schema/views.exsd"/>
```

Extension declaration – plugin.xml

```
<extension id="catalogView"
        point="org.eclipse.ui.views">
  <view name="Catalog"
       icon="icons/catview.gif"
       class="com.acme.CatalogView"/>
</extension>
```

Metaphor: disc spindle

Plug-in.xml

```
<plugin
      id = “com.example.tool"
      name = “Example Plug-in Tool"
      class = "com.example.tool.ToolPlugin">
  <requires>
      <import plugin = "org.eclipse.core.resources"/>
      <import plugin = "org.eclipse.ui"/>
  </requires>
  <runtime>
      <library name = “tool.jar"/>
  </runtime>
  <extension
      point = "org.eclipse.ui.preferencepages">
     <page id = "com.example.tool.preferences"
        icon = "icons/knob.gif"
        title = “Tool Knobs"
        class = "com.example.tool.ToolPreferenceWizard“/>
  </extension>
  <extension-point
      name = “Frob Providers“
      id = "com.example.tool.frobProvider"/>
</plugin>
```

Plug-in.xml

```
<plugin
      id = “com.example.tool"
      name = “Example Plug-in Tool"
      class = "com.example.tool.ToolPlugin">
  <requires>
      <import plugin = "org.eclipse.core.resources"/>
      <import plugin = "org.eclipse.ui"/>
  </requires>
  <runtime>
      <library name = “tool.jar"/>
  </runtime>
  <extension
      point = "org.eclipse.ui.preferencepages">
     <page id = "com.example.tool.preferences"
        icon = "icons/knob.gif"
        title = “Tool Knobs"
        class = "com.example.tool.ToolPreferenceWizard“/>
  </extension>
  <extension-point
      name = “Frob Providers“
      id = "com.example.tool.frobProvider"/>
</plugin>
```

Plug-in identification

Plug-in.xml

```
<plugin
      id = “com.example.tool"
      name = “Example Plug-in Tool"
      class = "com.example.tool.ToolPlugin">
  <requires>
      <import plugin = "org.eclipse.core.resources"/>
      <import plugin = "org.eclipse.ui"/>
  </requires>
  <runtime>
      <library name = “tool.jar"/>
  </runtime>
  <extension
      point = "org.eclipse.ui.preferencepages">
     <page id = "com.example.tool.preferences"
        icon = "icons/knob.gif"
        title = “Tool Knobs"
        class = "com.example.tool.ToolPreferenceWizard“/>
  </extension>
  <extension-point
      name = “Frob Providers“
      id = "com.example.tool.frobProvider"/>
</plugin>
```

Plug-in identification

Other plug-ins needed

Plug-in.xml

```
<plugin
      id = “com.example.tool"
      name = “Example Plug-in Tool"
      class = "com.example.tool.ToolPlugin">
  <requires>
      <import plugin = "org.eclipse.core.resources"/>
      <import plugin = "org.eclipse.ui"/>
  </requires>
  <runtime>
      <library name = “tool.jar"/>
  </runtime>
  <extension
      point = "org.eclipse.ui.preferencepages">
     <page id = "com.example.tool.preferences"
       icon = "icons/knob.gif"
       title = “Tool Knobs"
       class = "com.example.tool.ToolPreferenceWizard“/>
  </extension>
  <extension-point
      name = “Frob Providers“
      id = "com.example.tool.frobProvider"/>
</plugin>
```

Plug-in identification

Other plug-ins needed

Location of plug-in’s code

Plug-in.xml

```
<plugin
      id = “com.example.tool"
      name = “Example Plug-in Tool"
      class = "com.example.tool.ToolPlugin">
  <requires>
      <import plugin = "org.eclipse.core.resources"/>
      <import plugin = "org.eclipse.ui"/>
  </requires>
  <runtime>
      <library name = “tool.jar"/>
  </runtime>
  <extension
      point = "org.eclipse.ui.preferencepages">
     <page id = "com.example.tool.preferences"
       icon = "icons/knob.gif"
       title = “Tool Knobs"
       class = "com.example.tool.ToolPreferenceWizard“/>
  </extension>
  <extension-point
      name = “Frob Providers“
      id = "com.example.tool.frobProvider"/>
</plugin>
```

Plug-in identification

Other plug-ins needed

Location of plug-in’s code

Declare contribution this plug-in makes

Plug-in.xml

```
<plugin
      id = “com.example.tool"
      name = “Example Plug-in Tool"
      class = "com.example.tool.ToolPlugin">
  <requires>
      <import plugin = "org.eclipse.core.resources"/>
      <import plugin = "org.eclipse.ui"/>
  </requires>
  <runtime>
      <library name = “tool.jar"/>
  </runtime>
  <extension
      point = "org.eclipse.ui.preferencepages">
     <page id = "com.example.tool.preferences"
        icon = "icons/knob.gif"
        title = “Tool Knobs"
        class = "com.example.tool.ToolPreferenceWizard“/>
  </extension>
  <extension-point
      name = “Frob Providers“
      id = "com.example.tool.frobProvider"/>
</plugin>
```

Plug-in identification

Other plug-ins needed

Location of plug-in’s code

Declare contribution this plug-in makes

Declare new extension p open to contributions fr other plug-ins

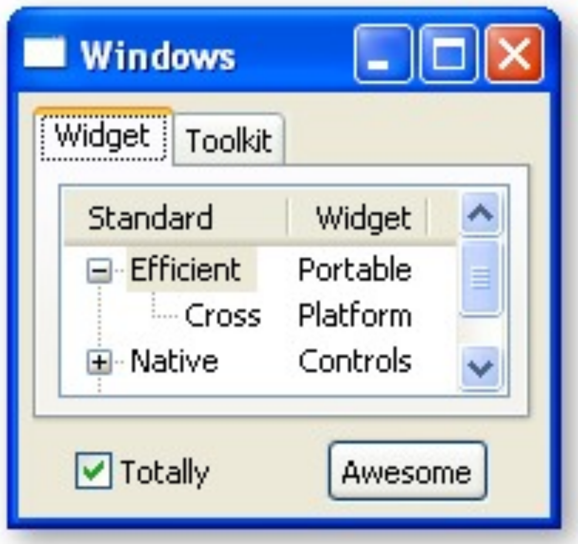
Eclipse Platform

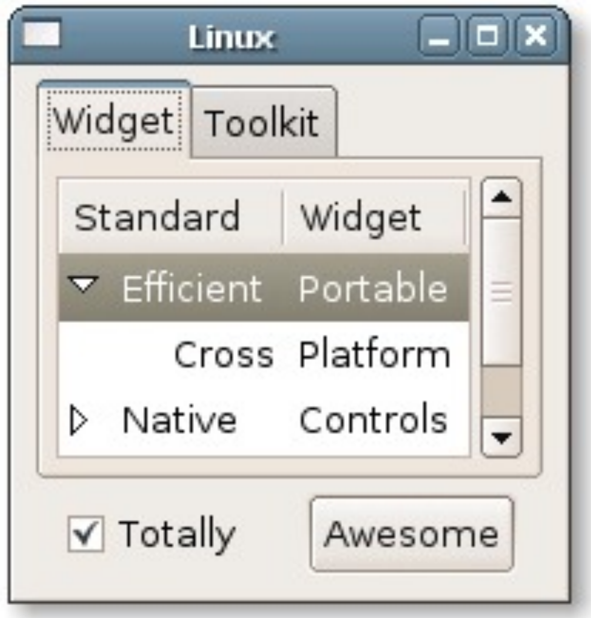
- North Western University, Boston, MA: *Introduction to Eclipse plugin development*

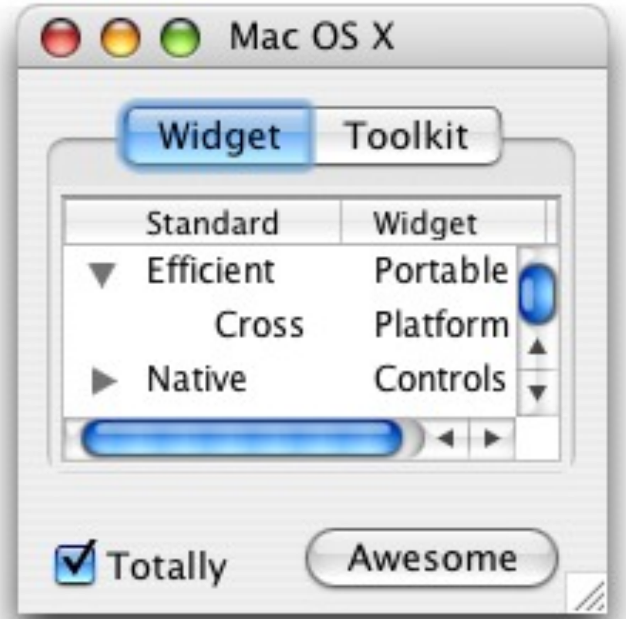
Some Eclipse Platform Components

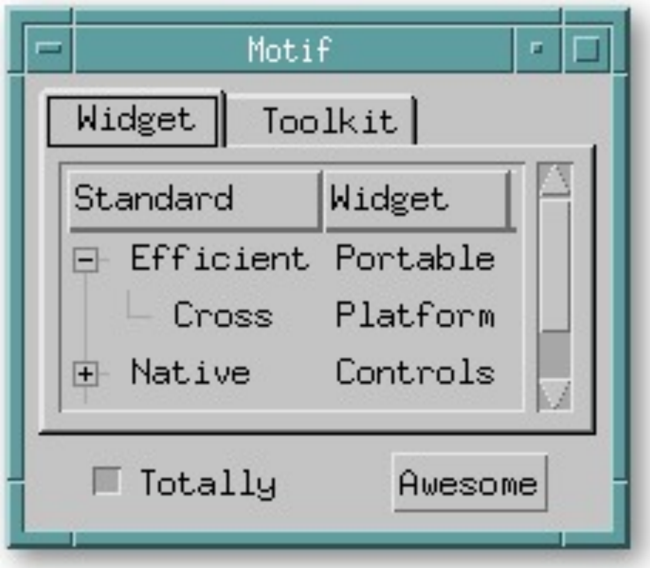
- SWT - Standard Widget Toolkit
- JFace – Framework providing higher-level UI abstractions
- Workbench – Provides reusable and extensible UI metaphors
- Text - Framework(s) for building high-function text editors
- UI Forms - Framework for building forms-based views and editors
- GEF - Framework for building rich graphical editors

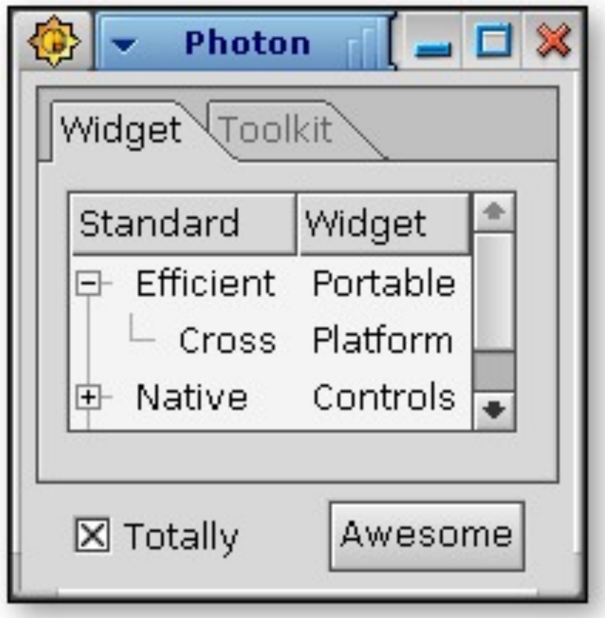
SWT - Standard Widget Toolkit

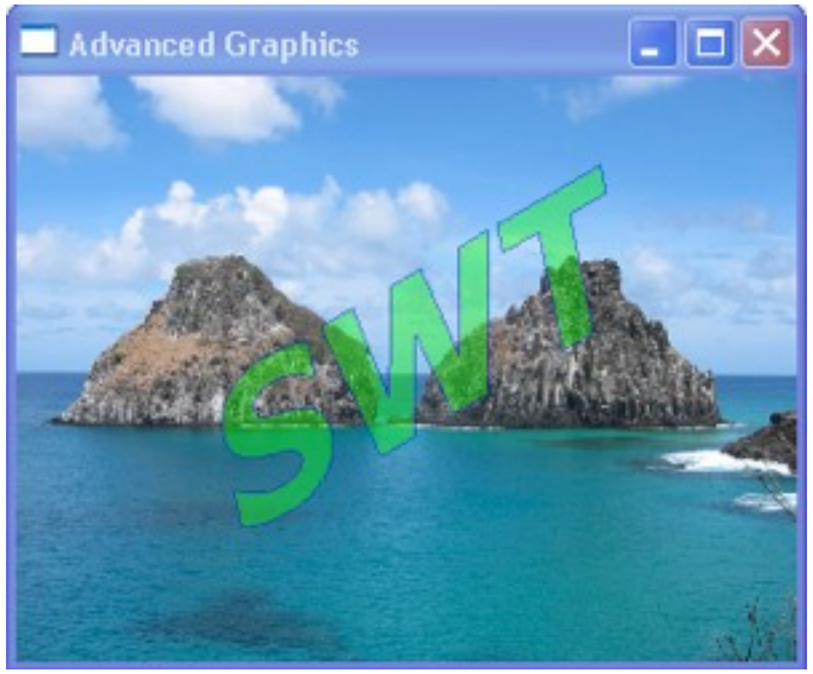








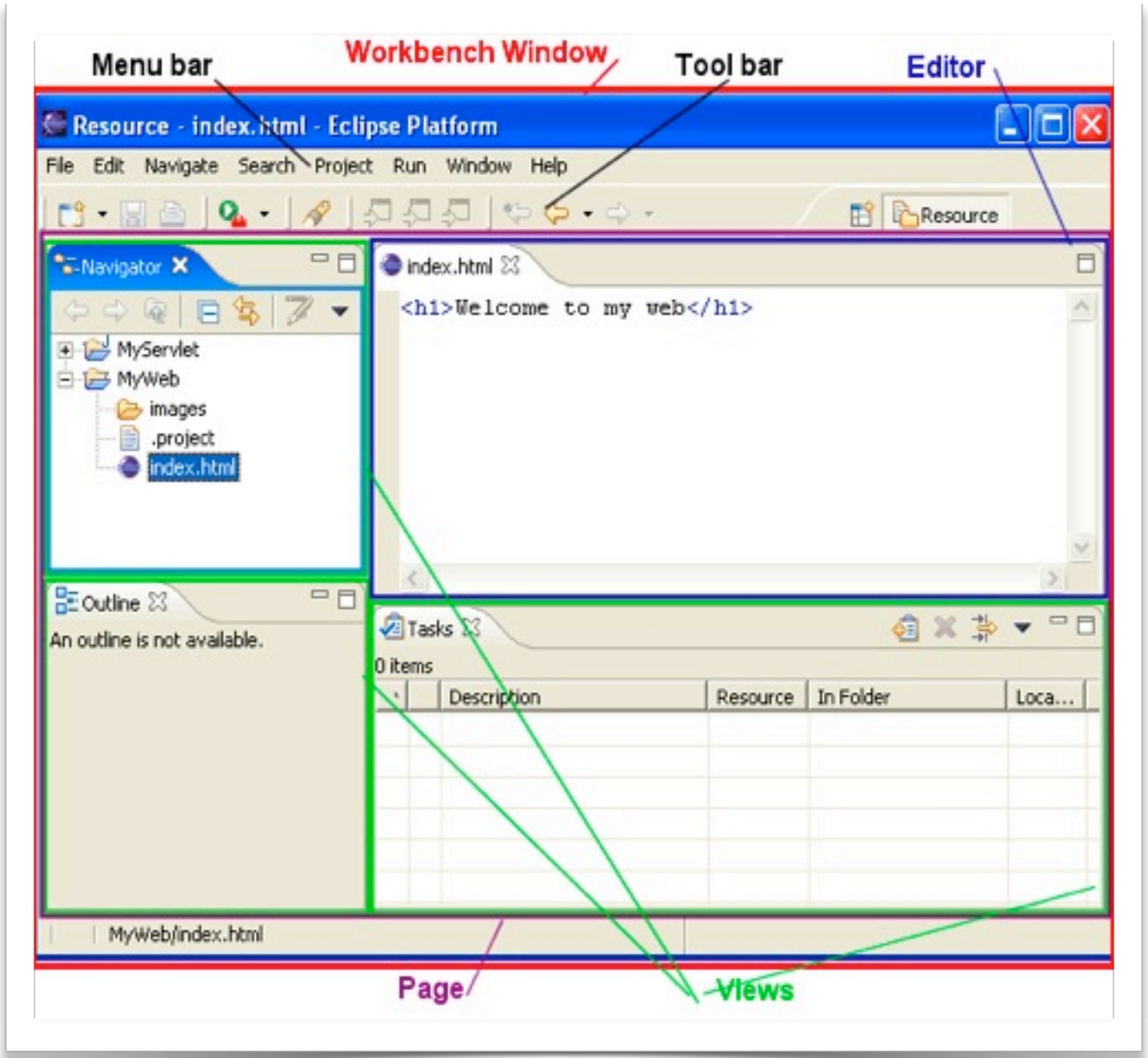




JFace

- Framework on top of SWT providing higher-level UI abstractions
  - Application window: menu bar, tool bar, content area & status line
  - Viewers (MVC pattern)
  - Actions, action bars (abstracts menu items, tool items)
  - Preference and wizard framework

Workbench



```
<h1>Welcome to my web</h1>
```

An outline is not available.

0 items

	Description	Resource	In Folder	Loca...

MyWeb/index.html

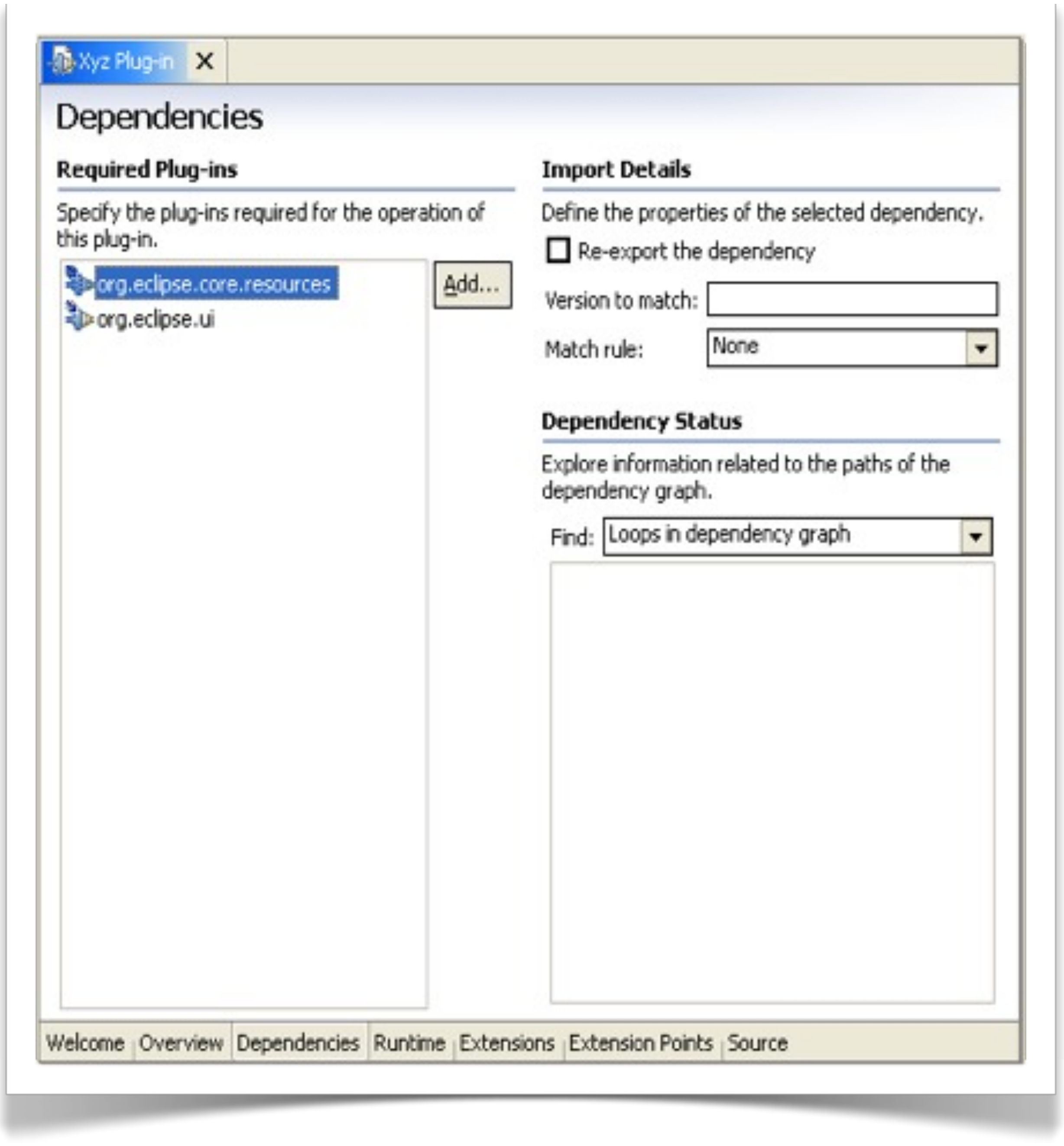
- Defines reusable and extensible UI metaphors
- Leverages extension point mechanism and JFace abstractions.
- Provides:
  - Views
  - Editors
  - Action sets
  - Perspectives
  - Wizards
  - Preference pages
  - Commands and Key Bindings
  - Undo/Redo support
  - Presentations and Themes
  - ...

Text Editor Framework

- Framework(s) for building high-function text editors
  - document infrastructure
    - text manipulation through text edits
    - positions and linked position manager
    - template support
    - projection (aka folding) support
  - source viewer framework
    - provides Text-, SourceViewer and SourceViewerConfiguration
    - concept of annotations, annotations painter, hovers
    - concept of content assist
    - incremental syntax coloring (presentation reconciler)
    - incremental outline update (model reconciler)
    - formatter infrastructure
  - text editor framework
    - leverages source viewer framework for use in workbench editors
    - provides AbstractTextEditor

UI Forms

- Form consisting of multiple FormParts
- Extra widgets:
  - FormText (marked-up text)
  - ScrolledForm
  - Section
  - MasterDetailsBlock
- Extra layouts:
  - TableWrapLayout (HTML-like)
  - ColumnLayout (newspaper-like)
- Flat look, lightweight borders
- Forms-based multi-page editor
- Used extensively in PDE



Specify the plug-ins required for the operation of this plug-in.

org.eclipse.core.resources
org.eclipse.ui

Define the properties of the selected dependency.

☐ Re-export the dependency

Version to match:

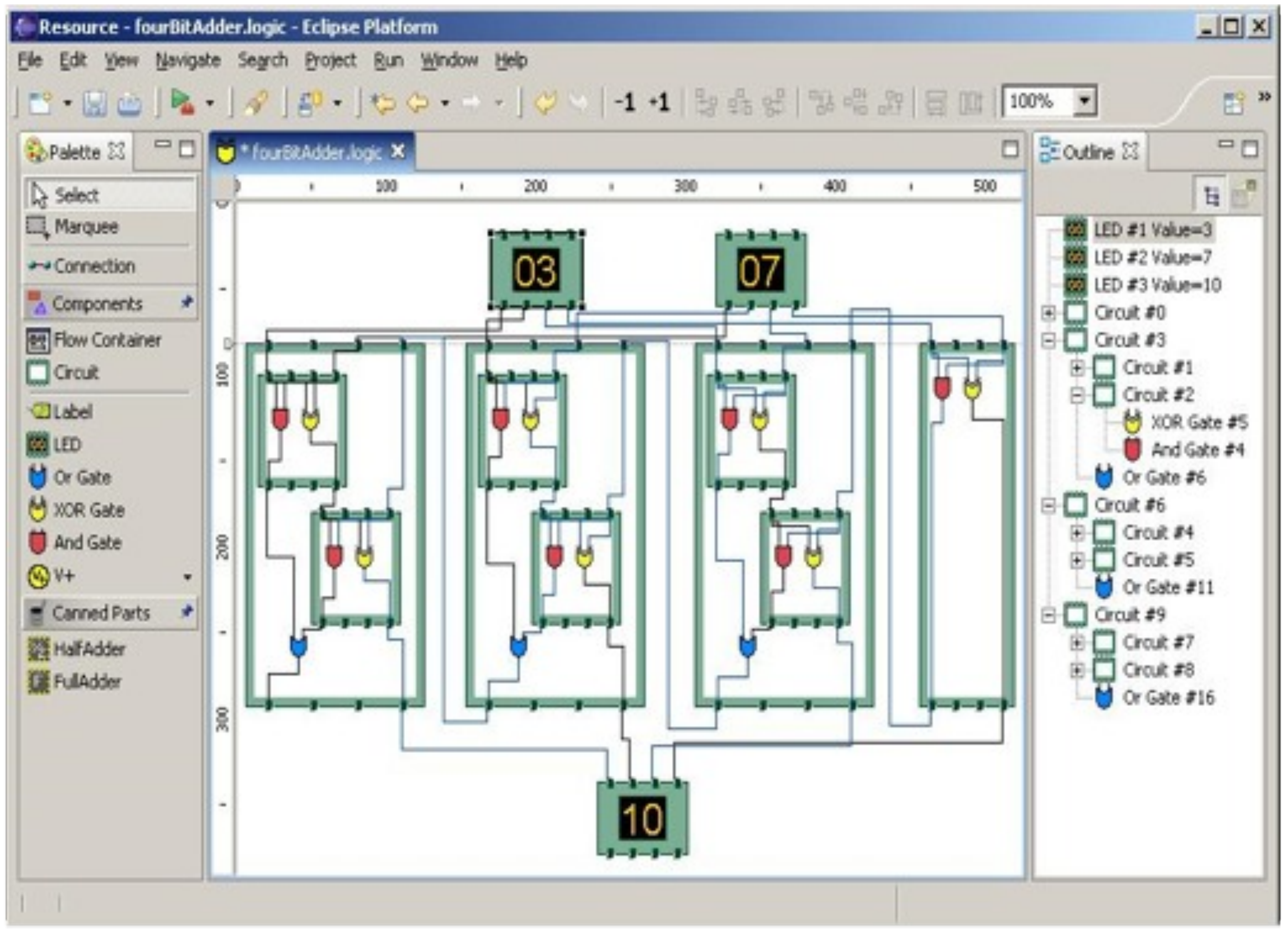
Match rule: None

Explore information related to the paths of the dependency graph.

Find: Loops in dependency graph

GEF (Graphical Editor Framework)

- Framework for building rich graphical editors
  - Draw2D - structured graphics drawing framework
  - Graphical editor framework:
    - MVC architecture
    - Undo/Redo support
    - Palette and common tools
      for manipulating objects
    - Integration with Properties and Outline view



User Assistance Components

- Eclipse Help – Help UI on top of an extensible help content model
- Intro support – Provides the “welcome experience” for your product
- Cheat sheets – Provides guidance through complex tasks

Plug-in Development Environment (PDE)

Plug-in Development Environment

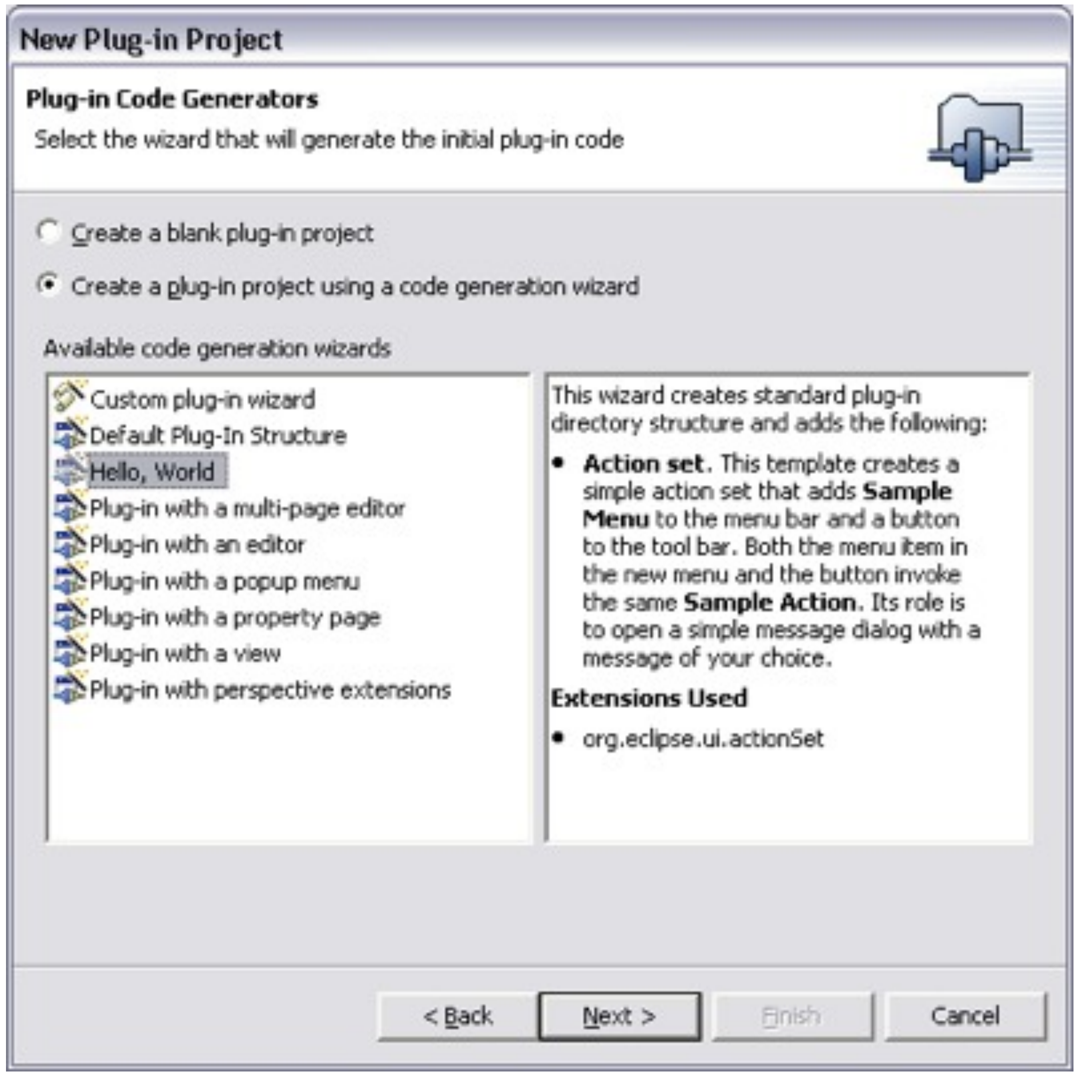
- PDE = Plug-in development environment
- Specialized tools for developing Eclipse plug-ins
- Built atop Eclipse Platform and JDT
  - Implemented as Eclipse plug-ins
  - Using Eclipse Platform and JDT APIs and extension points
- Included in Eclipse Project releases
  - Separately installable feature
  - Part of Eclipse SDK drops

PDE Goals

- To make it easier to develop Eclipse plug-ins
- Support self-hosted Eclipse development

PDE Templates

- PDE templates for creating simple plug-in projects



New Plug-in Project

Plug-in Code Generators

Select the wizard that will generate the initial plug-in code

( ) Create a blank plug-in project

(•) Create a plug-in project using a code generation wizard

Available code generation wizards

- Custom plug-in wizard
- Default Plug-In Structure
- Hello, World
- Plug-in with a multi-page editor
- Plug-in with an editor
- Plug-in with a popup menu
- Plug-in with a property page
- Plug-in with a view
- Plug-in with perspective extensions

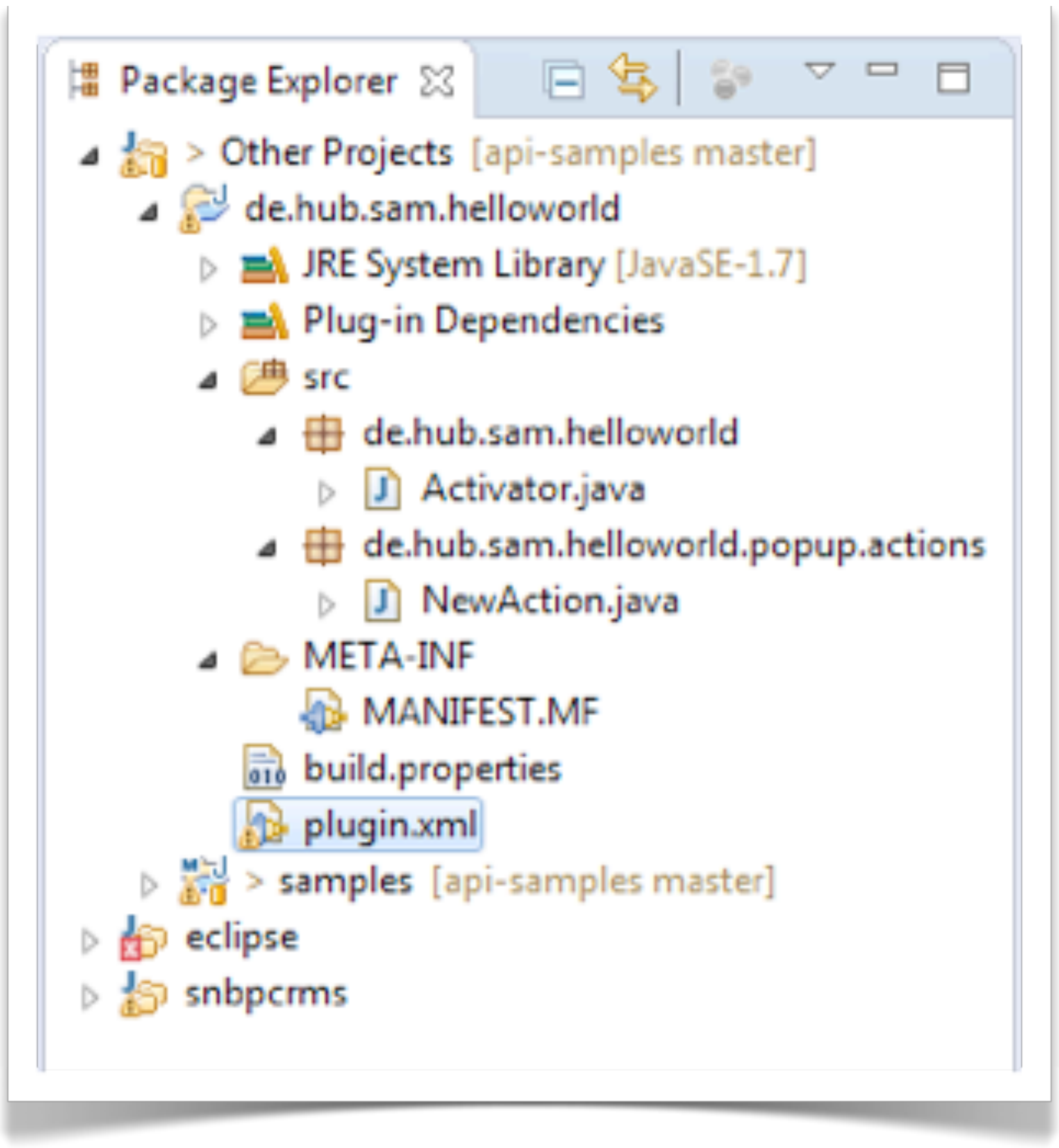
This wizard creates standard plug-in directory structure and adds the following:

- **Action set**. This template creates a simple action set that adds **Sample Menu** to the menu bar and a button to the tool bar. Both the menu item in the new menu and the button invoke the same **Sample Action**. Its role is to open a simple message dialog with a message of your choice.

Extensions Used

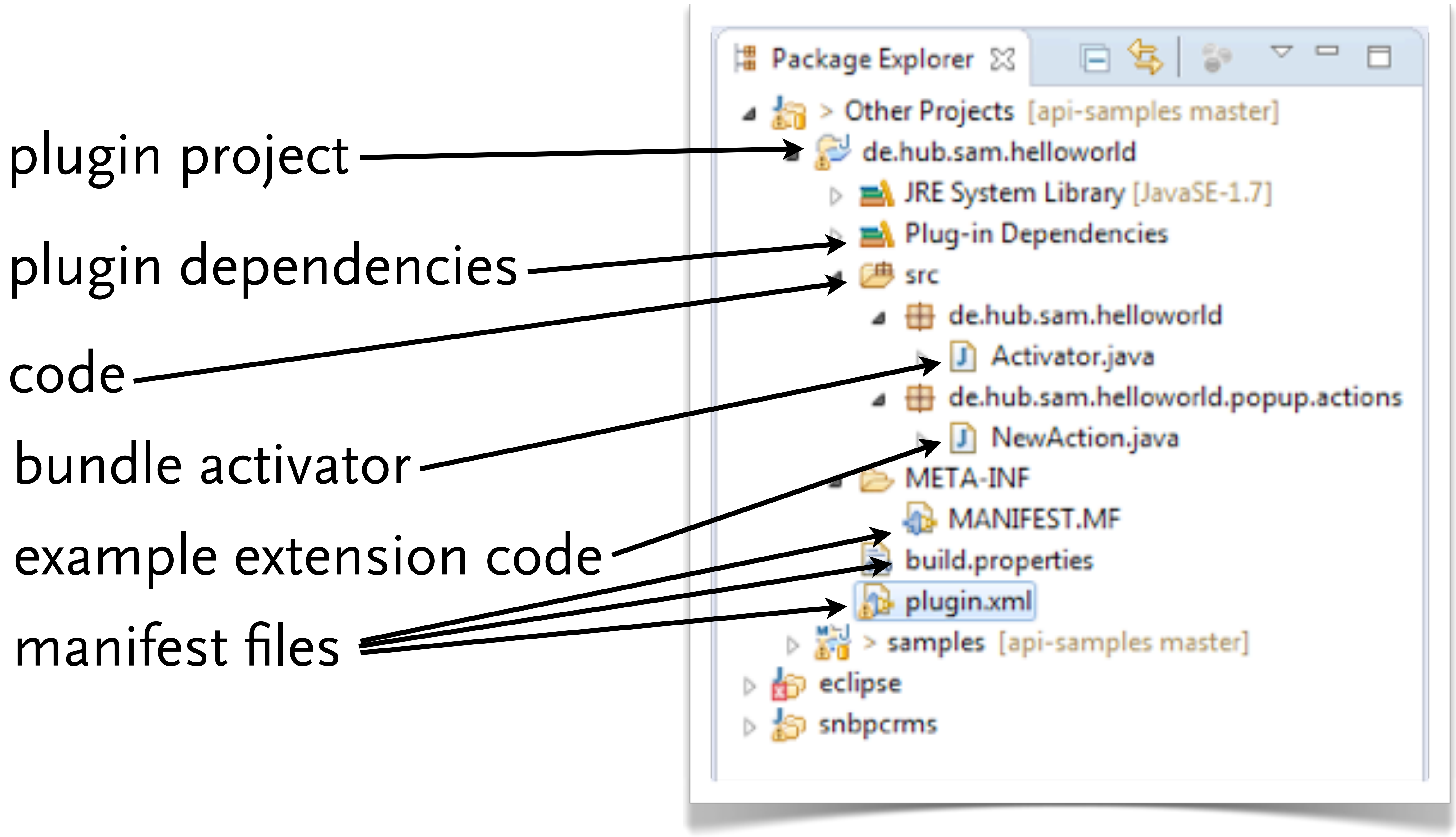
- org.eclipse.ui.actionSet

PDE Plugin Structure


Package Explorer

Other Projects [api-samples master]
  de.hub.sam.helloworld
    JRE System Library [JavaSE-1.7]
    Plug-in Dependencies
    src
      de.hub.sam.helloworld
        Activator.java
      de.hub.sam.helloworld.popup.actions
        NewAction.java
    META-INF
      MANIFEST.MF
    build.properties
    plugin.xml
  samples [api-samples master]
eclipse
snbpcrms

PDE Plugin Structure



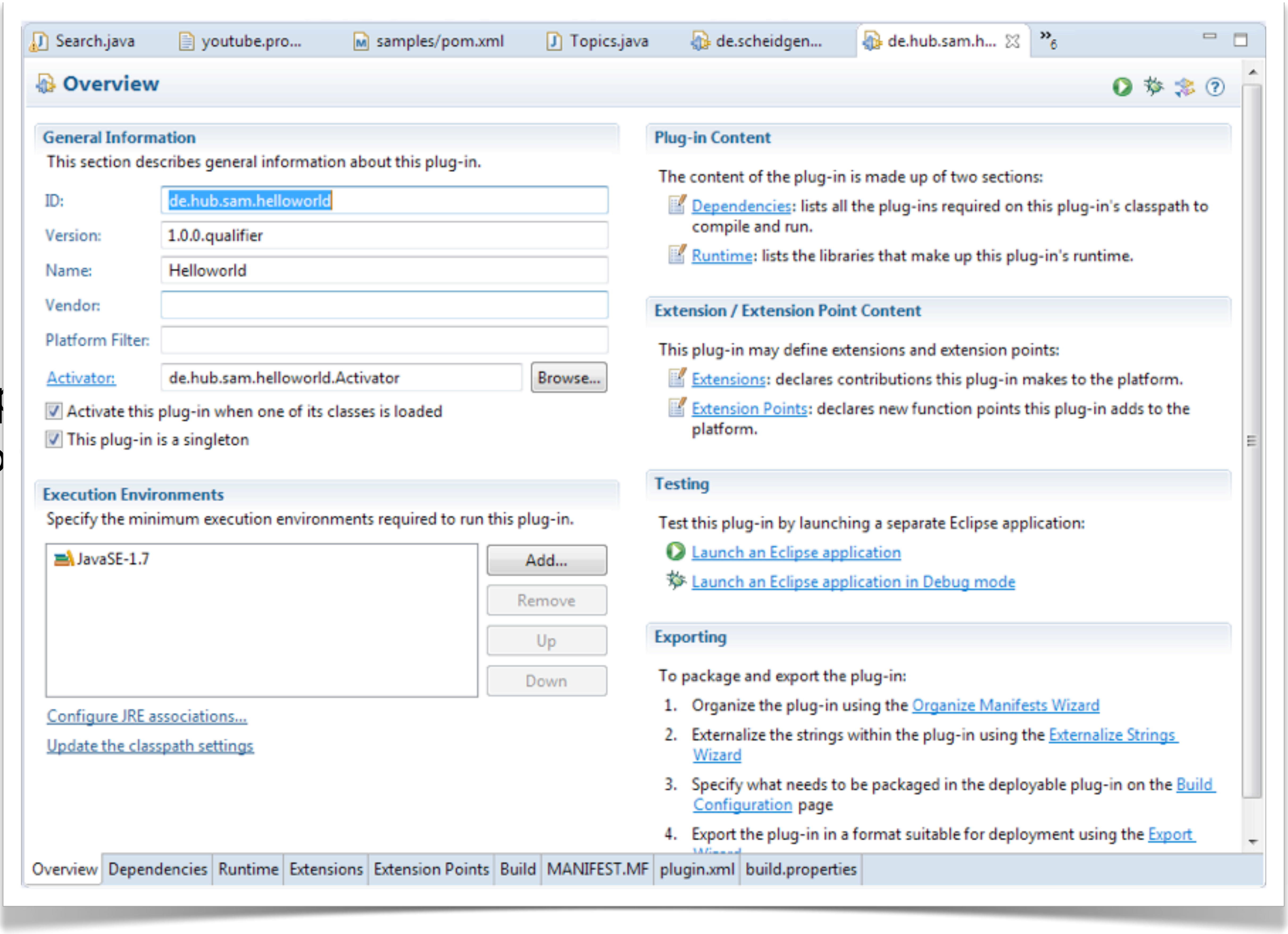
Package Explorer

- Other Projects [api-samples master]
  - de.hub.sam.helloworld
    - JRE System Library [JavaSE-1.7]
    - Plug-in Dependencies
    - src
      - de.hub.sam.helloworld
        - Activator.java
      - de.hub.sam.helloworld.popup.actions
        - NewAction.java
    - META-INF
      - MANIFEST.MF
    - build.properties
    - plugin.xml
  - > samples [api-samples master]
- eclipse
- snbpcrms

PDE Manifest Editor

- Specialized PDE editor
for plug-in manifest files
  - MANIFEST.MF
  - plugin.xml
  - build.properties

▶ Sp
fo


Search.java | youtube.pro... | samples/pom.xml | Topics.java | de.scheidgen... | de.hub.sam.h...

Overview

General Information
This section describes general information about this plug-in.

ID:	de.hub.sam.helloworld
Version:	1.0.0.qualifier
Name:	Helloworld
Vendor:	
Platform Filter:	
Activator:	de.hub.sam.helloworld.Activator

Browse...

☑ Activate this plug-in when one of its classes is loaded
☑ This plug-in is a singleton

Execution Environments
Specify the minimum execution environments required to run this plug-in.

JavaSE-1.7

Add... | Remove | Up | Down

Configure JRE associations...
Update the classpath settings

Plug-in Content
The content of the plug-in is made up of two sections:
Dependencies: lists all the plug-ins required on this plug-in's classpath to compile and run.
Runtime: lists the libraries that make up this plug-in's runtime.

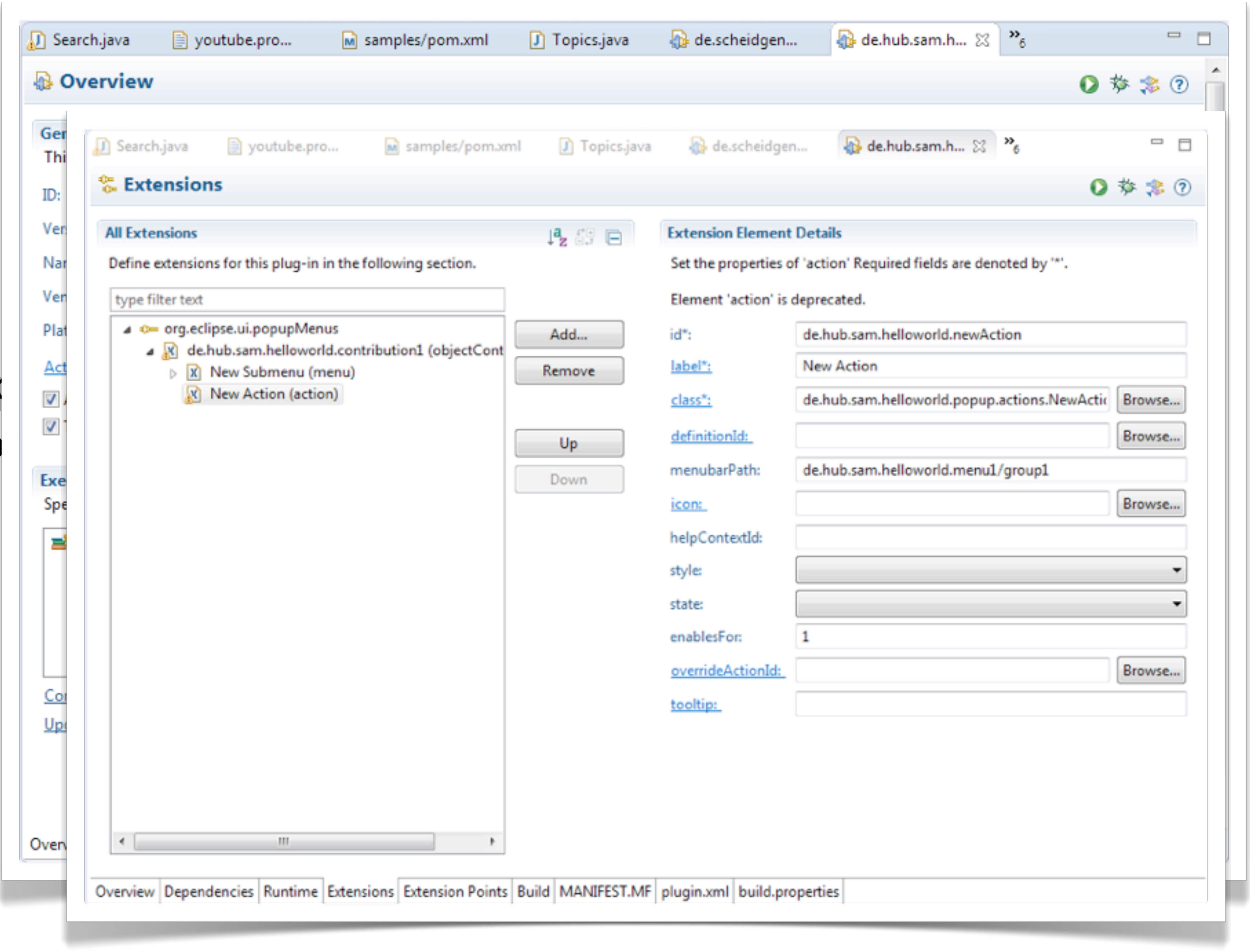
Extension / Extension Point Content
This plug-in may define extensions and extension points:
Extensions: declares contributions this plug-in makes to the platform.
Extension Points: declares new function points this plug-in adds to the platform.

Testing
Test this plug-in by launching a separate Eclipse application:
Launch an Eclipse application
Launch an Eclipse application in Debug mode

Exporting
To package and export the plug-in:
1. Organize the plug-in using the Organize Manifests Wizard
2. Externalize the strings within the plug-in using the Externalize Strings Wizard
3. Specify what needs to be packaged in the deployable plug-in on the Build Configuration page
4. Export the plug-in in a format suitable for deployment using the Export

Overview | Dependencies | Runtime | Extensions | Extension Points | Build | MANIFEST.MF | plugin.xml | build.properties

- Sp
  fo



Search.java | youtube.pro... | samples/pom.xml | Topics.java | de.scheidgen... | de.hub.sam.h...

Overview

Extensions

All Extensions

Define extensions for this plug-in in the following section.

type filter text

- org.eclipse.ui.popupMenus
  - de.hub.sam.helloworld.contribution1 (objectCont
    - New Submenu (menu)
    - New Action (action)

Add... | Remove | Up | Down

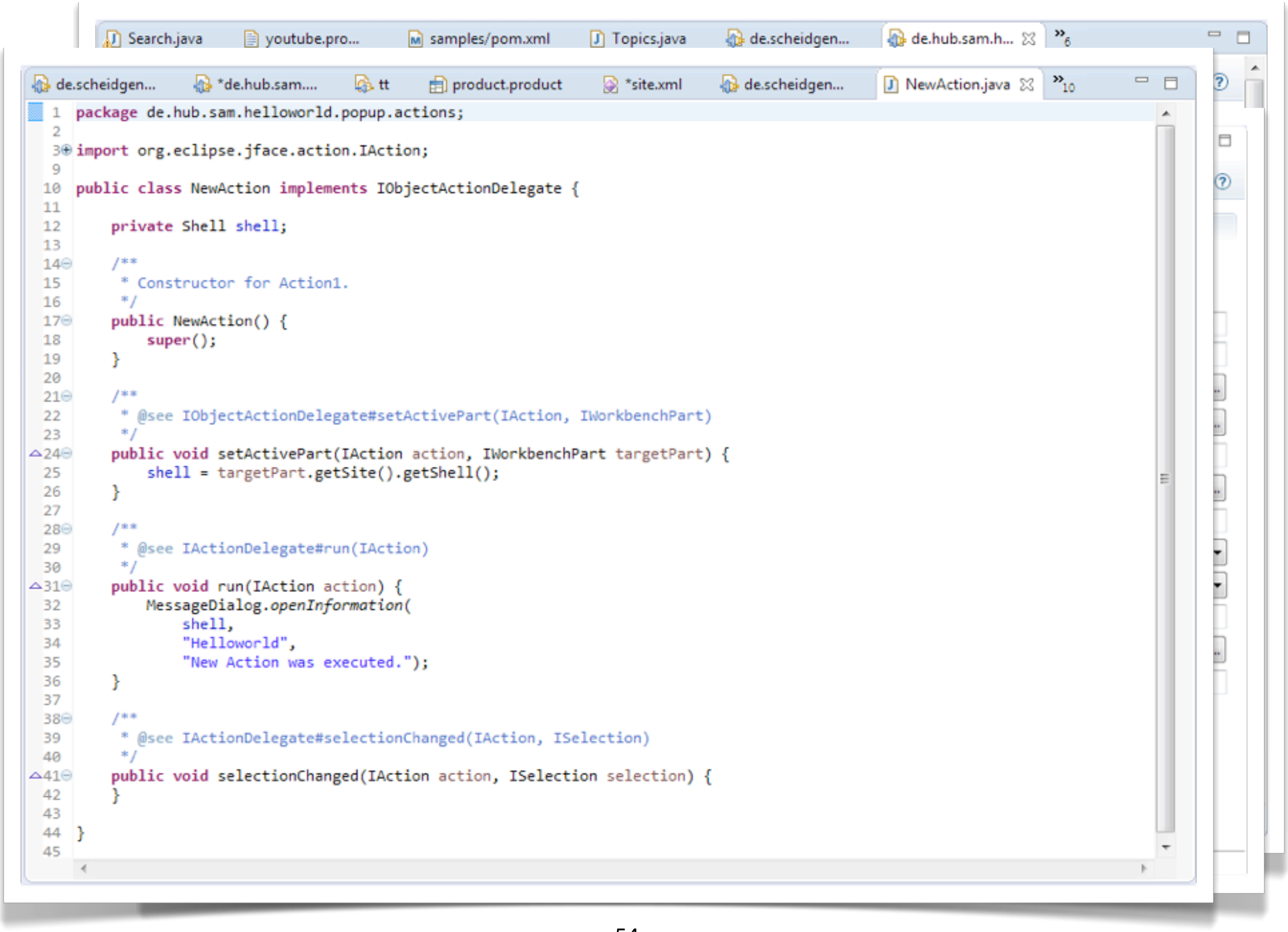
Extension Element Details

Set the properties of 'action' Required fields are denoted by "*".

Element 'action' is deprecated.

id*:	de.hub.sam.helloworld.newAction
label*:	New Action
class*:	de.hub.sam.helloworld.popup.actions.NewActic
definitionId:	
menubarPath:	de.hub.sam.helloworld.menu1/group1
icon:	
helpContextId:	
style:	
state:	
enablesFor:	1
overrideActionId:	
tooltip:	

Overview | Dependencies | Runtime | Extensions | Extension Points | Build | MANIFEST.MF | plugin.xml | build.properties



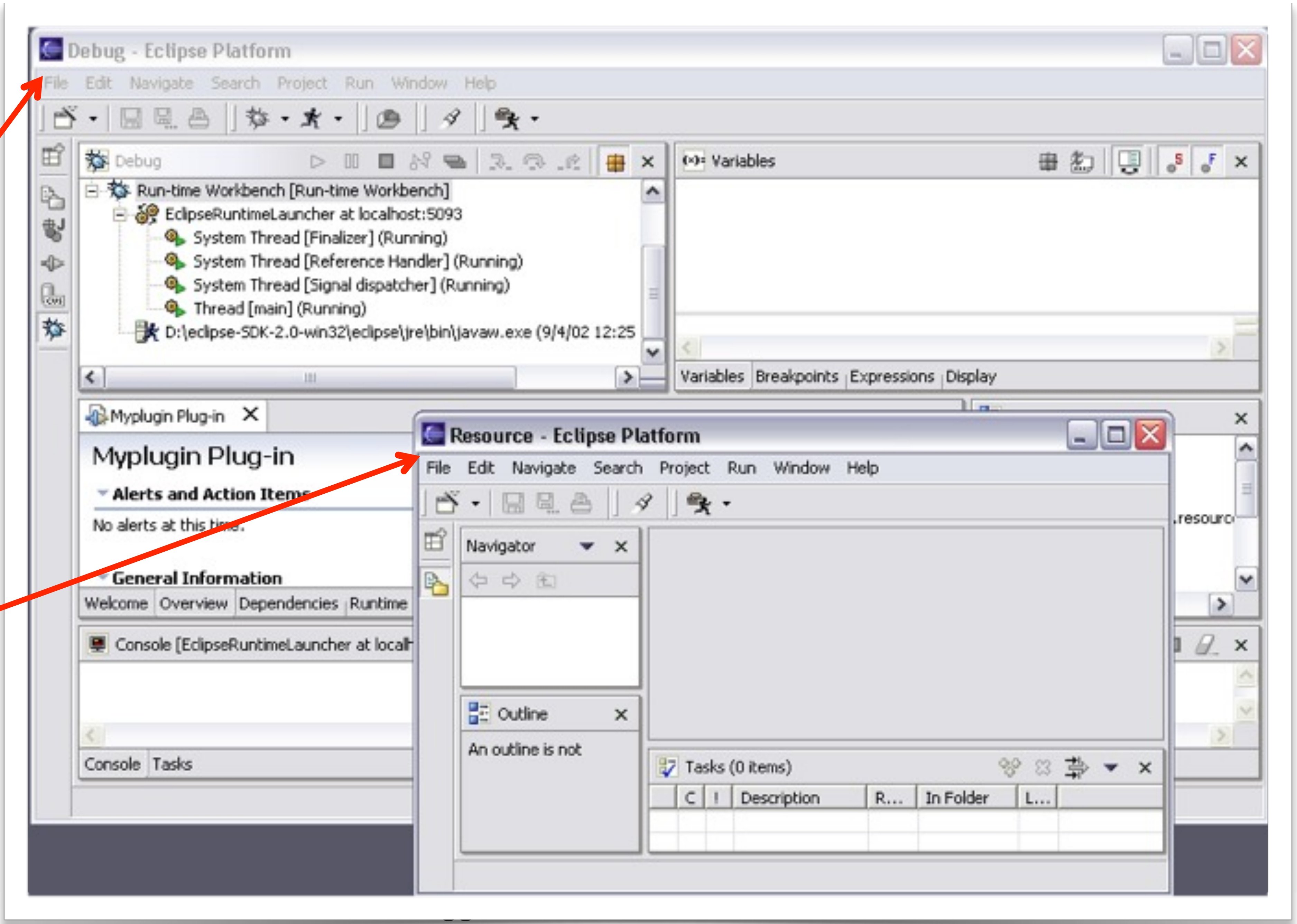
```java
package de.hub.sam.helloworld.popup.actions;

import org.eclipse.jface.action.IAction;

public class NewAction implements IObjectActionDelegate {

    private Shell shell;

    /**
     * Constructor for Action1.
     */
    public NewAction() {
        super();
    }

    /**
     * @see IObjectActionDelegate#setActivePart(IAction, IWorkbenchPart)
     */
    public void setActivePart(IAction action, IWorkbenchPart targetPart) {
        shell = targetPart.getSite().getShell();
    }

    /**
     * @see IActionDelegate#run(IAction)
     */
    public void run(IAction action) {
        MessageDialog.openInformation(
            shell,
            "Helloworld",
            "New Action was executed.");
    }

    /**
     * @see IActionDelegate#selectionChanged(IAction, ISelection)
     */
    public void selectionChanged(IAction action, ISelection selection) {
    }

}
```

PDE

- PDE runs and debugs another Eclipse workbench

1. Workbench running PDE (host)

2. Run-time workbench (target)


Debug - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Debug
Variables
Variables Breakpoints Expressions Display
Myplugin Plug-in
Myplugin Plug-in
Alerts and Action Items
General Information
Welcome Overview Dependencies Runtime
Console Tasks
Resource - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Navigator
Outline
Tasks (0 items)
C ! Description R... In Folder L...

Run-time Workbench [Run-time Workbench]
EclipseRuntimeLauncher at localhost:5093
System Thread [Finalizer] (Running)
System Thread [Reference Handler] (Running)
System Thread [Signal dispatcher] (Running)
Thread [main] (Running)
D:\eclipse-SDK-2.0-win32\eclipse\jre\bin\javaw.exe (9/4/02 12:25

No alerts at this time.

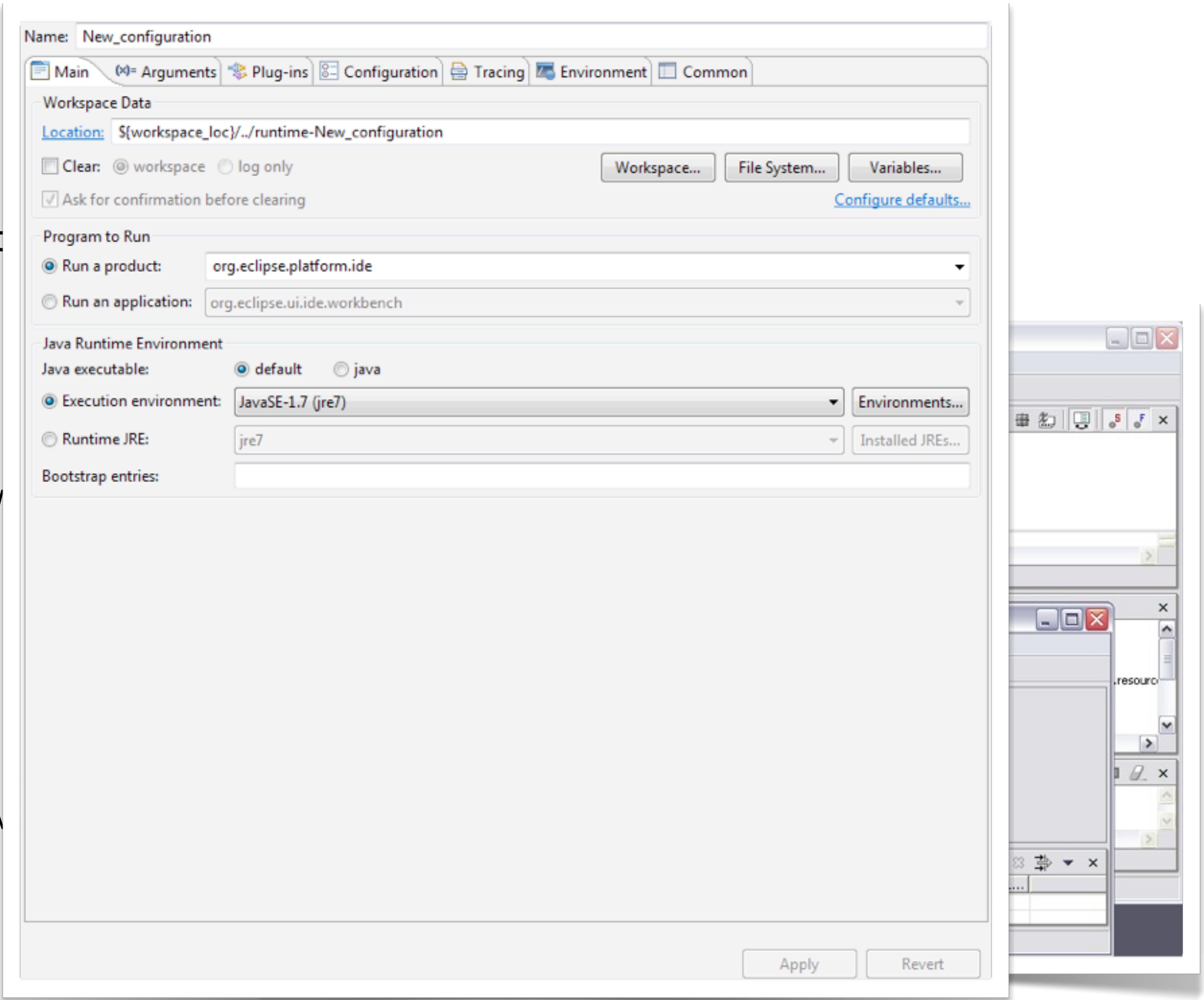
Console [EclipseRuntimeLauncher at localh

An outline is not

▶ F

1. W
ru

2.
w



Name: New_configuration

Workspace Data

Location: ${workspace_loc}/../runtime-New_configuration

[ ] Clear: workspace log only

[✓] Ask for confirmation before clearing

Program to Run

[✓ Run a product:] org.eclipse.platform.ide

Run an application: org.eclipse.ui.ide.workbench

Java Runtime Environment

Java executable: [✓ default] java

[✓ Execution environment:] JavaSE-1.7 (jre7)

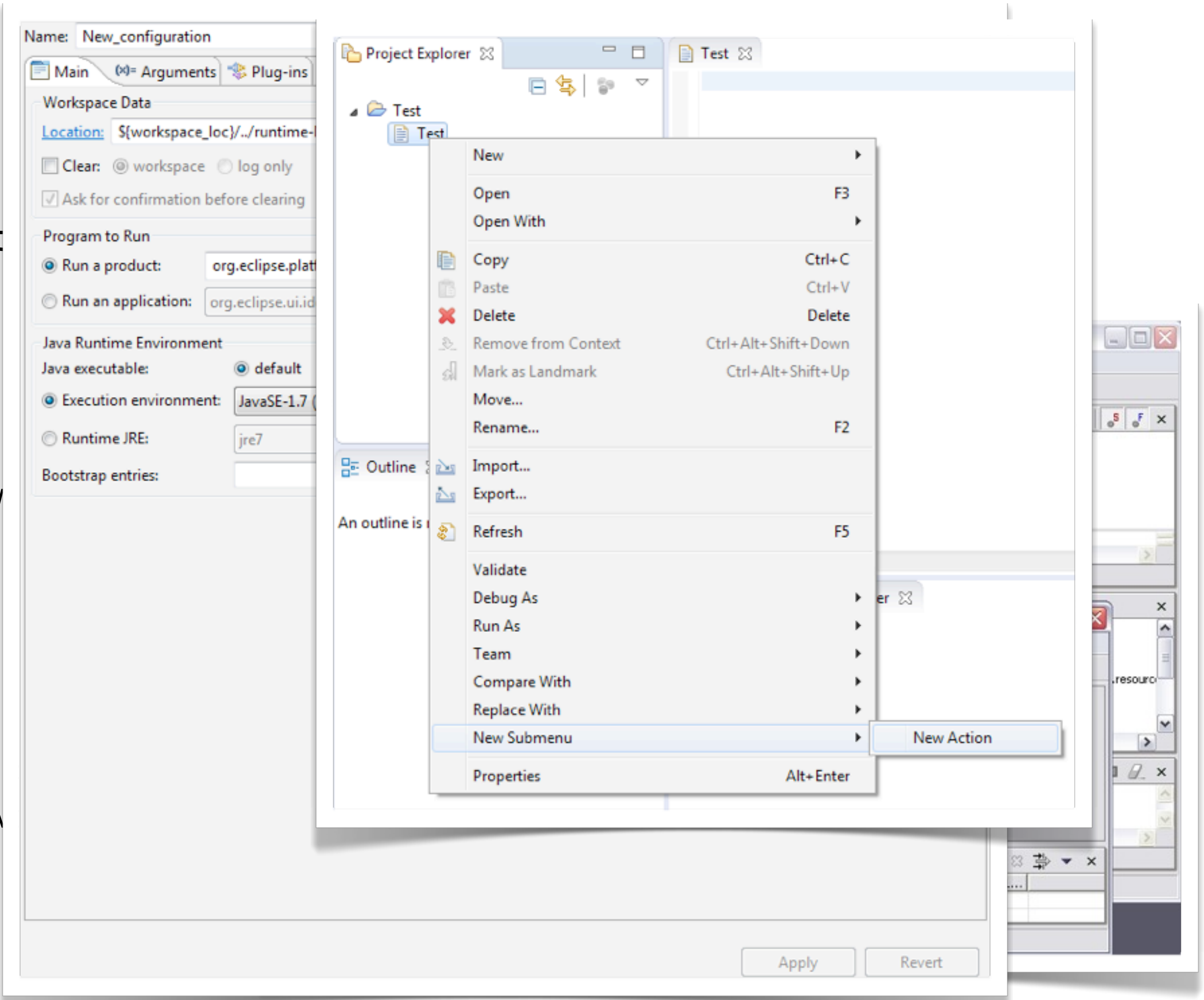
Runtime JRE: jre7

Bootstrap entries:

- F

1. W
   ru

2.
   W


Name: New_configuration

Workspace Data
Location: ${workspace_loc}/../runtime-
Clear: workspace log only
Ask for confirmation before clearing

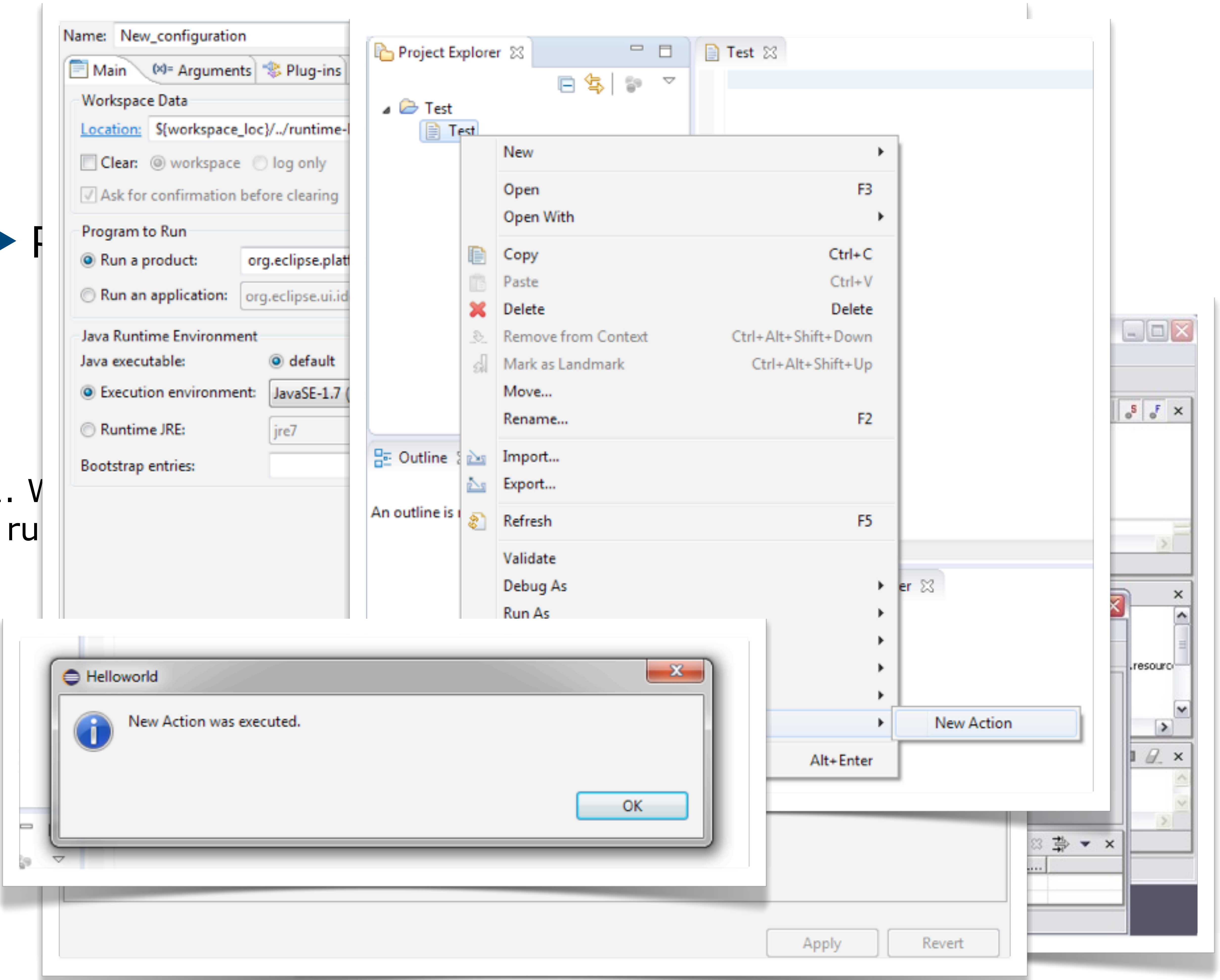
Program to Run
Run a product: org.eclipse.plat
Run an application: org.eclipse.ui.id

Java Runtime Environment
Java executable: default
Execution environment: JavaSE-1.7
Runtime JRE: jre7
Bootstrap entries:

An outline is

▶ F

1. W
ru



Name: New_configuration

Workspace Data
Location: ${workspace_loc}/../runtime-
Clear: workspace log only
Ask for confirmation before clearing

Program to Run
Run a product: org.eclipse.plat
Run an application: org.eclipse.ui.id

Java Runtime Environment
Java executable: default
Execution environment: JavaSE-1.7 (
Runtime JRE: jre7
Bootstrap entries:

Helloworld
New Action was executed.

Summary

- PDE makes it easier to develop Eclipse plug-ins
- PDE is basis for self-hosted
  Eclipse development

Plug-in Distribution Options

Distribution Options

- Plugin
- Feature
- Rich Client Applications (RCP) or simple applications
- update site
- p2 (OSGi provisioning)

Plugin

- build.properties describes how the plugin is exported
- exports into .jar file via eclipse
- can be manually put into eclipse installations
- dependencies, versions, target platforms are not checked, inherently unsafe

Features

- Special PDE project type: feature project
- Describe feature via feature.xml and a special editor
    - plugins
    - depending plugins and features
    - target platform
    - versions
    - license
- Can be bundled into applications
- Can be served via update sites

(Rich Client) Applications

- Special PDE extension point: application
- Special PDE file type and editor: product configuration
  - based on launch configuration (specific application configuration) or application
  - configuration contains
    - plugins and features (and dependencies)
    - target platform (for different OSes)
    - branding, licesing, splash screen

Update Site

- Special PDE project type: update project
- Special PDE file editor: site.xml
  - categories
  - features
- Can be exported and served via web server or p2 repository

p2

- p2 is an extensible provisioning platform for OSGi
- lots of UI for Equinox-based applications
- allows you to create
  - add-on manager for RCP applications
  - installer
  - configuration management system
  - self updating of applications
  - repository

Summary

- lots of distribution options
- most important: plugin vs. application
- software modeling tools and DSLs are usually distributed as plugin and features