Eclipse Rich Client Applications

Overview of the Generic Workbench
(Eclipse RCP UI)

Presented by Nick Edgar, IBM Ottawa
Just what does “RCP” mean?!?

- The Eclipse OSGi runtime (dynamic plug-ins)?
- The Eclipse Generic Workbench?
- The foundation for Lotus Workplace?
- “Really Cool Platform”?
- A Platform we want to sell to really Rich Clients?

- Answer: All the above?
Just what does “RCP” mean?!?

- The Eclipse OSGi runtime (dynamic plug-ins)?
- The Eclipse Generic Workbench?
- The foundation for Lotus Workplace?
- “Really Cool Platform”?
- A Platform we want to sell to really Rich Clients?

Answer: All the above
(except the last one -- Eclipse is still open source).

- “Rich Client Platform”
  A Platform for building Client applications with Rich functionality
What is the Eclipse RCP?

That was then:

“Eclipse is a kind of universal tool platform - an open extensible IDE for anything and nothing in particular.”

This is now:

Eclipse is an open, extensible platform for any application.
Why are we doing this?

Just look around at EclipseCon for non-IDE uses of Eclipse:

- “Building Large-Scale Enterprise Applications with Eclipse”
  (Michael Bechauf, SAP AG)

- “UrbanSim” – urban growth simulation platform
  (Bjorn Freeman-Benson, U. of Washington)

  (Mike Taylor, Instantiations)

- “Eclipse-based Apps – Java on the Desktop Revisited
  (Todd Williams, Genuitec)

- “Lotus Workplace: Rich Client Platform”
  (Richard Wilson, IBM Lotus)

- “Experiences with Rich Client Application Development”
  (Frank Gerhardt, SENS; Chris Wege, DaimlerChrysler)
Logiball GDF Suite, an industrial-strength Geographic Data Management System (courtesy of Frank Gerhardt)
Life Sciences / BioTech Workbench (mockup) (courtesy of Elias Torres, WebAhead group, IBM)
Lotus Workplace Personal Productivity (mockup)
(courtesy of Matthew Hatem, Lotus RCP group)
Goals for Eclipse RCP UI (Generic Workbench)

- Without losing focus on Eclipse as a tools platform, open it up to allow it to be used for arbitrary applications.
- Remove IDE personality from the Workbench proper.
- Remove assumptions that the Workspace is the data model.
- Make the Workbench configurable (and get out of the driver’s seat).
- Make most other components optional (rich function, low footprint).
- Minimize API breakage.
Eclipse Platform 2.1 (pre-RCP)

- Help
- Update
- Compare
- Debug
- Search
- Team/CVS

UI

Text
IDE Views

Workbench (with IDE personality)

JFace

SWT

Resources

Runtime

Primary Application
Eclipse Platform 3.0 (post-RCP)

- Help (optional)
- Update (optional)
- Text (optional)
- IDE
- Text
- Compare
- Debug
- Search
- Team/CVS

Primary Application

Eclipse RCP

UI (Generic Workbench)

Resources (optional)

- JFace
- SWT
- Runtime (OSGi)
API Changes: Extension Points

A few IDE- or Text-specific extension points have moved. Workbench retains most extension points for rich function.

Workbench (unchanged):
- Action Sets
- Commands
- Contexts
- Decorators
- Editors
- New/Import/Export Wizards
- Perspectives
- Popup Menus
- Preference Pages
- Property Pages
- Views
- Working Sets

Moved to IDE layer:
- Marker Help and Resolutions
- Marker and Project Nature Images
- Resource Filters

Moved to Text component:
- Document Providers
- Marker Updaters

New to Workbench in 3.0:
- Activities
- Contexts
- Themes
API Changes: Java API

A handful of breaking API changes were needed to remove the bias on the Workspace as the underlying data model. Equivalents for most of these are provided at the IDE level.

- **IWorkbenchPage**
  - openEditor(IFile, …)
  - openEditor(IMarker, …)
  - openSystemEditor(IFile)

- **IEditorPart**
  - gotoMarker(IMarker)

- **IEditorLauncher**
  - open(IFile)

- **IEditorRegistry**
  - get/setDefaultEditor(IFile, …)
  - getImageDescriptor(IFile)

- **IWorkbench**
  - getMarkerHelpRegistry()

- **IDE**
  - openEditor(IWorkbenchPage, IFile, …)
  - openEditor(IWorkbenchPage, IMarker, …)
  - openSystemEditor(IWorkbenchPage, IFile)

- **IDE.gotoMarker(IEditorPart, IMarker, …) // caller**
  - IGotoMarker.gotoMarker(IMarker) // implementor

- **IEditorLauncher**
  - open(IPath)

- **IDE**
  - get/setDefaultEditor(IFile, …)
  - getImageDescriptor(IFile)

- **IDE**
  - getMarkerHelpRegistry
Compatibility Story

- Porting guide
  - Describes all breaking API changes and how to adapt to them.

- For manifest files:
  - plugin.xml files are tagged with version (no tag means <= 2.1).
  - Runtime automatically converts pre-3.0 plug-ins to 3.0 structure:
    - Converts plug-in dependencies and extension point ids
    - PDE wizards for development-time conversion

- For API changes:
  - Binary compatibility fragment (runtime only)
Binary compatibility for removed methods ?!? 

2.1 (before):

interface IWorkbenchPage {
  IViewPart showView(String viewId);
  IEditorPart openEditor(IFile file); // to remove
  ...
}

3.0 (after):

interface IWorkbenchPage {
  // empty
}

Development time:

interface ICompatibleWorkbenchPage {
  // empty
}

Runtime:

(in org.eclipse.ui.workbench.compatibility fragment)

interface ICompatibleWorkbenchPage {
  IEditorPart openEditor(IFile file); // to remove
}
Configuring the Workbench

- New APIs to allow the application to configure the Workbench for its needs, in package org.eclipse.ui.application.
- These privileged APIs are not available to regular plug-ins.

<table>
<thead>
<tr>
<th>Workbench implements</th>
<th>Application implements</th>
</tr>
</thead>
<tbody>
<tr>
<td>IWorkbenchConfigurer</td>
<td>WorkbenchAdvisor</td>
</tr>
<tr>
<td>- setSaveAndRestore(boolean)</td>
<td>- initialize(workbenchConfigurer)</td>
</tr>
<tr>
<td>- getWindowConfigurer(window)</td>
<td>- getInitialWindowPerspectiveId()</td>
</tr>
<tr>
<td>- ...</td>
<td>- fillActionBars(window, actionBarConfigurer, flags)</td>
</tr>
<tr>
<td>IWorkbenchWindowConfigurer</td>
<td>- pre/postStartup()</td>
</tr>
<tr>
<td>- setTitle(String)</td>
<td>- pre/postShutdown()</td>
</tr>
<tr>
<td>- setShowMenuBar(boolean)</td>
<td>- pre/postWindowOpen(windowConfigurer)</td>
</tr>
<tr>
<td>- setShowCoolBar(boolean)</td>
<td>- pre/postWindowClose(windowConfigurer)</td>
</tr>
<tr>
<td>- setShowStatusLine(boolean)</td>
<td>- eventLoopIdle(Display)</td>
</tr>
<tr>
<td>- setShowShortcutBar(boolean)</td>
<td>- ...</td>
</tr>
<tr>
<td>- ...</td>
<td></td>
</tr>
</tbody>
</table>
Configuring the Workbench: Control flow

Diagram courtesy of Ed Burnette.
Relationship to OSGi Runtime (Equinox)

- Workbench runs on top of the Runtime (obviously)

- Workbench is now “dynamic aware”:
  - Reacts to plug-ins getting installed and uninstalled dynamically.
  - Adds/removes perspectives, views, action sets, etc., updating UI accordingly.
  - Provides event listeners for downstream plug-ins to react to Workbench changes in turn.
Footprint

- What is the minimal footprint?
  - 4.8M (as of 3.0 M6):
    - Runtime/OSGi: 0.9M
    - SWT: 1.4M
    - JFace: 0.5M
    - Workbench: 2.0M
Where can I find out more?

RCP UI Proposal page:
eclipse.org/platform > UI > Proposals > Rich Client Platform

More at EclipseCon:
- “Implementing Rich Client Applications” Technology Exchange (tomorrow at 17:00)
- RCP BOF, other talks

- Ed Burnette’s RCP tutorials (on proposal page)
- Browser example (on proposal page)
- Platform newsgroup:
  news://news.eclipse.org/eclipse.platform