# Developing Eclipse Rich-Client Applications Tutorial

**Dr. Frank Gerhardt**
Gerhardt Informatics Kft.
fg@gerhardtinformatics.com

**Michael Scharf**
Wind River
eclipsecon@scharf.gr

# Versions

Version 1.6, March 2007, EclipseCon 2007 (Eclipse 3.3M5eh)

Version 1.5, March 2006, EclipseCon 2006 (Eclipse 3.2M5a)

Version 1.4, September 2005, /ch/open Workshoptage, Zurich (Eclipse 3.1)

Version 1.3, June 2005, iX-Konferenz, Heidelberg (Eclipse 3.1)

Version 1.2, February 2005: EclipseCon 2005 (Eclipse 3.1M5a)
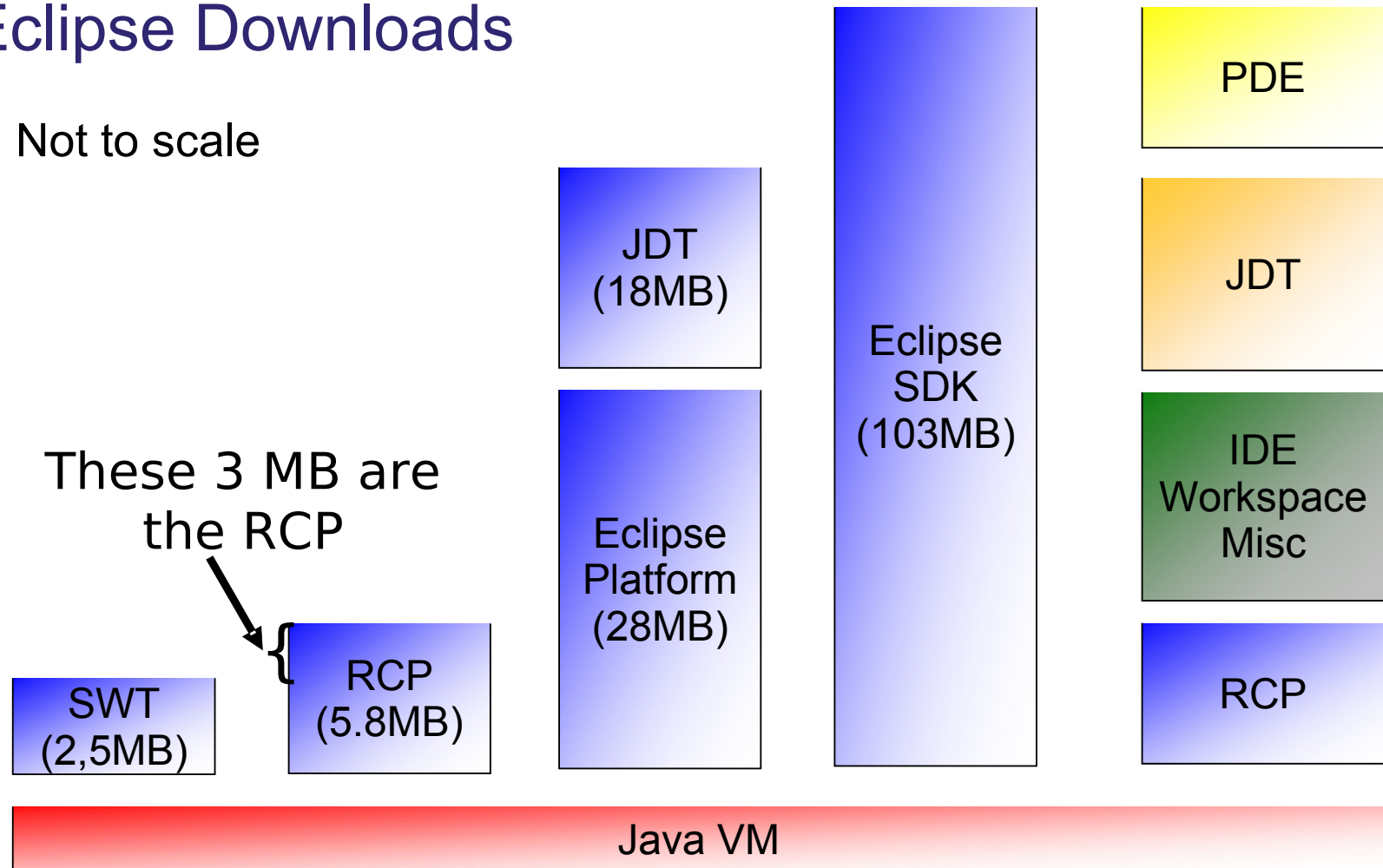
Version 1.1, January 2005, OOP, Munich (Eclipse 3.1M4)

Version 1.0, June 2004: iX-Konferenz, Heidelberg (Eclipse 3.0RC2)
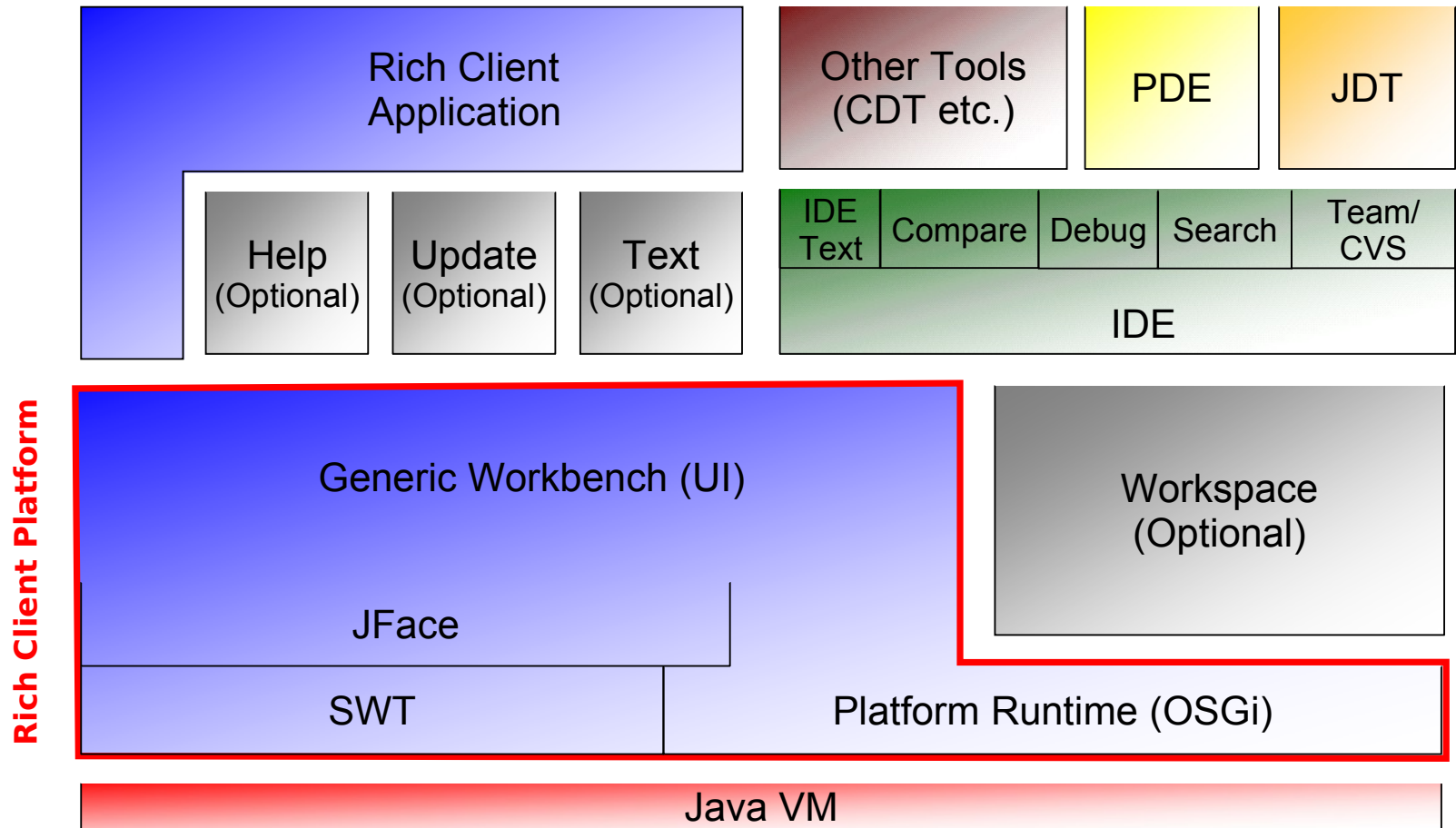
# About the Tutorial

- The tutorial is full of hands-on excercises
    - Not a lecture
- These slides provide an architectural overview at the beginning of the tutorial

- Requirements
    - Notebook
    - Java 1.4 or higher installed
    - Eclipse SDK 3.3M5eh installed (slight differences from 3.2.x!)

- We provide the required software on CD-ROM and USB stick
    - Slides
    - Eclipse distributions (Linux GTK, Mac, Win32)

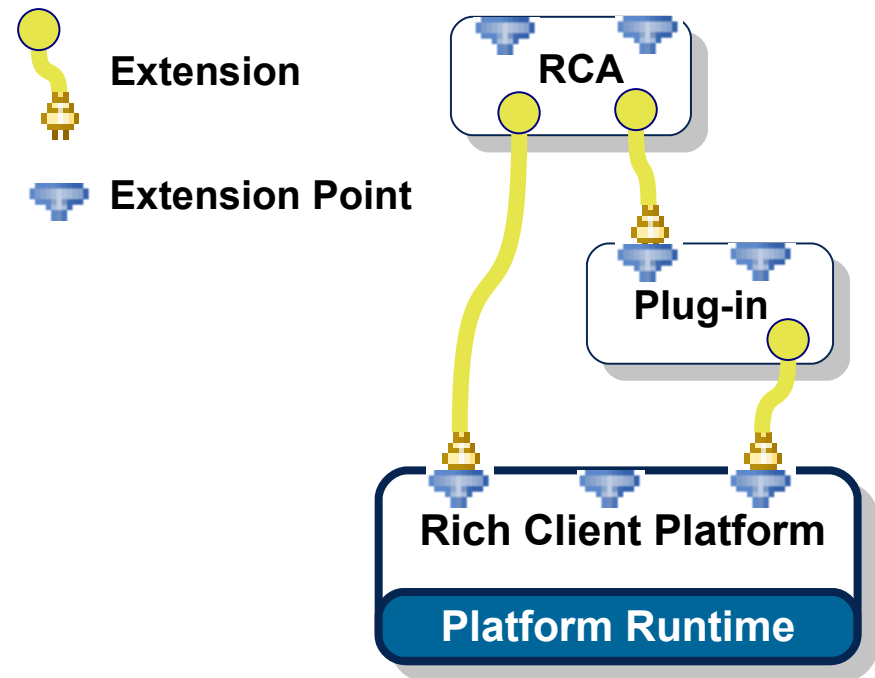# Eclipse Downloads

- Not to scale

JDT
(18MB)

Eclipse
SDK
(103MB)

PDE

JDT

These 3 MB are
the RCP

Eclipse
Platform
(28MB)

IDE
Workspace
Misc

RCP
(5.8MB)

RCP

SWT
(2,5MB)

Java VM

# Eclipse Architecture

Developing Eclipse Rich-Client Applications | © 2007 by Frank Gerhardt and Michael Scharf
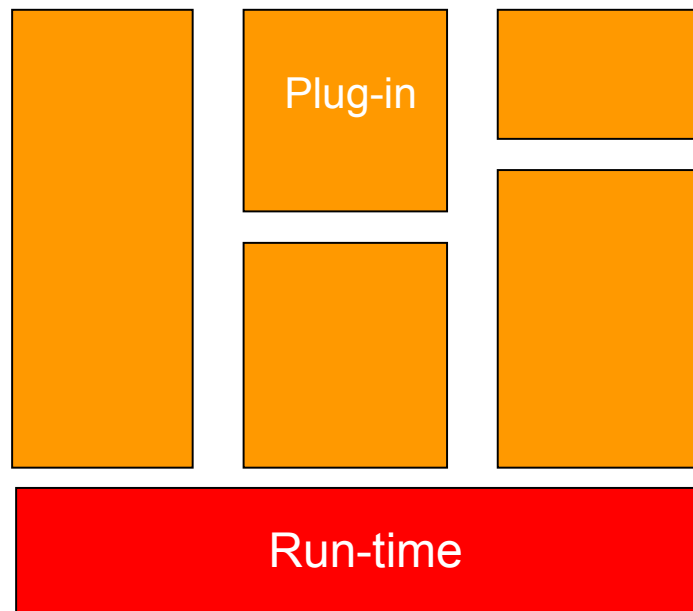
# Platform Runtime (org.eclipse.core.runtime)

- Plug-ins
  - The Eclipse synonym to an OSGi bundle
  - Declares the component model
    - Extensions
    - Extension points

- Platform runtime
  - Contains OSGi and the plug-in runtime
  - Is the runtime environment for bundles / plug-ins
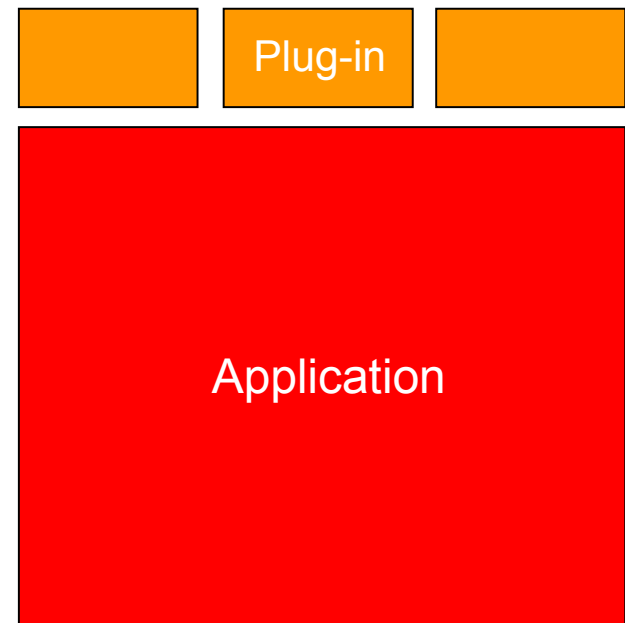  - Responsible for Jobs and preferences

**Extension**

**Extension Point**

**RCA**

**Plug-in**

**Rich Client Platform**

**Platform Runtime**

Developing Eclipse Rich-Client Applications  |  © 2007 by Frank Gerhardt and Michael Scharf

# Platform vs. extensible application

Eclipse is a platform. It has a very small kernel



Platform

Extensible Application

Developing Eclipse Rich-Client Applications | © 2007 by Frank Gerhardt and Michael Scharf

# Eclipse is an Iceberg

- Separation of declaration and implementation
  - Startup time: O(#used plug-ins), not O(# installed plug-ins)

Developing Eclipse Rich-Client Applications | © 2007 by Frank Gerhardt and Michael Scharf
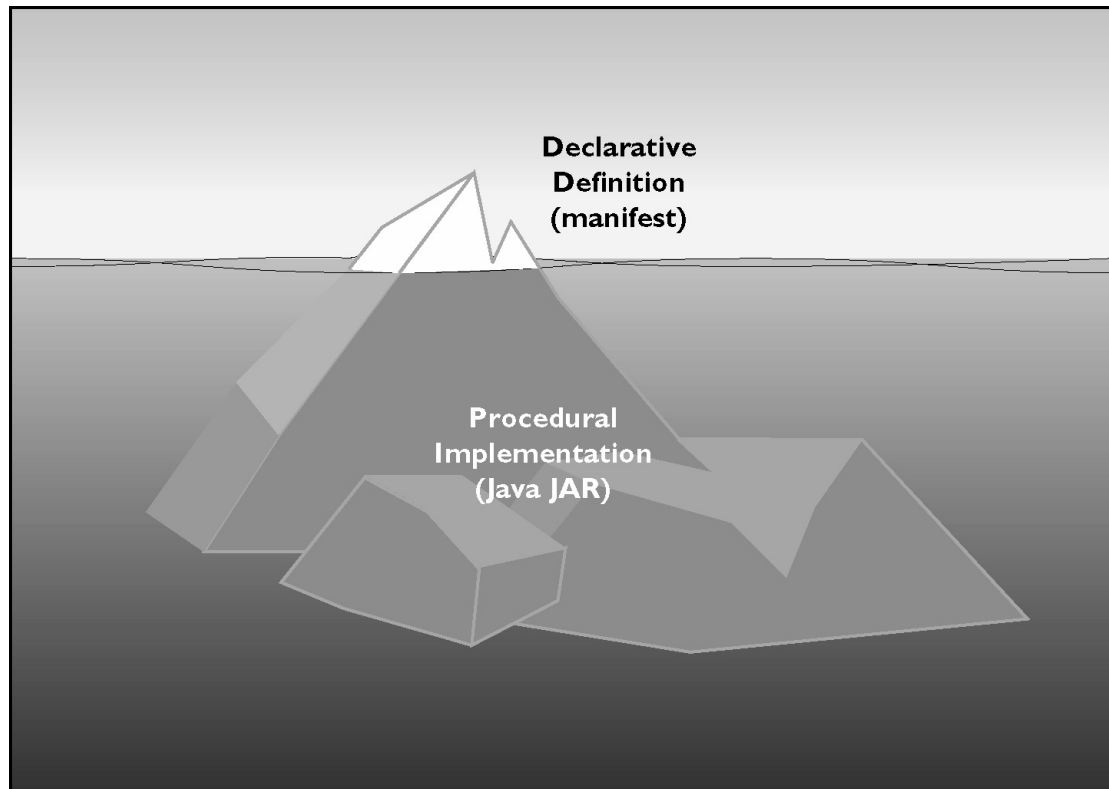
# Plug-in Namespaces

- Each plug-in has its own classloader
- Requests are delegated to the responsible classloader
- Therefore: Management of plug-in dependencies are essential
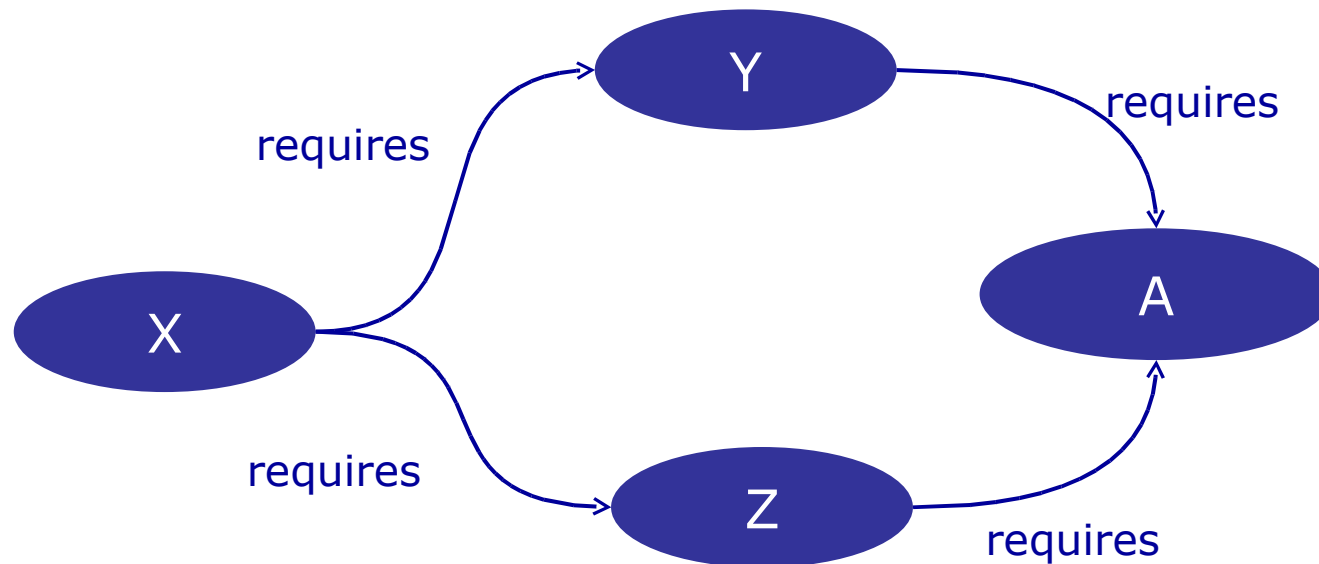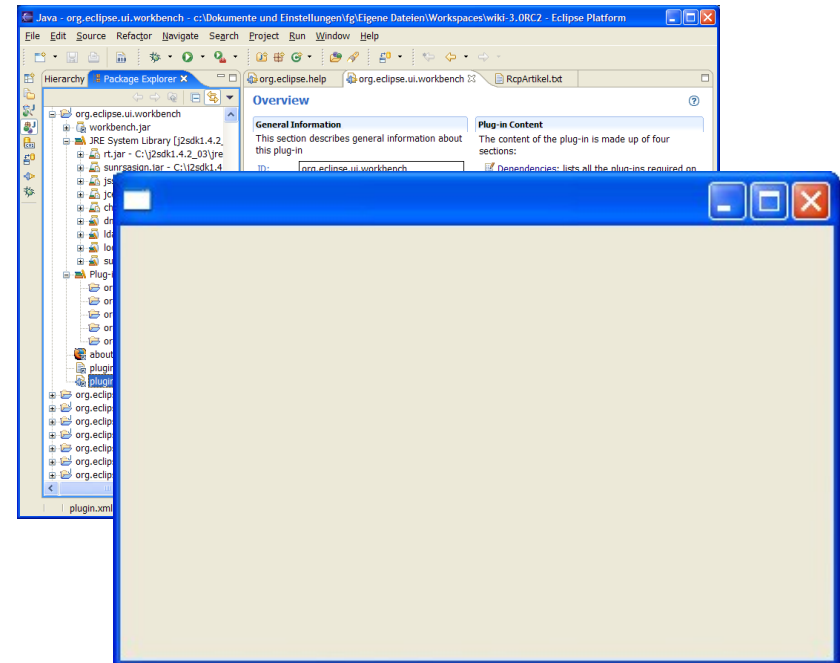
# SWT & JFace

- SWT
  - Native GUI-Widgets
  - Not OO (e.g. API-compliance between different OS only due to conventions)

- JFace
  - Abstraction over SWT
    - Viewer
    - Forms-API
    - Wizards / Dialogs / Actions
    - MVC / Command Pattern

# Workbench

- Workbench
    - Contributes the empty window
    - Adds support for
        - Menubars
        - Toolbars
        - Perspectives
        - Views / editors
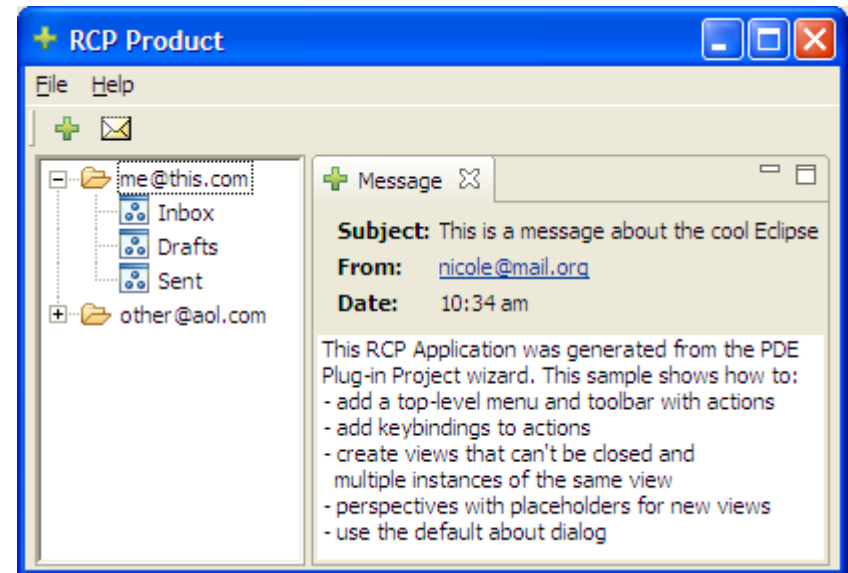        - Preferences
        - And many more extensionpoints

# Key Concepts

- Plug-in / Bundle
    - The basic component in Eclipse with **dependencies** to other plug-ins / bundles.
- Extension Points and Extentions
    - The **declarative wiring** between the Eclipse framework and custom code
- Feature
    - A **group** of plug-ins with a version number
- Product
    - A **deployable application** with branding, defined in terms of plug-ins or features
- Update-site
    - A **directory**, usually on a web server, for the initial download and update of features

Developing Eclipse Rich-Client Applications | © 2007 by Frank Gerhardt and Michael Scharf

# Tutorial Overview

- Part I: From Hello, World! to Workbench window

- Part II: From RCP Mail to a product with
  - Update Manger
  - Help System
  - Extension Point
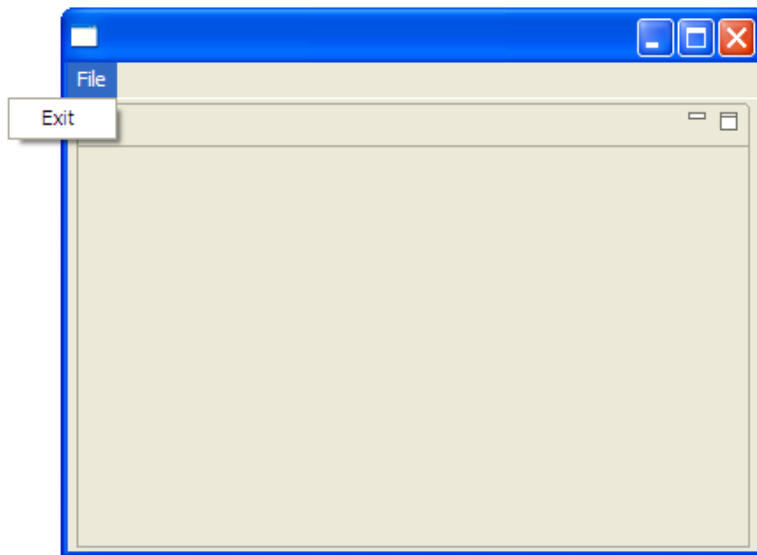
Developing Eclipse Rich-Client Applications  |  © 2007 by Frank Gerhardt and Michael Scharf

# Legend

Feature

Jar

Extension

Plug-in manifest

Package

Extensions

Plug-in

Bundle

Extension Point

Plug-in Dependencies

Class

Interface

| a class |
| --- |
| main() |

| an interface |
| --- |
| run() |

**Feature X**

**Plugin A**

**Plug-in B**

**Product**

**Feature A**

**Feature B**

Developing Eclipse Rich-Client Applications  |  © 2007 by Frank Gerhardt and Michael Scharf

HelloEclipse

main()

Developing Eclipse Rich-Client Applications | © 2007 by Frank Gerhardt and Michael Scharf

**rcpsimple**

HelloEclipse

main()

Developing Eclipse Rich-Client Applications  |  © 2007 by Frank Gerhardt and Michael Scharf

**rcpsimple**

MyApplication

start()

depends   impl

⬛ myapplication

declares

⬛ application

**IApplicaiton**

**start()**

**org.eclipse.equinox.app**

**rcpsimple**

MyApplication

run()

myapplication

declares

impl

depends

application

**IApplication**

**start()**

**org.eclipse.equinox.app**

depends

**org.eclipse.ui**

Developing Eclipse Rich-Client Applications | © 2007 by Frank Gerhardt and Michael Scharf

**rcpsimple**

create

create

**MyWorkbenchAdvisor**

getInitialPerspectiveId()

**MyWorkbenchWindowAdvisor**

preWindowOpen()

**MyActionBarAdvisor**

makeActions()
fillMenuBar()
fillActionBar()

new

**MyApplication**

run()

**MyPerspective**

createInitialLayout()

impl

myapplication

declares

calls

impl

myperspective

declares

depends

depends

**application**

**IPlatformRunnable**

**run()**

**perspectives**

IPerspectiveFactory
createInitialLayout()

PlatformUI
createAndRunWorkbench()

**org.eclipse.core runtime**

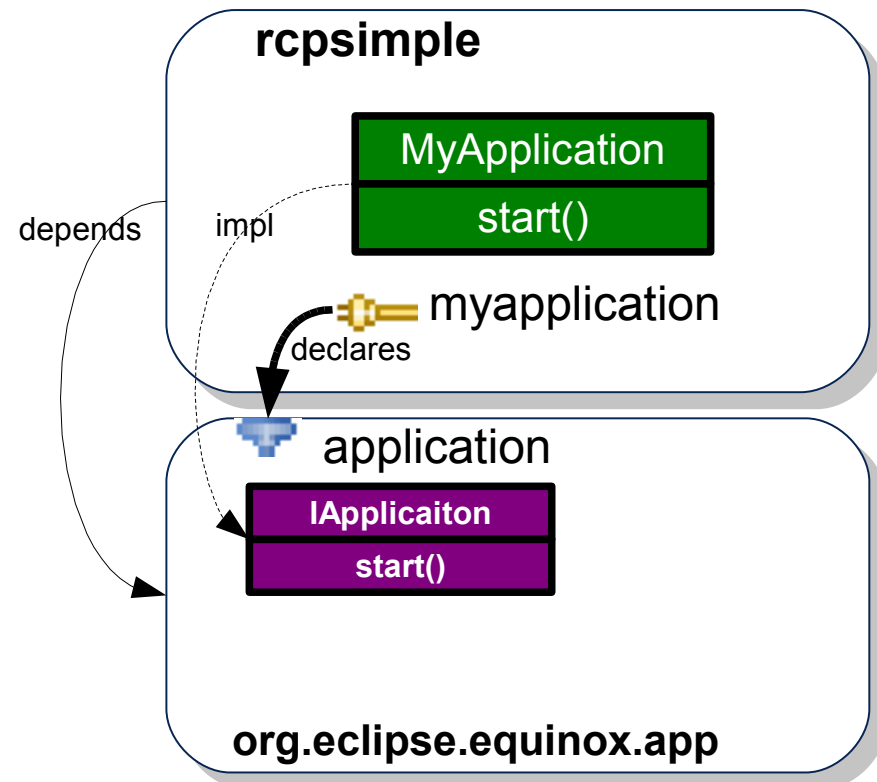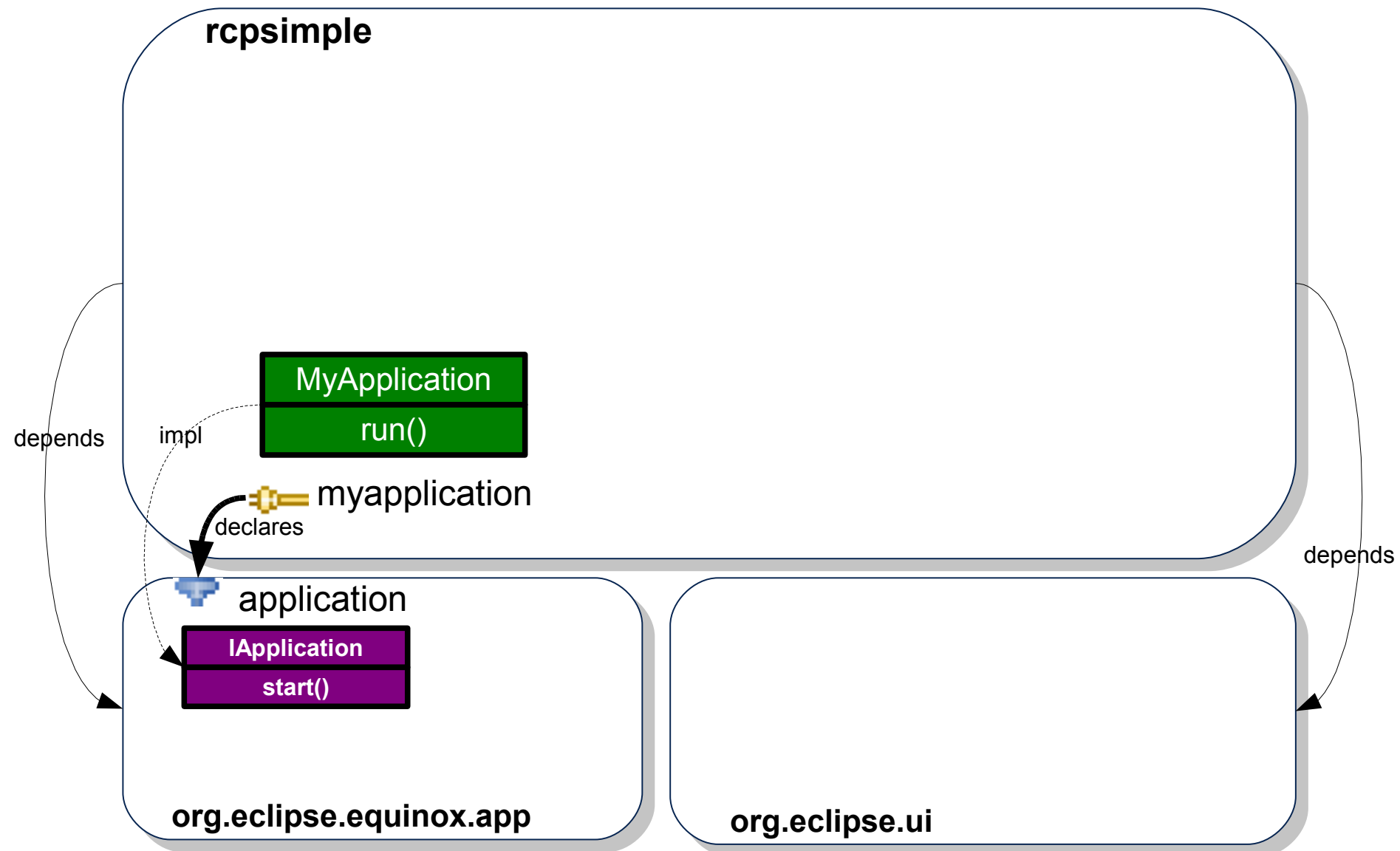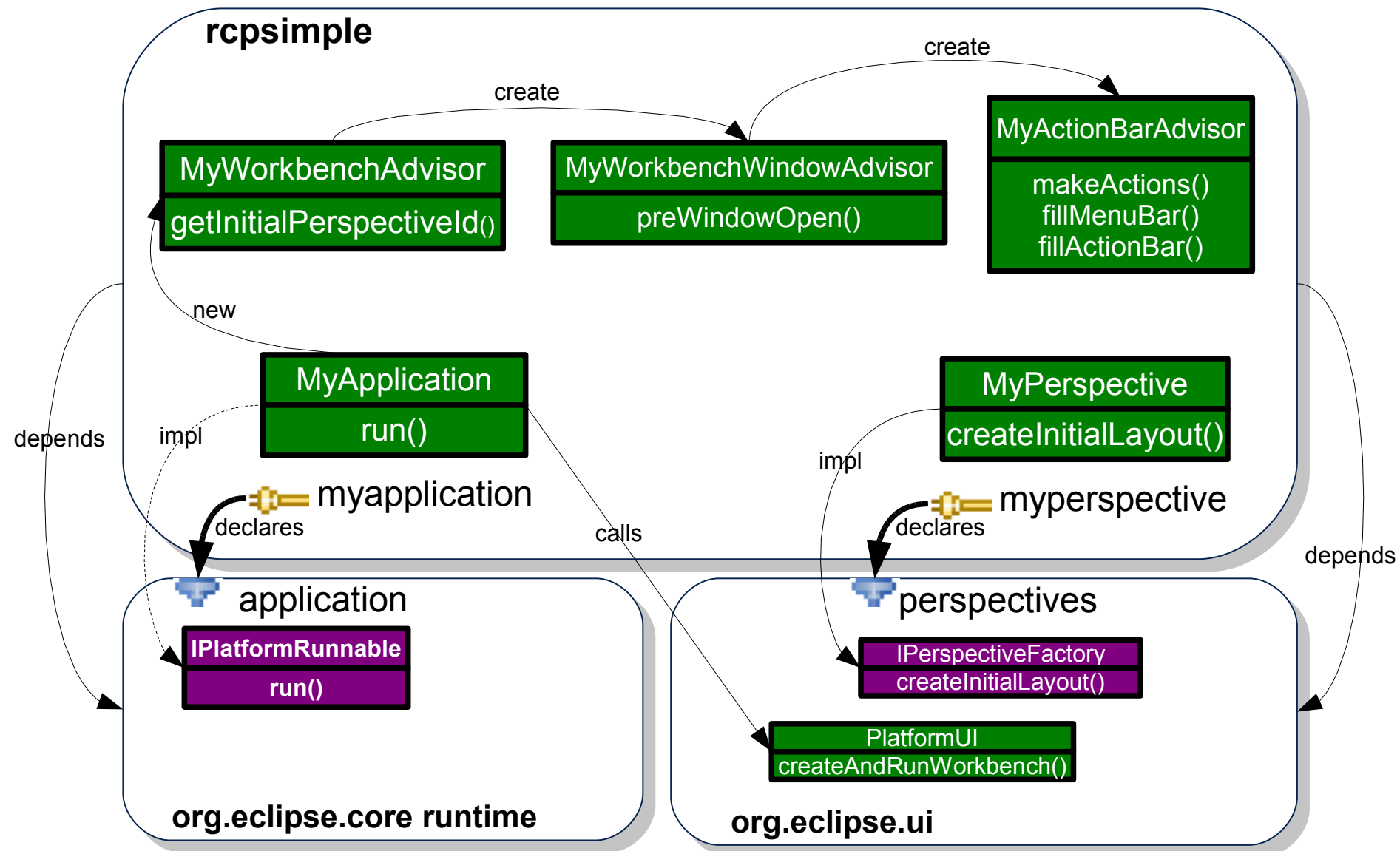**org.eclipse.ui**

# Attention, Attention!

- Watch us pair-programming
- Listen to us
- Ask us questions
- Try to understand what, why, how
- Observe the way we use the IDE

- Do not work in parallel
  - It is not an exercise in synchronous mouse-clicking
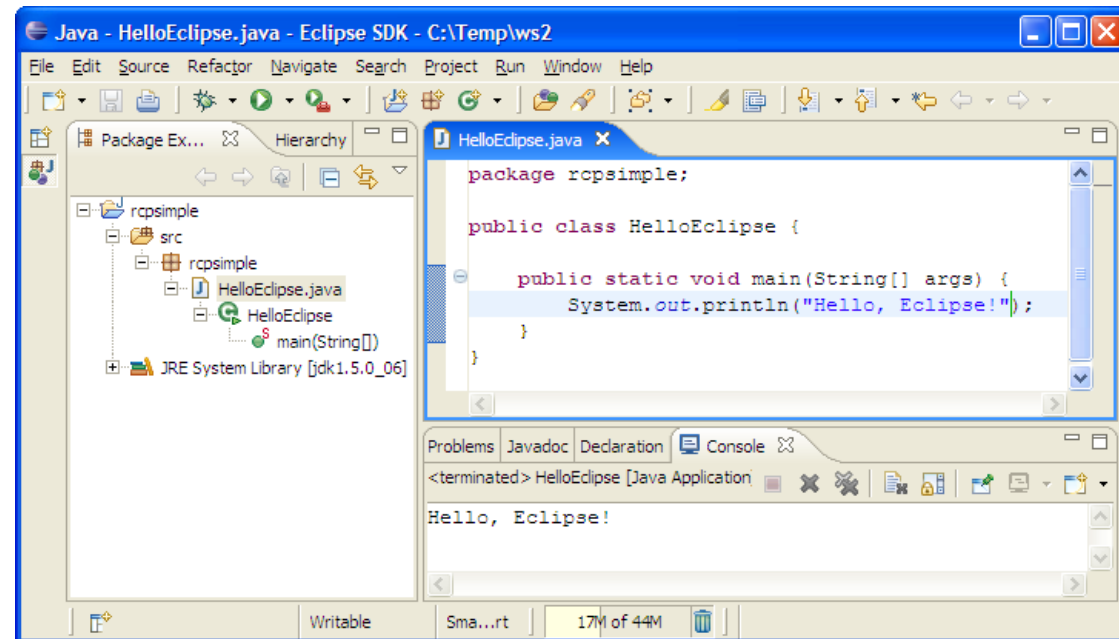  - You will miss many important points

- You will have enough time to do the exercises

- We have some additional exercises if you finish faster than the others, or have a look at the Tips&Tricks PDF.

# Exercise: Hello, Eclipse – POJO style

- Write a hello world application in
  - a Java project called rcpsimple
  - Package rcpsimple
- Run it

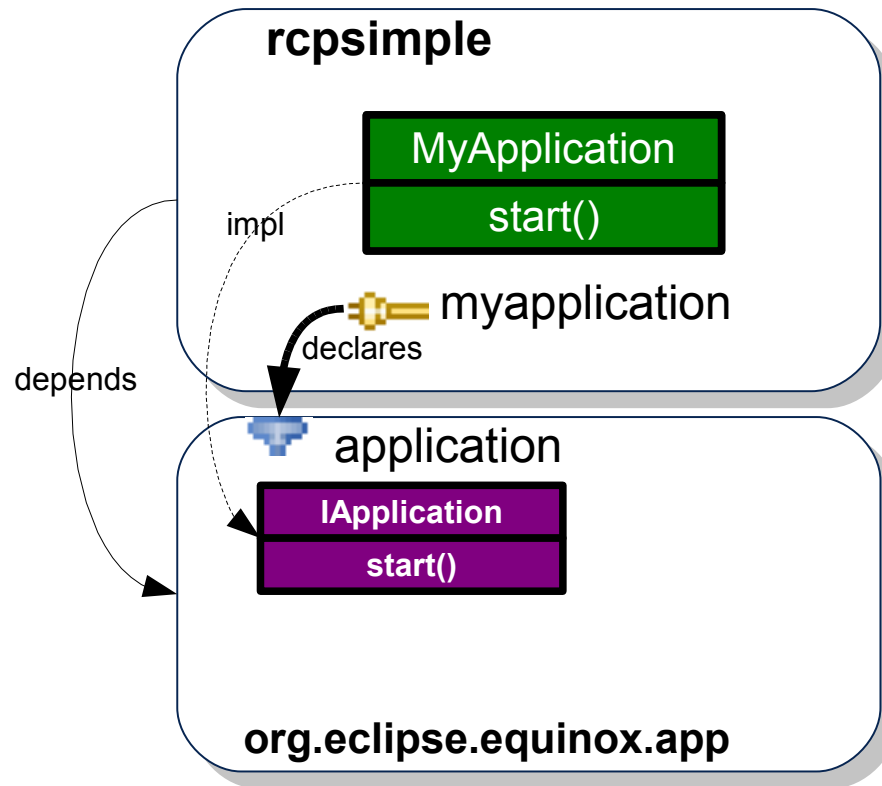**HelloEclipse**

**main()**

Developing Eclipse Rich-Client Applications  |  © 2007 by Frank Gerhardt and Michael Scharf

**rcpsimple**

HelloEclipse

main()

Developing Eclipse Rich-Client Applications | © 2007 by Frank Gerhardt and Michael Scharf

Developing Eclipse Rich-Client Applications | © 2007 by Frank Gerhardt and Michael Scharf

**Hello Plug-in**

**Application**

**run()**
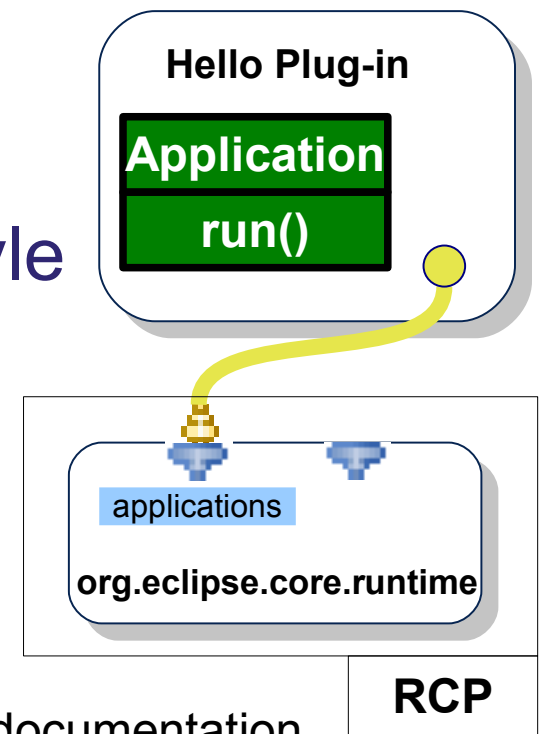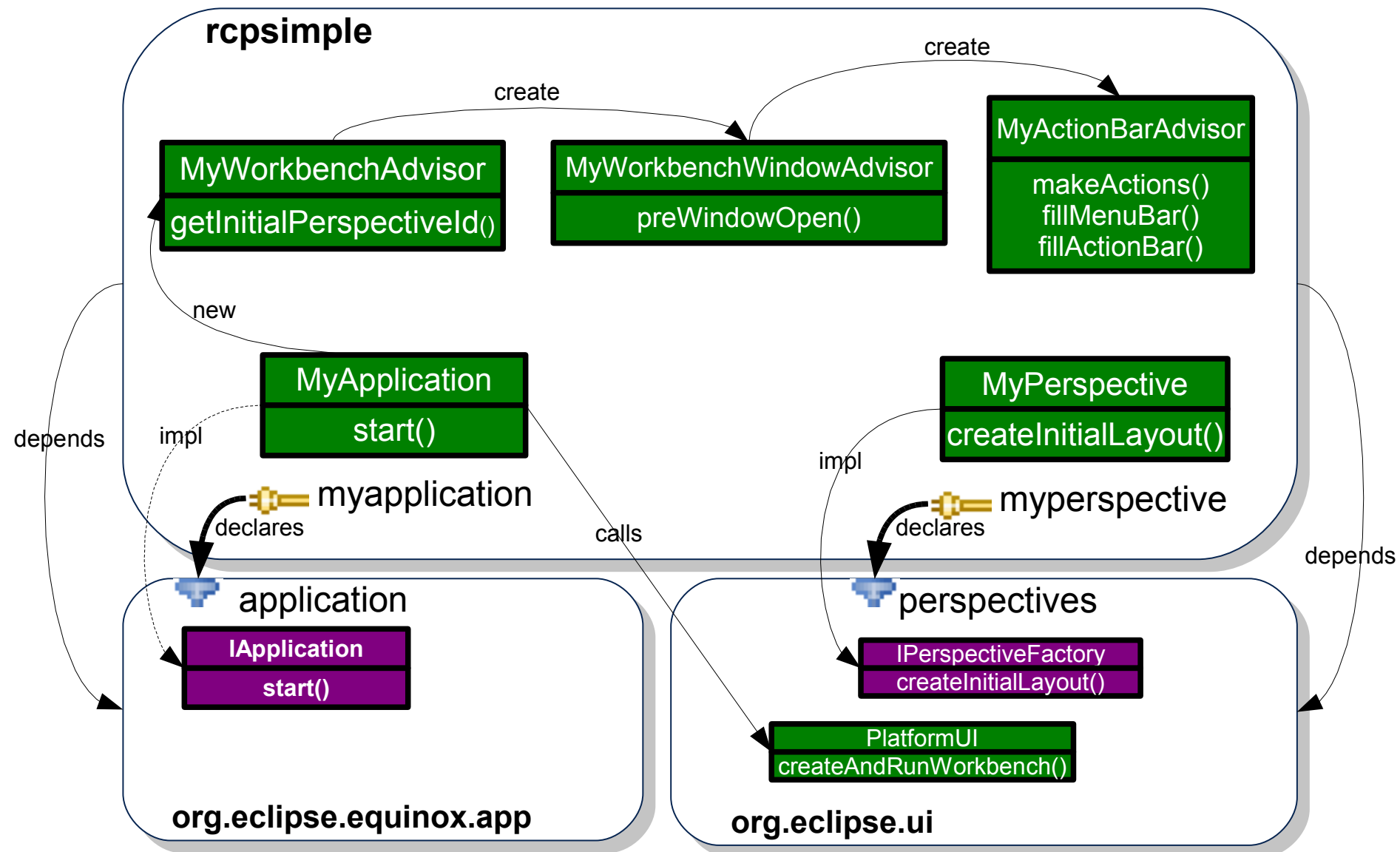
# Exercise: Hello, Eclipse – Plug-in style

- Goal: run Hello, Eclipse! as an RCP application
- Convert the project to a plug-in project
    - update classpath
- Create an application extension
    - Which, where?
    - Search for extension point *application*, browse documentation
- Add extenstion application, accept to add dependency
    - Uncheck „Show only extension points from the required plug-ins"
        - (alternateively declare a dependency on o.e.core.runtime first)
- Define the application extension (with children)
    - run, get ClassCastException, -consoleLog, look at the Eclipse code
    - adapt application class, add IplatformRunnable, move method body
    - Rename HelloEclipse to MyApplication

applications

**org.eclipse.core.runtime**

**RCP**

# Optional Exercise: Print out command line params

- Look at context

**rcpsimple**

MyWorkbenchAdvisor
getInitialPerspectiveId()

create

MyWorkbenchWindowAdvisor
preWindowOpen()

create

MyActionBarAdvisor
makeActions()
fillMenuBar()
fillActionBar()

new

MyApplication
start()

MyPerspective
createInitialLayout()

impl

myapplication

declares

calls

impl

myperspective

declares

depends

depends

application

**IApplication**
**start()**

perspectives

IPerspectiveFactory
createInitialLayout()

PlatformUI
createAndRunWorkbench()

**org.eclipse.equinox.app**

**org.eclipse.ui**

Developing Eclipse Rich-Client Applications | © 2007 by Frank Gerhardt and Michael Scharf

# Exercise: Open a Workbench window

- Goal: create and run the workbench
  - Clue: PlatformUI.createAndRunWorkbench(display, advisor);
- Want to search in Eclipse jar files?
  - Add all plug-ins to Java Search
- Where is PlatformUI?
  - Add all plug-ins to Java search, use Open Type dialog
- Add dependency
- Use Quick Fixes to flesh out the method backwards
- Which display?
  - Clue: get display from PlatformUI
- Which advisor to use?
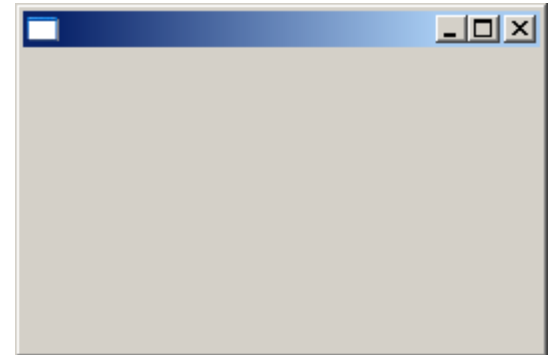  - Roll your own
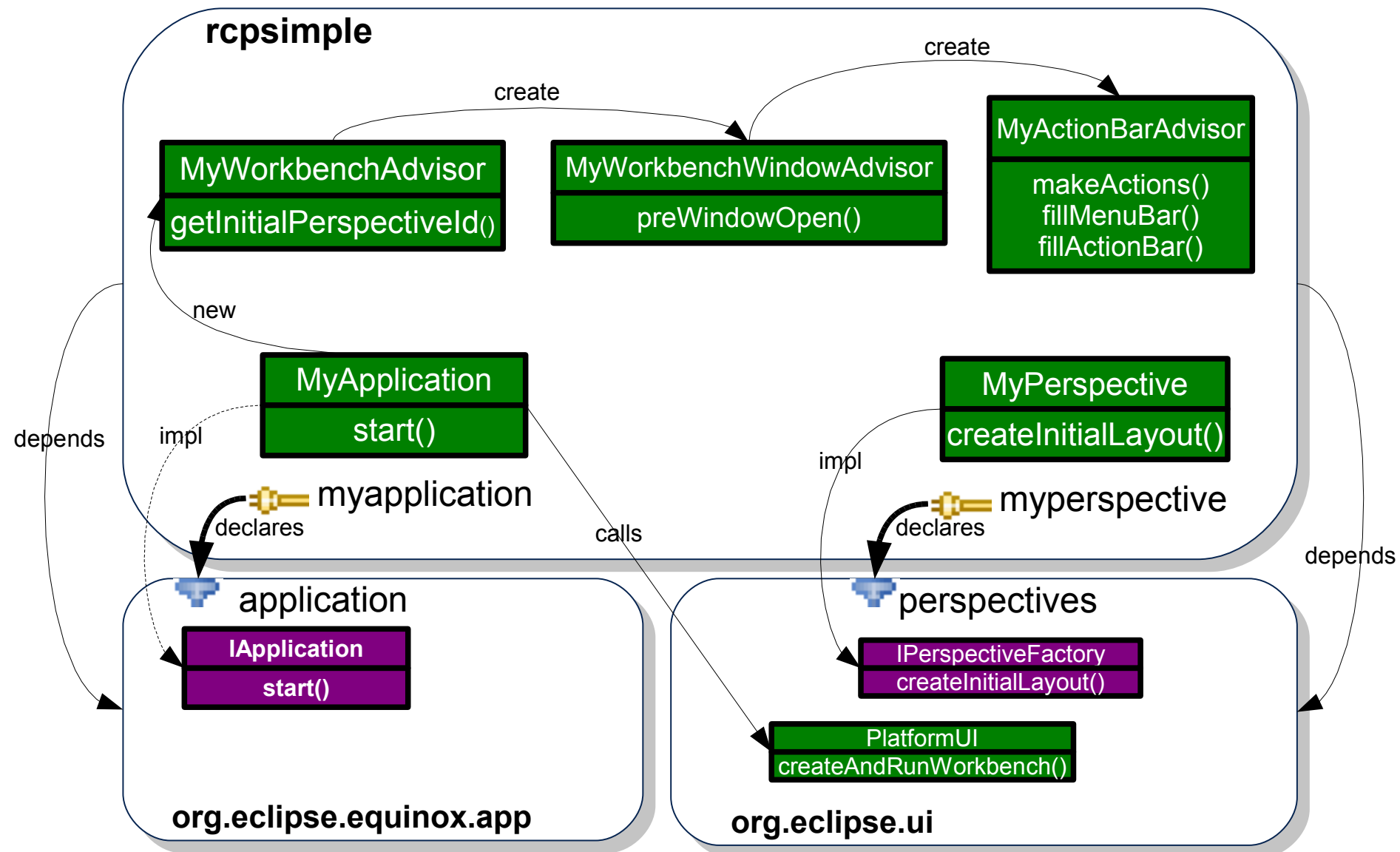- Run it
  - Update launch config
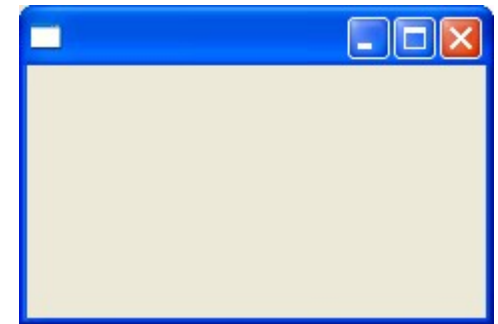
# Optional Exercise: Customize window

- Change size
- Show status line
- Change window title

# Exercise: Add a default perspective

- Return a perspective id
  - run (fails)
- Define an extension myperspective
  - Run (works)
  - Window shows up

**rcpsimple**

create

create

**MyWorkbenchAdvisor**

getInitialPerspectiveId()

**MyWorkbenchWindowAdvisor**

preWindowOpen()

**MyActionBarAdvisor**

makeActions()
fillMenuBar()
fillActionBar()

new

**MyApplication**

start()

**MyPerspective**

createInitialLayout()

depends

impl

myapplication

declares

calls

impl

myperspective

declares

depends

application

**IApplication**

**start()**

perspectives

**IPerspectiveFactory**
createInitialLayout()

PlatformUI
createAndRunWorkbench()

**org.eclipse.equinox.app**

**org.eclipse.ui**

Developing Eclipse Rich-Client Applications  |  © 2007 by Frank Gerhardt and Michael Scharf
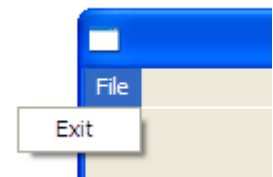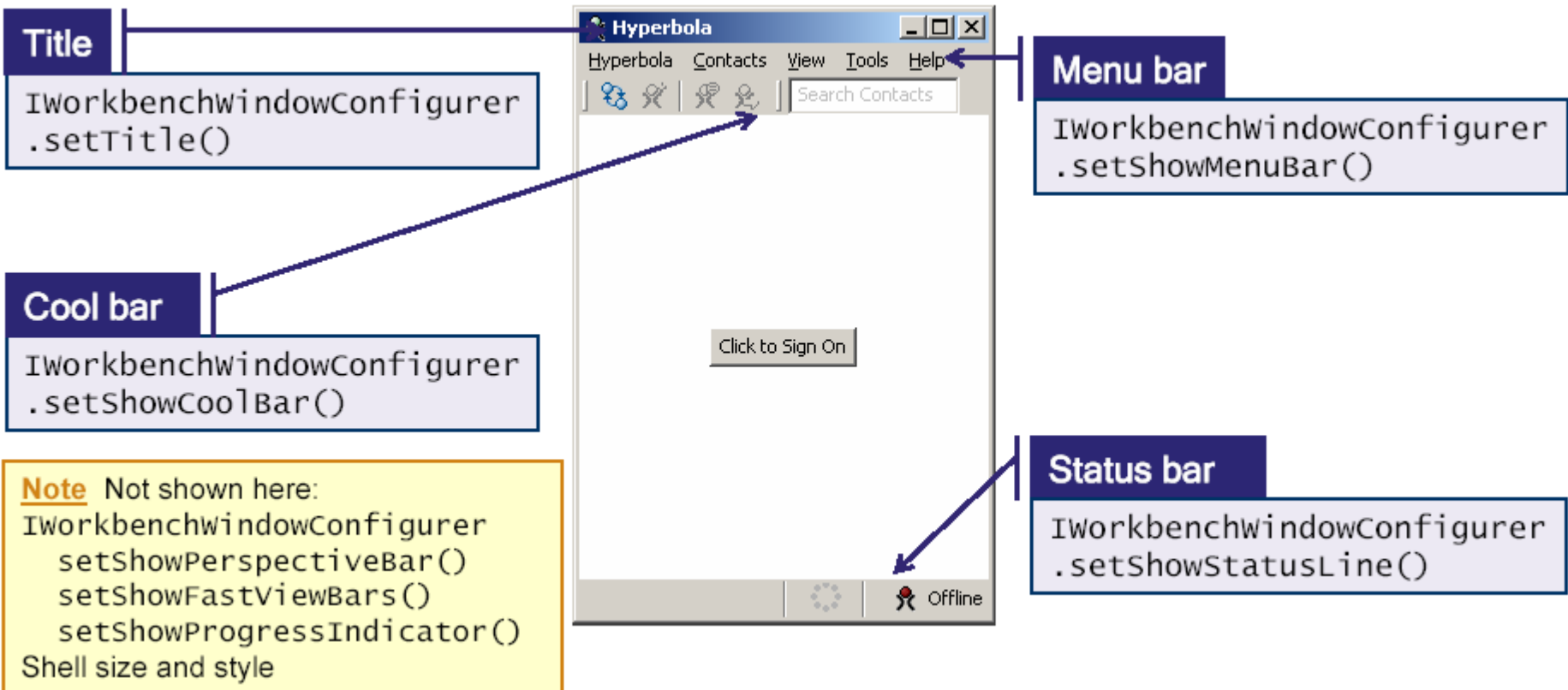
# Exercise: Create a WorkbenchWindowAdvisor

- Clue: Override createWorkbenchWindowAdvisor
  - Roll your own
  - Run
- Change the window size
  - Clue: preWindowOpen
  - Clue: get the window configurer
- Monkey see, monkey do
  - Have a look what others are doing here, browse Eclipse sources
  - Set size, hide the coolbar
  - Run (smaller window, looks nicer)

# Exercise: create an ActionBarAdvisor

- Clue: createActionBarAdvisor
  - Roll your own
  - Run
- Monkey see, monkey do
  - Browse subclasses of ActionBarAdvisor
    - Clue: WorkbenchActionBuilder
- Override in MyActionBarAdvisor
  - MakeActions
    - Create quit action
      - Hint: look for an action factory and how it is used by clients
  - fillMenuBar
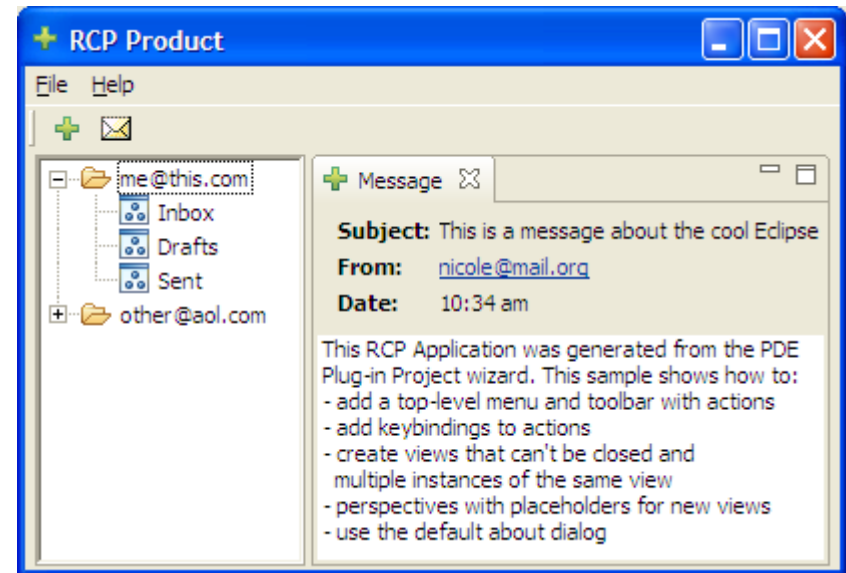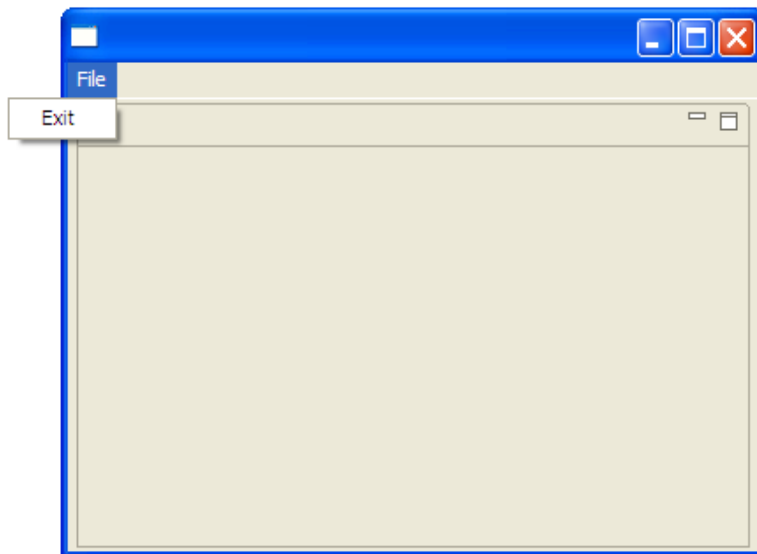    - create File menu
- Run

**Title**

```
IWorkbenchWindowConfigurer
.setTitle()
```

**Menu bar**

```
IWorkbenchWindowConfigurer
.setShowMenuBar()
```

**Hyperbola**

Hyperbola   Contacts   View   Tools   Help

Search Contacts

Click to Sign On

Offline

**Cool bar**

```
IWorkbenchWindowConfigurer
.setShowCoolBar()
```

**Status bar**

```
IWorkbenchWindowConfigurer
.setShowStatusLine()
```

**Note** Not shown here:
```
IWorkbenchWindowConfigurer
   setShowPerspectiveBar()
   setShowFastViewBars()
   setShowProgressIndicator()
```
Shell size and style

```
public HyperbolaWorkbenchWindowAdvisor extends WorkbenchWindowAdvisor {

  HyperbolaWorkbenchWindowAdvisor(IWorkbenchWindowConfigurer configurer) {
    super(configurer);
  }

  public void preWindowOpen() {
    IWorkbenchWindowConfigurer configurer = getWindowConfigurer();
    configurer.setTitle("Hyperbola");
    configurer.setInitialSize(new Point(275, 475));
    configurer.setShowProgressIndicator(true);
    configurer.setShowPerspectiveBar(false);
  }

  public ActionBarAdvisor createActionBarAdvisor(
                               IActionBarConfigurer configurer) {
    return new HyperbolaActionBarAdvisor(configurer);
  }
```
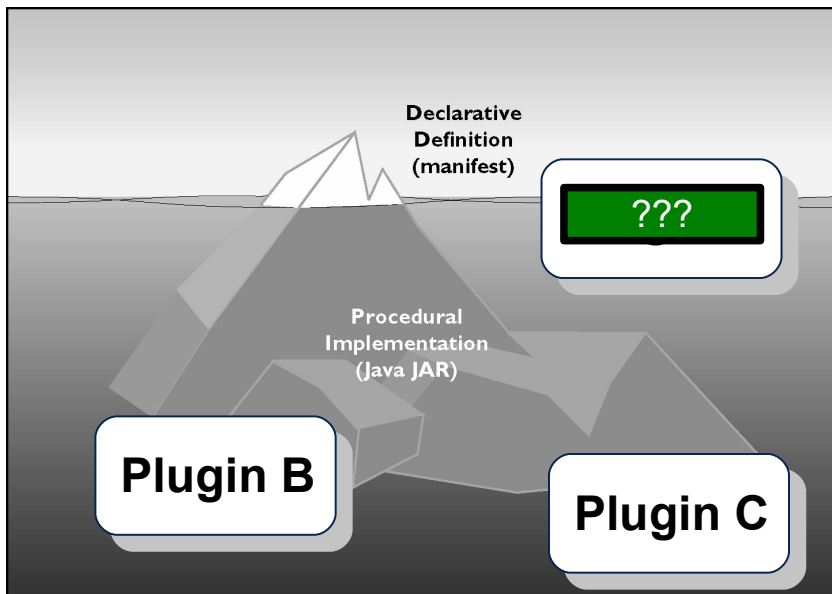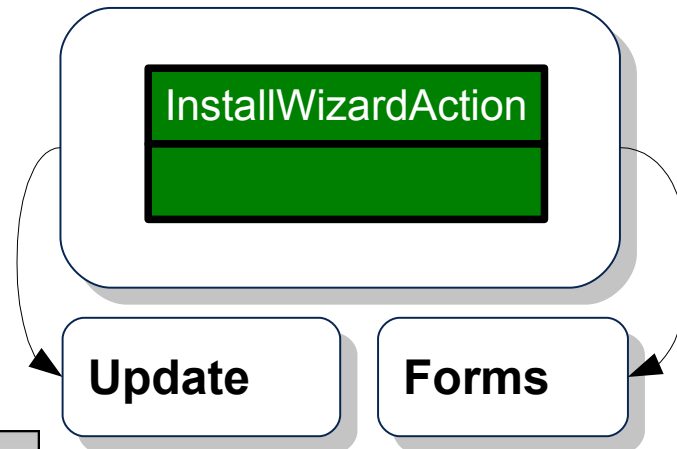
Layout and appearance settings

Factory method for the action bar advisor

# Tutorial Overview

- Part I: From Hello, World! to Workbench window

- Part II: From RCP Mail to a product with

  - Update Manger

  - Help System

**rcpmail**

InstallWizardAction

Update    Forms

Declarative
Definition
(manifest)

???

Procedural
Implementation
(Java JAR)

**Plugin B**

**Plugin C**

**rcpmail product**

**rcpmail-feature**

**Deploy to c:\temp**

rcpmail

rcpmail 1.0.0

InstallWizardAction

refers to

**rcpmail Update Site**

**Update** → **Forms**

includes

**Rich Client Platform**

Developing Eclipse Rich-Client Applications | © 2007 by Frank Gerhardt and Michael Scharf

**rcpmail product**

**rcpmail-feature**

**Deploy to c:\temp**

rcpmail 1.0.0

update

refers to

deploy

**rcpmail Update Site**

**rcpmail 1.0.1**

**rcpmail**

InstallWizardAction

**Update** → **Forms**

includes

**rcphelp**

**Table of contents**

**Help...**

**Rich Client Platform**

Developing Eclipse Rich-Client Applications | © 2007 by Frank Gerhardt and Michael Scharf

# Rich client application as product - rcpmail

- An Eclipse product is a stand-alone program
- May be packaged and delivered as one or more features
- Defines the application to run
- Products add branding to applications





RCP feature

ank Gerhardt and Michael Scharf
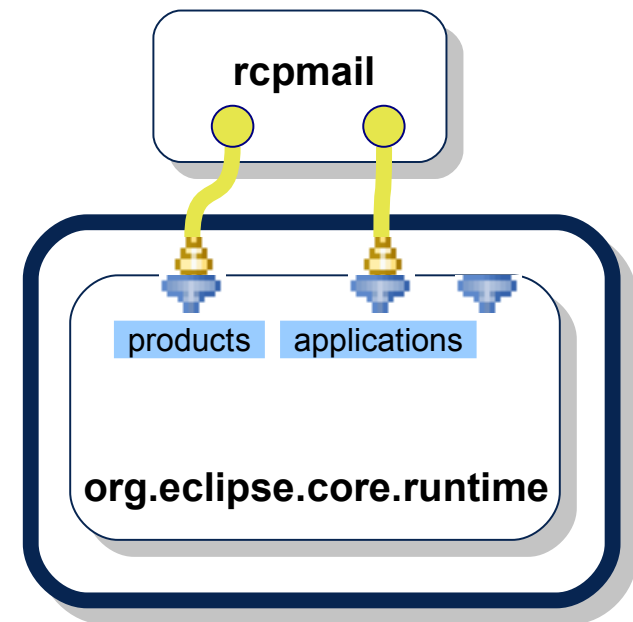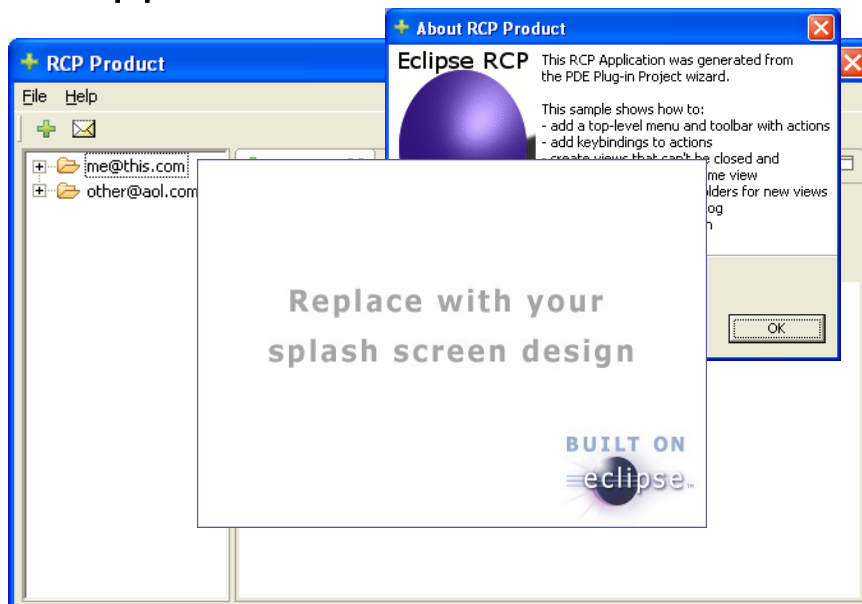
# Exercise: Generate RCP Mail

- Generate application called „rcpmail" using the RCP mail template
- Observe splash screen and about dialog
- Inspect „product" extension
- Inspect runtime configuration

**Plug-in Properties**

Plug-in ID: rcpmail
Plug-in Version: 1.0.0
Plug-in Name: Rcpmail Plug-in
Plug-in Provider:
Classpath:

**Plug-in Options**

☑ Generate an activator, a Java class that controls the plug-in's life cycle
  Activator: rcpmail.Activator
☑ This plug-in will make contributions to the UI

**Rich Client Application**
Would you like to create a rich client application? ⊙ Yes ○ No

**Configuration Area**
☑ Use default location
  Location: ${workspace_loc}/.metadata/.plugins/org.eclipse.pde.core/Eclipse Application
☐ Clear the configuration area before launching    Workspace...  File System...  Variables...

osgi.splashPath=file\:c\:/tools/eclipse/workspace-rcptut/rcpmail/

☑ Create a plug-in using one of the templates

Available Templates:

- 🧩 Hello RCP
- 🧩 RCP application with an intro
- 🧩 RCP application with a view
- 🧩 **RCP Mail Template**

⊙ Choose plug-ins and fragments to launch from the list

☐ ☑ Workspace Plug-ins
  └ ☑ rcpmail (1.0.0)
☐ ☐ Target Platform
  ├ ☐ com.ibm.icu (3.4.2)
  └ ☐ com.ibm.icu.source (3.4.2)

- org.eclipse.core.runtime.produc
  └ X RCP Product (product)
    └ X RCP Product (product)
      ├ X aboutText (property)
      ├ X windowImages (property
      └ X **aboutImage (property)**

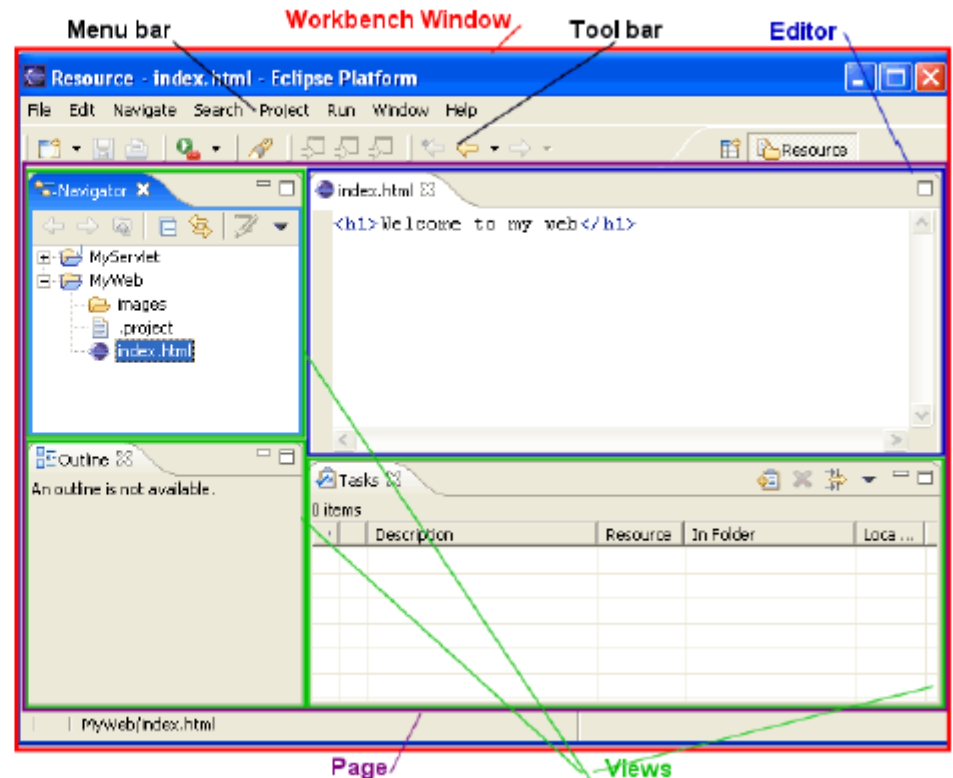# Workbench Structure

**Workbench (1)**

Window (0 – N)

**Page (0 – 1)**
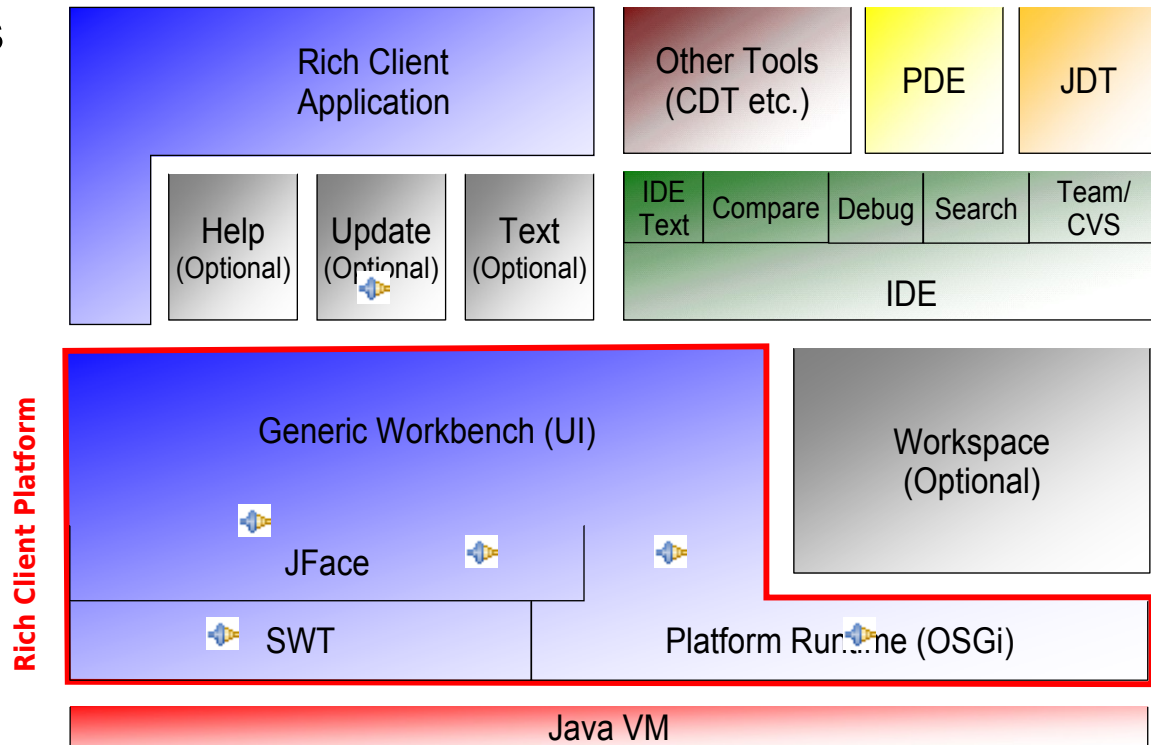
Perspective (0 – N), 1 active

**View (0 – N)**

**Editor (0 – N)**

# Add Update Manager

- Update Manager allows to update features in deployed application
- But, Update Manager UI is not part of RCP
- You can reuse all plug-ins from SDK
- You can add external plug-ins as well
- We have to distribute all plug-ins, which we depend on

# Exercise: Add Update Manager

- Import all plug-ins as binary plug-ins
- File search for Find and Install in *.properties files
- Find the user the key
  - Be aware of &
- Copy the extension to rcpmail
- Review and adapt the extension
- Copy resources as needed
- Add dependecy
- Configure plug-in localization
- Update launch config
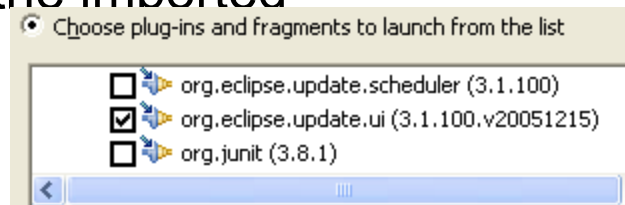- If it does not run: delete the imported plugins)

!MESSAGE Bundle update@plugins/rcpmail_1.0.0.jar was not resolved.
!SUBENTRY 2 rcpmail 2 0 2006-03-03 13:49:57.271
!MESSAGE Missing required bundle org.eclipse.update.ui_0.0.0.

# Features

- Features group plug-ins and features
- Eclipse is packaged as as set of features
- Configuration manager works with features
- Features can be updated – orphaned plug-ins cannot

**Feature X**

**Feature Y**

**Feature Y**

**Plugin A**
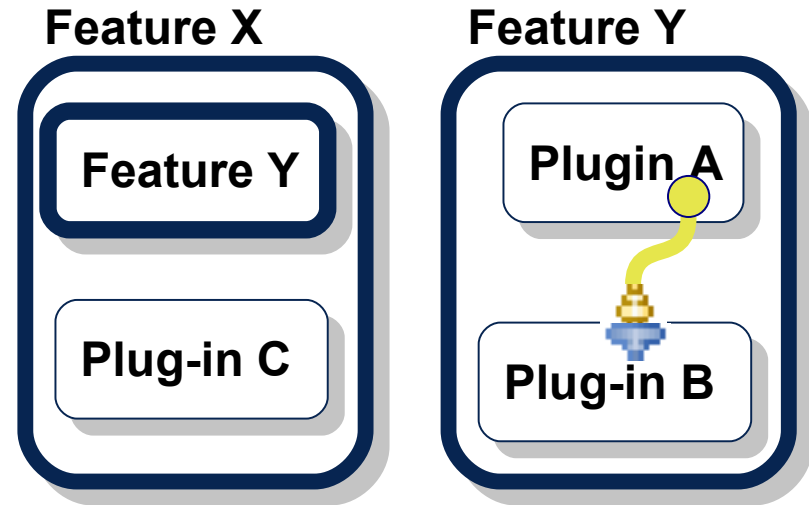
**Plug-in C**

**Plug-in B**

# Exercise: Create rcpmail-feature

- Update URL
    - file:/temp/rcpmail-site/
    - Do put the slash at the end!!!!
- Add all plug-ins required for Update Manager
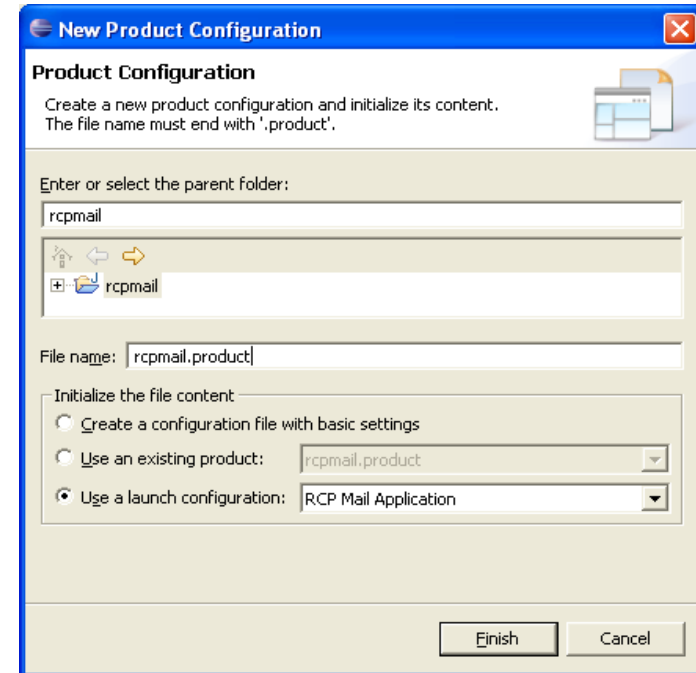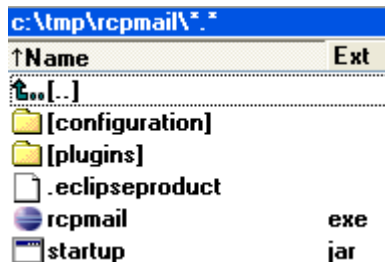    - See launch config

# Product Export

- Product export requires
  - Underlying plug-ins / features
  - Config file
  - Launcher
- All components need to be packaged
- Eclipse provides product export wizard for this purpose
- Product configuration file contains necessary information

Rcpmail feature

RCP feature
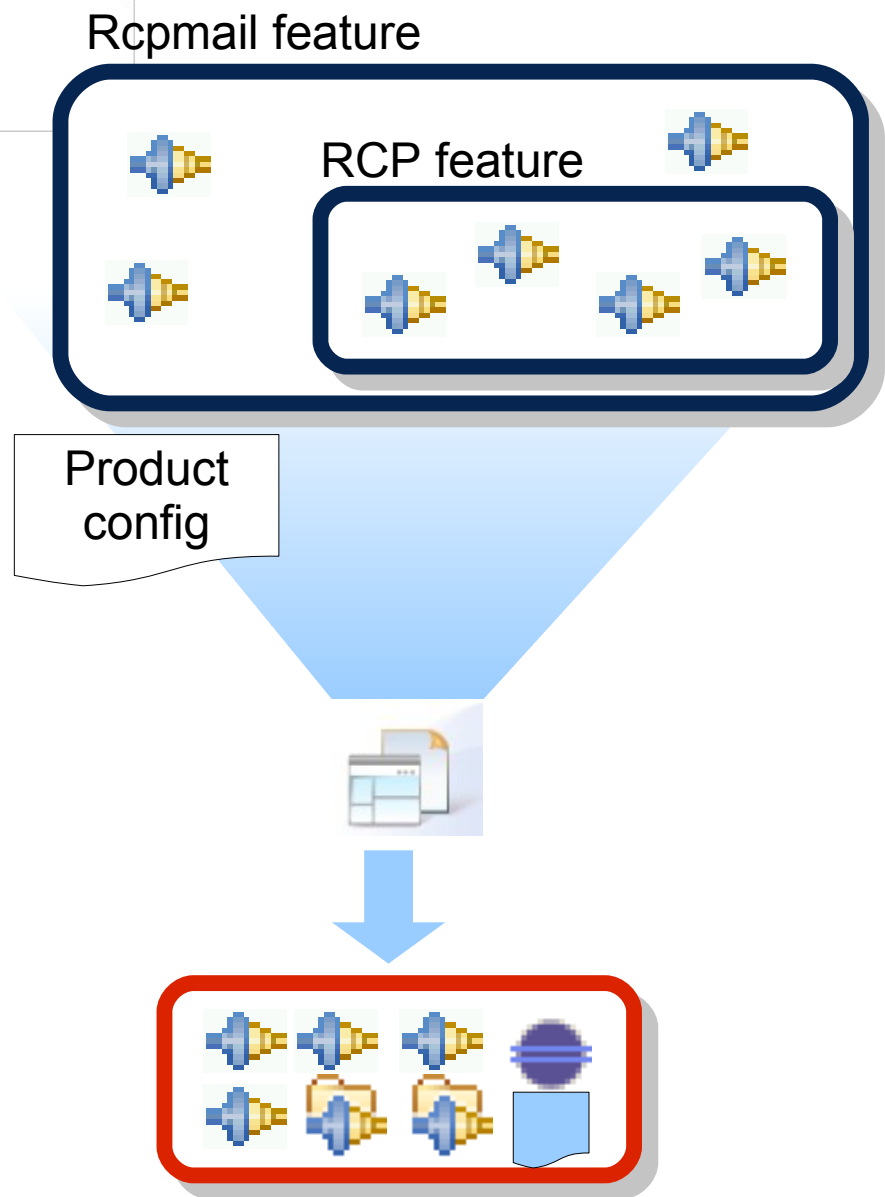
Product config

# Exercise: Export product using wizard

- Check launch config
- Add *.properties to Build
- Create product configuration
    - Based on launch config
    - Feature-based, add rcpmail feature
    - Generate default config.ini
- Export using the wizard
- Run rcpmail product

**New Product Configuration**

**Product Configuration**
Create a new product configuration and initialize its content.
The file name must end with '.product'.

Enter or select the parent folder:

rcpmail

⊞ 📂 rcpmail

File name: rcpmail.product

Initialize the file content
- ○ Create a configuration file with basic settings
- ○ Use an existing product:      rcpmail.product
- ● Use a launch configuration:   RCP Mail Application

Finish      Cancel

c:\tmp\rcpmail\*.*
| ↑Name | Ext |
|---|---|
| 📂..[..] | |
| 📁 [configuration] | |
| 📁 [plugins] | |
| 📄 .eclipseproduct | |
| 🌐 rcpmail | exe |
| 🗔 startup | jar |

**Program Launcher**
Customize the executable that is used to launch the product:

Launcher Name:

rcpmail

Export Destination
- ○ Archive file:
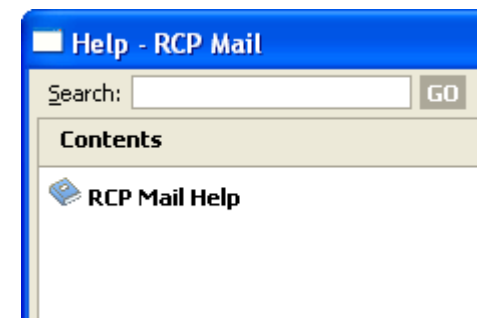- ● Directory:      C:\tmp

**rcpmail product**

**rcpmail-feature**

**Deploy to c:\temp**

rcpmail 1.0.0

update

**rcpmail Update Site**

refers to

deploy

**rcpmail 1.0.1**

**rcpmail**

InstallWizardAction

**Update** → **Forms**

**rcphelp**

**Table of contents**

**Help...**

includes

**Rich Client Platform**
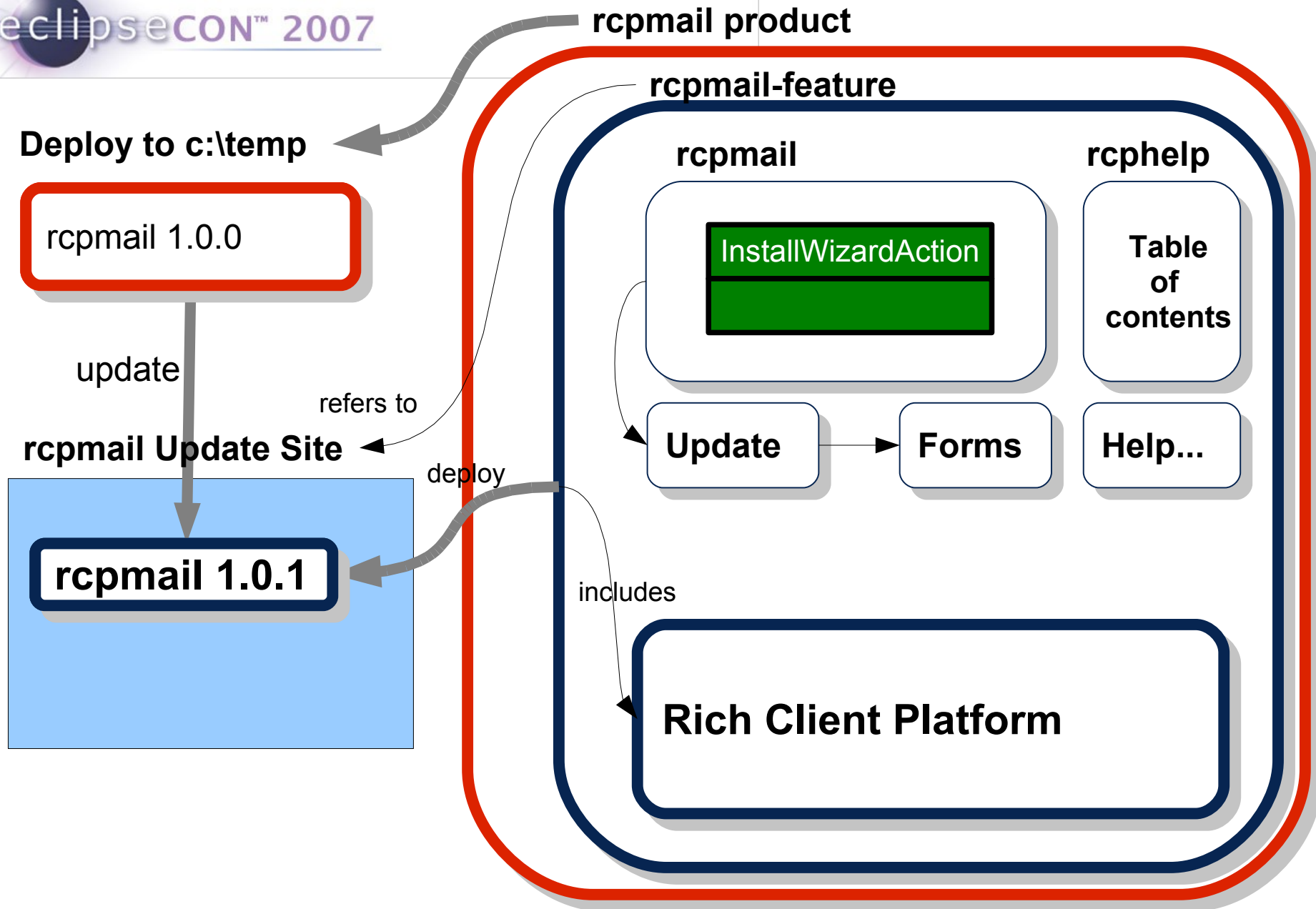
# Exercise: Add Help

- Generate Help plug-in
- Add help to action bar
- Update launch configuration
    - o.e.help.*
    - o.a.lucene
    - o.e.tomcat
- Update dependencies
- Add dependencies to rcpmail feature

| Help - RCP Mail |
|---|
| Search: [        ] [GO] |
| **Contents** |
| 📖 RCP Mail Help |

Developing Eclipse Rich-Client Applications | © 2007 by Frank Gerhardt and Michael Scharf

# Add Update Site

- Configure update site in feature descriptor
- Change your plugin
- Update version to 1.0.1 in plug-in and feature
- Create update site
- Build all
- Start RCP application
- Open Update Manager and update features
- Restart the application

Branding Plug-in: [                    ] [Browse...]

Update Site URL: [file:/tmp/rcpmail-site/]

Update Site Name: [                    ]

Synchronization Options
- ○ Force feature version into plug-in and fragment manifests
- ● Copy versions from plug-in and fragment manifests
- ○ Force versions defined in the feature into plug-in and fragment manifests

**Managing the Site**
1. Add the features to be published on the site.
2. For easier browsing of the site, categorize the
3. Build the features.

    rcpbase (1.0.0)
    rcpmail (1.0.1)

Select the features to install:
☑ file:/tmp/rcpmail-site/site.xml
    ☑ Other
       ☑ rcpmail-feature 1.0.1

# Add an Extension Point for Folders

- Create extension point schema in rcpmail
- Make TreeParent and TreeObject top level public classes
- Export rcpmail package from rcpmail plug-in
- Create new rcpcalendar plug-in
    - Add folder extension
    - Implement rcpcalendar.Calendar
    - Call super constructor with name „Calendar" as parameter
    - Add children, if you like
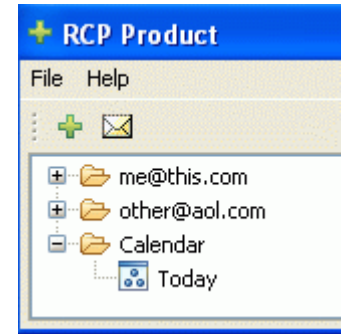
# Optional: Deploy again

- Add rcpcalendar to rcpmail feature
- Increment version numbers of rcpmail plug-in and feature
- Add new version of rcpmail to update site
- Update deployed rcpmail using update manager

# Summary: Key Concepts

- Plug-in / Bundle
  - The basic component in Eclipse with dependencies to other plug-ins / bundles.
- Extension Points and Extentions
  - The declarative wiring between the Eclipse framework and custom code
- Feature
  - A group of plug-ins with a version number
- Product
  - A deployable application with branding, defined in terms of plug-ins or features
- Update-site
  - A directory, usually on a web server, for the initial download and update of features

# Copyright

- © 2007 Frank Gerhardt and Michael Scharf
- Distributed under Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License
  http://creativecommons.org/licenses/by-nc-sa/3.0/us
- Contains material from „Developing for the RCP", Nick Edgar, Pascal Rapicault, EclipseCon 2005, licenced under Eclipse Public Licence 1.0

- Book:
  Eclipse Rich Client Platform: Designing,
  Coding, and Packaging Java Applications
  by Jeff McAffer and Jean-Michel Lemieux