From RCP to RCP/RAP Conversion and Single-Sourcing Techniques

Elias Volanakis
evolanakis@innoopract.com

Jordi Böhme López
jboehme@innoopract.com

-- last update: April 2008 --
Table of Contents

- Introduction
  - Enabling Plug-in reuse
  - Where are the Pitfalls?
  - How to do it?

- Converting an Application
  - Manage dependencies
  - Make it compile
  - Make it launch

- Single-Sourcing techniques
  - Move RCP-only functionality out of the way
  - One for all – hide the differences
  - Change Singletons to SessionSingletons

- Q & A
From **RCP** to **RAP**

**RCP - Rich Client Platform**

RCP enables

**component oriented development and assembly of client applications**

using **SWT, Jface and Workbench** technology
From RCP to RAP

RAP - Rich Client Ajax Platform

RAP enables component oriented development and assembly of client web applications

using SWT, Jface and Workbench technology
What is RAP?

“RCP for the Web”

- web-enabled port of Eclipse UI
- OSGi-based (plug-ins, dependency management)
- one Eclipse instance serving multiple users (i.e. UIs)

- same namespace & APIs → code reuse
Real World Example: Yoxos Install Manager
Real World Example: OnDemand Service
Enabling plug-in reuse

high amount of reuse possible

but there are some Pitfalls...
Enabling plug-in reuse

Pitfall #1

RAP is a subset of RCP → separate incompatible code
Enabling plug-in reuse

Pitfall #1

RAP is a subset of RCP $\rightarrow$ separate incompatible code

Examples of incompatible code:

- Image.dispose()
- 2D drawing (GC; Draw2D / GEF)
- Key Events
- Rich Text Editing (e.g. Java Editor)
- Native Dialogs (File & Directory Dialogs)
- Multiple Workbench Windows
- Cell Editors
Enabling plug-in reuse

Pitfall #2

“as-is” port does not make sense $\rightarrow$ divide functionality
Enabling plug-in reuse

Pitfall #2

“as-is” port does not make sense → divide functionality

Examples:

- Update / Install of the application
- File Access
- Printing
Enabling plug-in reuse

Pitfall #3

In RAP **one app** instance is used by **many users** → make app multi-user enabled
Enabling plug-in reuse

Pitfall #3

In RAP one app instance is used by many users → make app multi-user enabled

Troublemakers:

- Singletons → Session Singletons
- Memory Hogs (huge data model x n-users ?)
Plug-in Reuse – How to do it?

- It's work, not “magic”
- Start small
- Identify “Troublemakers”
  - Unsupported APIs and RCP/RAP-only code; Singletons; Memory Hogs
- Change dependencies
- Does it compile?
  - Refactor, Modularize, Refactor
- Does it make sense?
  - Refactor, Modularize, Refactor
- Does it work?
  - Do you have tests? Plus, manual testing; profiling; stress testing
Table of Contents

- Introduction
  - Enabling Plug-in reuse
  - Where are the Pitfalls?
  - How to do it?

- Converting an Application
  - Manage dependencies
  - Make it compile
  - Make it launch

- Single-Sourcing techniques
  - Move RCP-only functionality out of the way
  - One for all – hide the differences
  - Change Singletons to SessionSingletons

- Q & A
Step 0 – Getting the Content

1. Pass around the memory stick

2. Copy rcp_to_rap-<your_os>.zip to your HD
Step 1 – Running the RCP App

1. Unzip rcp_to_rap-<your_os>.zip
2. Start the contained Eclipse
   c:\rcp_to_rap\eclipse\eclipse.exe
3. Use the workspace from the .zip
   c:\rcp_to_rap\workspace
4. Import the projects
   File > Import > Existing projects into workspace > Select root directory
5. Run “MP3 Manager - RCP”
Step 1 – Running the RCP App

You should see something like this...
Step 2 – Using the RAP Target

0. Stop the RCP App
1. Window > Preferences > Plug-in Development > Target Platform

2. Change Location to use the RAP Target
c:\rcp_to_rap\target\eclipse

3. Reload
   Should have 36 plug-ins

4. Ok
What is a Target Platform?

- **Eclipse IDE**
  - Executes
  - Works on
  - IDE plug-ins

- **workspace plug-ins**
  - launches

- **RCP/RAP App**
  - Executes
  - target plug-ins
From RCP to RCP/RAP

RCP Target

- RCP only

RCP + RAP

RAP Target

- RAP only
From RCP to RCP/RAP

RCP Target

Plugin c.i.mp3m.ui.rcp

RCP only

Plugin c.i.mp3m.ui.rap

RAP Target

RCP + RAP

c.i.mp3m.ui

RAP only
Managing Dependencies

Common UI Code

- must compile against RCP and RAP impl. of Eclipse UI

- introduce dependency to both implementations

1. Require Bundle
2. Import Package
Step 3 – Managing Dependencies

1. Open
   com.siemens.ct.mp3m.ui/META-INF/MANIFEST.MF

2. Add dependencies to
   + org.eclipse.rap.ui
   + org.eclipse.rap.ui.forms

3. Make ui dependencies
   optional
3, 2, 1, Launch...

How to launch the RAP Application?

1. Need RAP UI plug-in
2. Need an EntryPoint

Done by 'com.innoopract.mp3m-ui.rap' for your convenience.
Step 4 – Launching the RAP Application

1. Run “MP3 Manager – RAP”

Note: normally you would now have several compile errors in the code. We have commented out problematic code. We'll address these issues as we go.
Table of Contents

- Introduction
  - Enabling Plug-in reuse
  - Where are the Pitfalls?
  - How to do it?

- Converting an Application
  - Manage dependencies
  - Make it compile
  - Make it launch

- Single-Sourcing techniques
  - Move RCP-only functionality out of the way
  - One for all – hide the differences
  - Change Singletons to SessionSingletons

- Q & A
Moving RCP-only stuff out of the way

Examples:
- org.eclipse.ui.bindings
- org.eclipse.ui.splashHandlers
- things that don't make sense in a multi-user app

1. Moved contributions (plugin.xml)
2. Moved Java classes

```
<extension
   point="org.eclipse.ui.splashHandlers">
  <splashHandler
    class="com.siemens.ct.mp3m.handlers.SplashHandler"
    id="com.siemens.ct.mp3m.splashHandler">
  </splashHandler>
</extension>
```
Replacing with similar functionality

Example #1
- KeyListener with ModifyListener

```java
public class AlbumRenamePage extends WizardPage
{
    /* implements KeyListener */

    implements ModifyListener
    {

        public void keyPressed(KeyEvent e) {
        }

        public void keyReleased(KeyEvent e) {
            updateModel();
        }

        public void modifyText(ModifyEvent e) {
            updateModel();
        }
    }
```
Replacing with similar functionality

Example #2:
- DirectoryDialog with InputDialog

```java
// DirectoryDialog d = new DirectoryDialog(window.getShell());
// String folder = d.open();

InputDialog d = new InputDialog(w, title, message, defaultValue,
                               new DirectoryInputValidator() );
int result = d.open();
```
Hiding API Differences

Examples:
- `Image.dispose()`
- `ActionFactory.OPEN_NEW_WINDOW.create(...)`

Idea:
- Use a common Interface
- Put differences in the implementation

How to obtain / instantiate the implementation?
Hiding API Differences

Extension point finds and instantiates a RAP or RCP implementation of an interface or class.

Note: this has nothing to do with OSGi services.
Possible alternative: OSGi Declarative Services
Hiding API Differences – How To

1. Create your interface
2. Implement your interface
3. Implement and register an IFacadeFactory

```xml
<extension
    point="com.innoopract.core.facadefactory.facadeFactories">
  <factory
      factory="com.innoopract.mp3m.ui.rcp.internal.ActionFacadeFactory"
      type="com.innoopract.mp3m.facades.IActionFacade"/>
</factory>
```

4. Obtain implementation

```java
Object object = FacadeActivator.getFacade( IActionFacade.class );
if( object instanceof IActionFacade ) {
  IActionFacade actionFacade = ( IActionFacade ) object;
  newWindowAction = actionFacade.createOpenNewWindowAction( window );
```
Step 5 – Hiding API differences

1. Uncomment lines 12, 19 in the `plugin.xml` of `com.innoopract.mp3m.ui.rap`
2. Have a look at lines 38-41 in the `plugin.xml` of `com.innoopract.mp3m.ui.rcp`
3. Open the type hierarchy for `IMusicFolderStore` and compare the two implementations
4. Open the type hierarchy for `IFacadeFactory`. Compare the two implementations of `MusicFolderStoreFactory`
5. Run your RAP App. You should now see a few entries in the views
Challenge: Singletons

RCP: one software instance per user
RAP: one software instance for all users

The Singleton pattern can be problematic.

Do you want one Singleton per software instance or one per user?

SessionSingletonBase can help
Singletons vs SessionSingletons
public class FooSingleton extends SessionSingletonBase {

    private FooSingleton() {}

    /* add more methods here */

    public static FooSingleton getInstance() {
        return (FooSingleton) getInstance(FooSingleton.class);
    }
}
Challenge: Lifecycle Divergence
Lifecycle Divergence Example

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

    Object object = ServiceActivator.getService(IMusicFolderStore.class);
    if (object instanceof IMusicFolderStore) {
        String musicFolders = ((IMusicFolderStore) object).loadFolders();
        MusicFolders.setMusicFolders(musicFolders);
    }
}

public void preStartup() {
    // DONE RAP - moved from Mp3mPlugin.start()
    Object object = ServiceActivator.getService(IMusicFolderStore.class);
    if (object instanceof IMusicFolderStore) {
        String musicFolders = ((IMusicFolderStore) object).loadFolders();
        MusicFolders.setMusicFolders(musicFolders);
    }
}```
Guarding against code breakage

- Two targets (RCP, RAP)
  - Put them on version control
  - Share them

- Two workspaces (RCP, RAP)

- Continuous build for both versions
  - PDE-Build + Hudson / CruiseControl

- Automated tests
  - Eclipse-Testing-Framework
  - JUnit Tests for RCP + RAP
  - PDE JUnit tests for RCP (work in progress for RAP)
Summary - From RCP to RAP

- It's work, not “magic”
- Start small
- Identify “Troublemakers”
- Change dependencies
- Does it compile?
- Does it make sense?
- Does it work?
Links & Resources

- These Slides
- RAP Homepage
- RAP Insights (blog)
- MP3 Manager for RCP

Image: © photocase.de
Thanks for attending!

- **Contact Info:**

  Elias Volanakis  
  evolanakis@innooprac.t.com

  Jordi Böhme López  
  jboehme@innooprac.t.com