Eclipse (3/3)

Deepak Dhungana
dhungana@ase.jku.at
Institute for System Engineering and Automation

Thomas Wuerthinger
wuerthinger@ssw.jku.at
Institute for System Software

Johannes Kepler University Linz, Austria
http://www.jku.at

Rich Client Platform

- The minimal set of plug-ins necessary to build applications
  - No IDE-specific parts
  - No Workspace
- Set of plug-ins is not fixed
  - Usually the workbench is used as the basis
  - Outline and properties views can be used
  - Text editor can be used
    - Limitations because some parts depend on the workspace
  - Form based editors can be used
  - GEF (Graphical Editing Framework) can be used
  - Workspace can be used if application requires a project model
- Startup code must be added
  - Define initial window layout
  - Define global menu and toolbar entries

Branding

- Application != Product
  - Application: Specifies main class that is started
  - Product: Branding and meta data
- Product descriptor
  - .product file with PDE editor
  - Some data is converted to extension in plugin.xml
    - Branding information
  - Some data is used when exporting a product
    - Icon for launcher application
- Product export
  - Export wizard “Eclipse product”
  - Creates distribution with all configured features and plug-ins
  - Cross-platform export possible
    - Launchers for different platforms
    - Requires installation of “RCP delta pack”

Branding

- First impression for user
  - Application icon
  - Splash screen
  - About dialog
- Layout of main window
  - Structure of main window and toolbar
  - Default look-and-feel suitable for many applications
    - Editor for files, views for additional information
  - Perspective often not necessary
    - Define initial perspective, but do provide actions for switching
- Customize layout
  - Everything can be changed
    - Trimming of views and editors
    - Non-rectangle main window
    - Or do not use workbench at all and provide your own main window
  - Really necessary?

Example Application

```
public class DrawingApplication implements IApplication {
  public Object start(IApplicationContext context)
          throws Exception {
    Display display = PlatformUI.createDisplay();

    try {



      int returnCode = PlatformUI.createAndRunWorkbench(display,
            new DrawingWorkbenchAdvisor());

      if (returnCode == PlatformUI.RETURN_RESTART) {
        return IApplication.EXIT_RESTART;
      }
      return IApplication.EXIT_OK;
    } finally {
      display.dispose();
    }
  }
}
```

Class looks very similar for all rich client applications

Initialize SWT

Add code that is run before main window is shown, e.g. login windows

Show workbench window

Check exit conditions

Example Application

```
<extension
    point="org.eclipse.core.runtime.applications"
    id="DrawingApplication">
  <application>
    <run class="at.ssw.drawit.internal.rcp.DrawingApplication“/>
  </application>
</extension>
```

Application extension point

ID without plug-in prefix

Main class

```
<extension
    point="org.eclipse.core.runtime.products"
    id="DrawingProduct">
  <product
      application="at.ssw.drawit.rcp.DrawingApplication"
      name="DrawIt">
    <property name="aboutText" value=“... "/>
    <property name="aboutImage“ value="branding/about.gif"/>
  </product>
</extension>
```

Product extension point

Branding information

Example Application

```
public class DrawingWorkbenchAdvisor extends WorkbenchAdvisor {

  public WorkbenchWindowAdvisor createWorkbenchWindowAdvisor(
          IWorkbenchWindowConfigurer configurer) {
    return new DrawingWorkbenchWindowAdvisor(configurer);
  }

  public String getInitialWindowPerspectiveId() {
    return UIUtilities.PERSPECTIVE_ID;
  }

  public void initialize(IWorkbenchConfigurer configurer) {
    configurer.setSaveAndRestore(true);
  }
}
```

Initial perspective for new workbench windows

Remember window layout when application is closed

Example Application

```
public class DrawingWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {
  public ActionBarAdvisor createActionBarAdvisor(
          IActionBarConfigurer configurer) {
    return new DrawingActionBarAdvisor(configurer);
  }

  public void preWindowOpen() {
    String prop = IWorkbenchPreferenceConstants.SHOW_TRADITIONAL_STYLE_TABS;
    PlatformUI.getPreferenceStore().setValue(prop, false);

    IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
    configurer.setInitialSize(new Point(900, 700));
    configurer.setShowFastViewBars(true);
    configurer.setShowCoolBar(true);
    configurer.setShowStatusLine(true);
    configurer.setShowMenuBar(true);
    configurer.setShowPerspectiveBar(false);
    configurer.setShowProgressIndicator(false);
  }
}
```

Property can also be set in configuration file

Visibility of design elements (Most cannot be changed by user)

Example Application

```java
public class DrawingActionBarAdvisor extends ActionBarAdvisor {
  protected void fillCoolBar(ICoolBarManager coolBar) {
    IToolBarManager fileToolBar = new ToolBarManager(coolBar.getStyle());
    fileToolBar.add(newAction);
    fileToolBar.add(new Separator(IWorkbenchActionConstants.MB_ADDITIONS));
    ...
  }

  protected void fillMenuBar(IMenuManager menuBar) {
    MenuManager fileMenu = new MenuManager("&File",
        IWorkbenchActionConstants.M_FILE);
    fileMenu.add(newAction);
    fileMenu.add(closeAction);
    menuBar.add(fileMenu);
  }

  protected void makeActions(final IWorkbenchWindow window) {
    newAction = new NewAction(window);
    register(newAction);
    closeAction = ActionFactory.CLOSE.create(window);
    register(closeAction);
  }
```

Define markers for declarative additions

My own action

Action provided by Eclipse

Example Application

```
<extension
    point="org.eclipse.core.runtime.adapters">
  <factory
      adaptableType="org.eclipse.ui.IEditorInput“



      class="at.ssw.drawit.internal.rcp.editor.RCPDrawingAdapterFactory">

    <adapter
        type="at.ssw.drawit.ui.editor.DrawingEditorAdapter“
    />
  </factory>
</extension>
```

Extension point for Adapter

Class that is adapted

Factory that creates adapter objects

Adapter class

Live Demo--

- Baghchal-- Rich Client Platform
- Tetris-- Defining your own extension points
- Features and Update Sites