



Rich Client Platform

Lukáš Bartoň
Hewlett-Packard

Netbeans Platform

For Rich Client Development

Jaroslav Tulach
Sun Microsystems

The Need for NetBeans and/or Eclipse

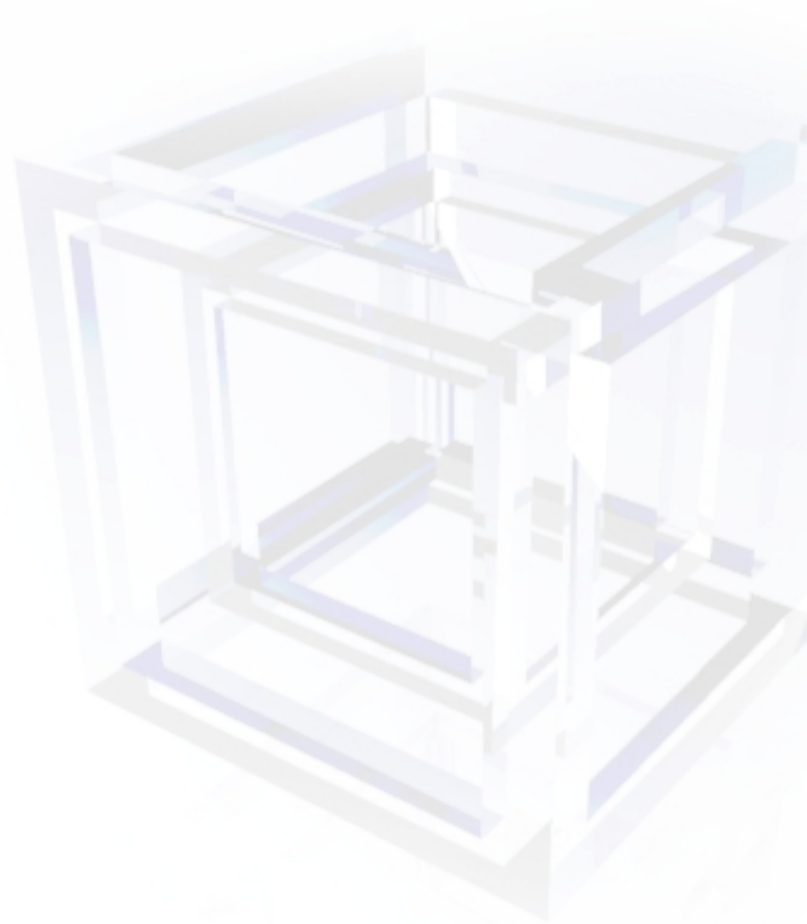
Don't write yet another framework, please!



Rest in piece to home made frameworks!

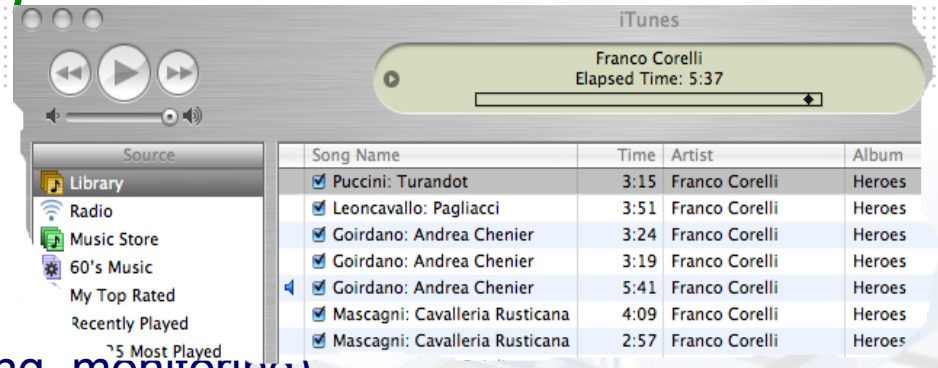
The Need for Modular Applications

- Applications get more complex
- Assembled from pieces
- Developed by distributed teams
- Components have complex dependencies
- Good architecture
 - Know your dependencies
 - Manage your dependencies



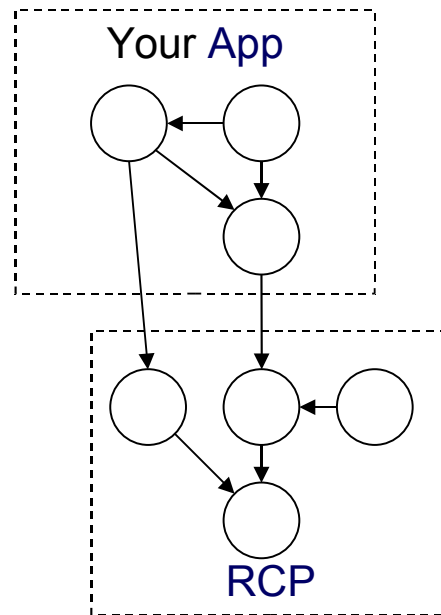
The Need for Rich Desktop Clients

- Web will not do it all
 - Real time interaction (dealing, monitoring)
 - Integration with OS (sound, etc.)
- 100% Java counts
 - Ease of administration and distribution
 - Plain Swing maybe too plain
- NetBeans Platform
 - The engine behind NetBeans IDE



Building Platforms (1/2)

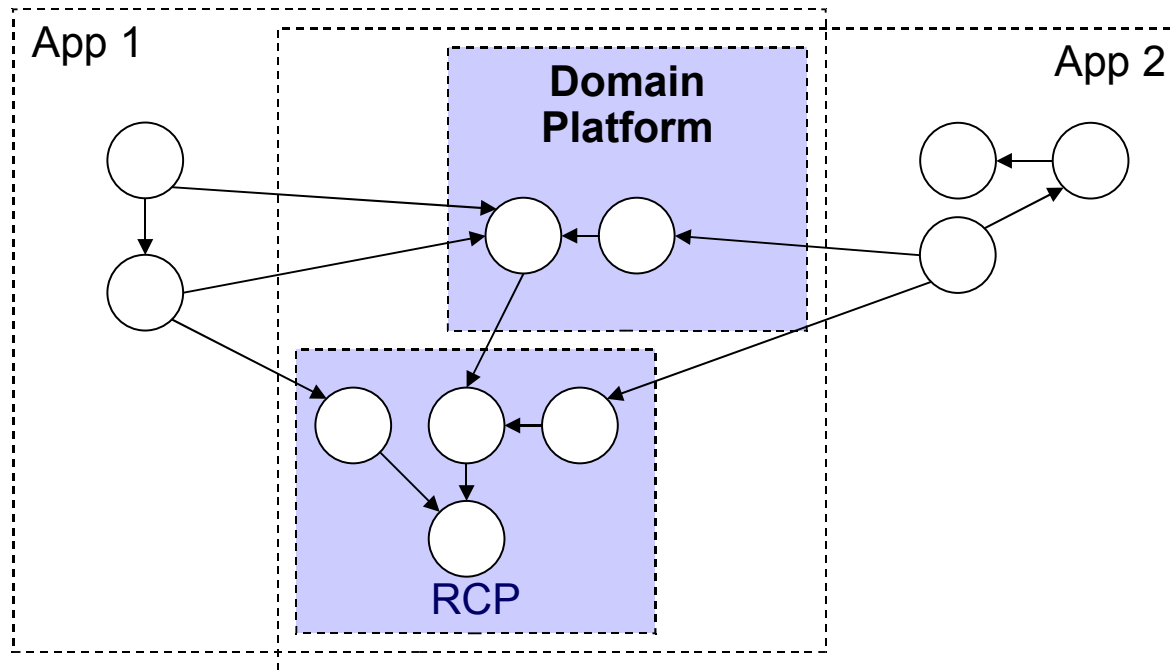
- It all starts with components
 - applications are composed of components that plug into the platform
- When starting development on Application, it is common to provide a handful of domain-specific components that sit directly on top of RCP



Building Platforms (2/2)



- It's natural for RCP development to spawn one or more "platforms"
 - A custom base for multiple development teams to build their applications upon



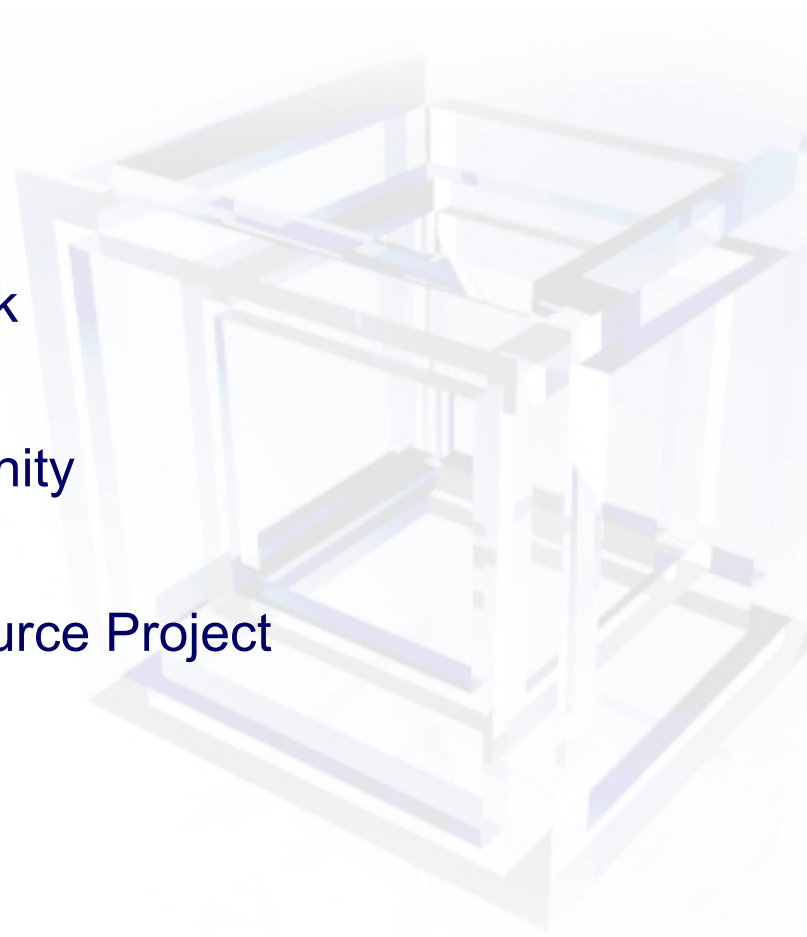


What is Eclipse?

- Eclipse is a Java IDE
- Eclipse is an IDE Framework
- Eclipse is a Tools Framework
- Eclipse is an Application Framework
- Eclipse is an Open Source Project
- Eclipse is an Open Source Community
- Eclipse is an Eco-System
- Eclipse is a Foundation

What is NetBeans?

- NetBeans is a Java IDE
- NetBeans is an IDE Framework
- NetBeans is a Tools Framework
- NetBeans is an Application Framework
- NetBeans is an Open Source Project
- NetBeans is an Open Source Community
- NetBeans is an Eco-System
- NetBeans is Sun sponsored Open Source Project



Eclipse Rich Client Platform

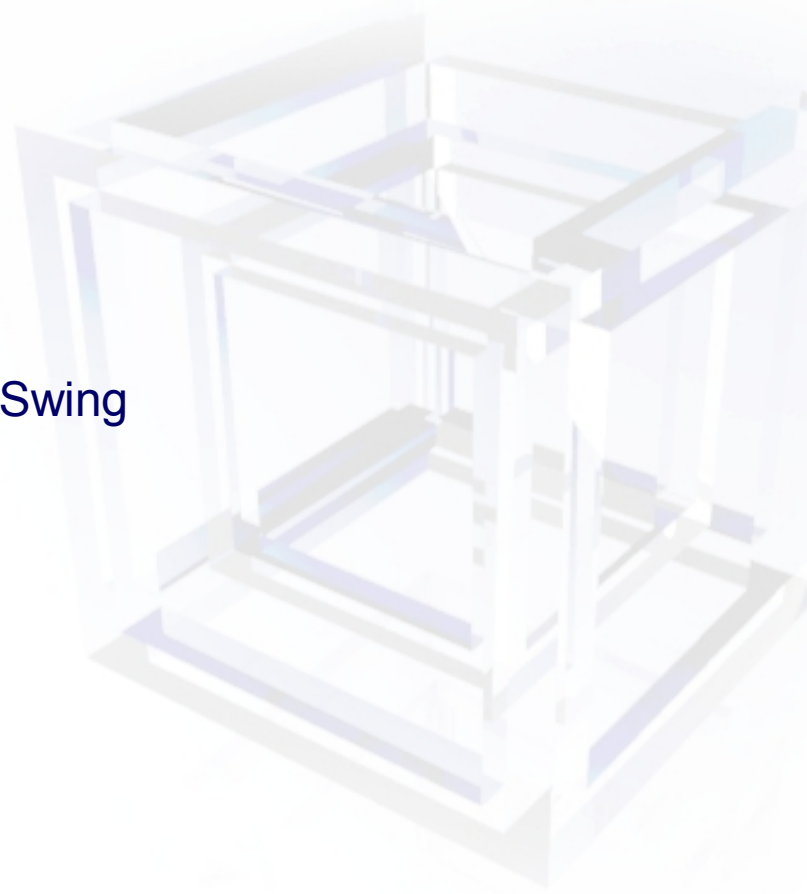


- SWT - The Standard Widget Toolkit
 - Access to the user-interface facilities of the operating systems on which it is implemented
- JFace
 - UI toolkit with classes for handling many common UI programming tasks that works with SWT
- UI
 - Base classes for UI components
- Equinox
 - Implementation of OSGI plug-in architecture
- Runtime
- Update

NetBeans Platform

for Rich Client Development

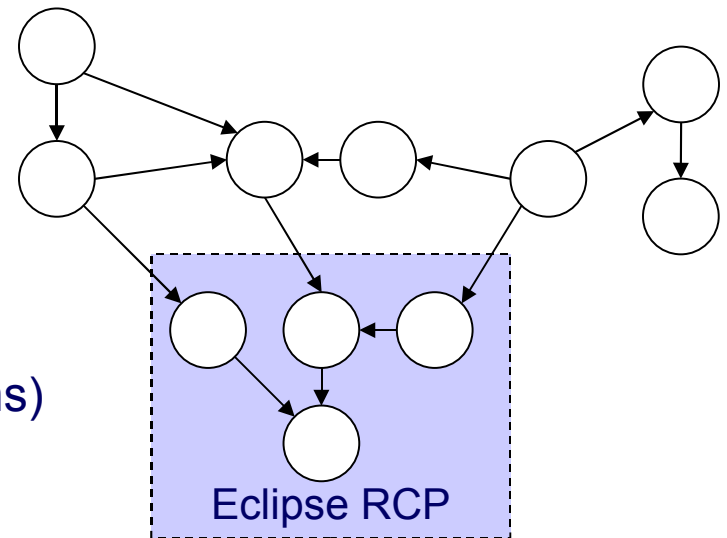
- 100% Java
 - no necessary 3rd party libraries
- Swing
 - the most standard Java toolkit
- Window System
 - docking framework extensions around Swing
- Module System
- Runtime
- Auto Update



Equinox (1/2)



- Equinox is the Eclipse component model
 - Based on OSGi R4 specification
 - Standard Java lacks an explicit notion of components
- Components == Bundles == Plug-in
 - Versioned
 - Defined declaratively
 - Dynamically loadable/unloadable
 - Support dynamic update and install
- Explicitly define
 - Dependencies
 - Runtime visibility
 - Interactions (extension points/extensions)



Equinox (2/2)



- Components integrate without interfering
 - Required components explicitly set
 - Unrelated components do not have direct access to one-another
- Downstream components can access upstream components through the extension mechanism
 - Downstream component registers (declaratively) an extension point
 - Dependent components register (declaratively) extensions
- Plug-in may be automatically started at platform startup
- All required plug-ins are loaded automatically
- Plug-in may have activator class

Modularity in NetBeans

Modify your application to be NetBeans module

- Module is any JAR file with enhanced manifest

```
Manifest-Version: 1.0
```

```
OpenIDE-Module:org.netbeans.modules.text/1
```

```
OpenIDE-Module-Specification-Version: 1.13
```

- What is this good for?
 - Identification of each library
 - Version specification
 - Dependencies

Modularity in NetBeans

Dependencies between modules

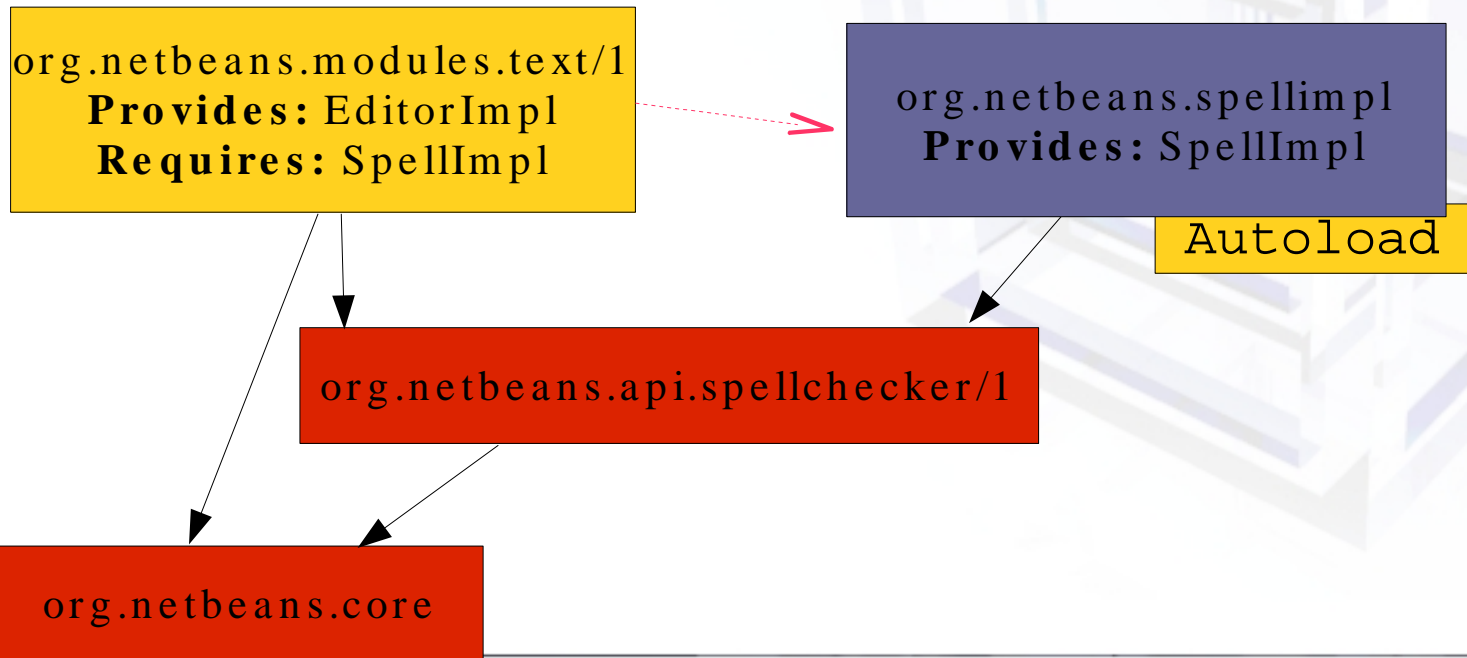
```
— OpenIDE-Module-Specification-Version: 1.13
OpenIDE-Module-Provides: EditorImpl
OpenIDE-Module-Module-Dependencies:
    org.netbeans.api.spellchecker/1 > 1.3,
    org.netbeans.core > 4.32
OpenIDE-Module-Requires: SpellImpl
```

- Types of modules
 - Regular
 - Autoload
 - Eager

Modularity in NetBeans

Module Enablement and ClassLoader Hierarchy

- Dependencies influence runtime classpath
- A module can turn other modules on



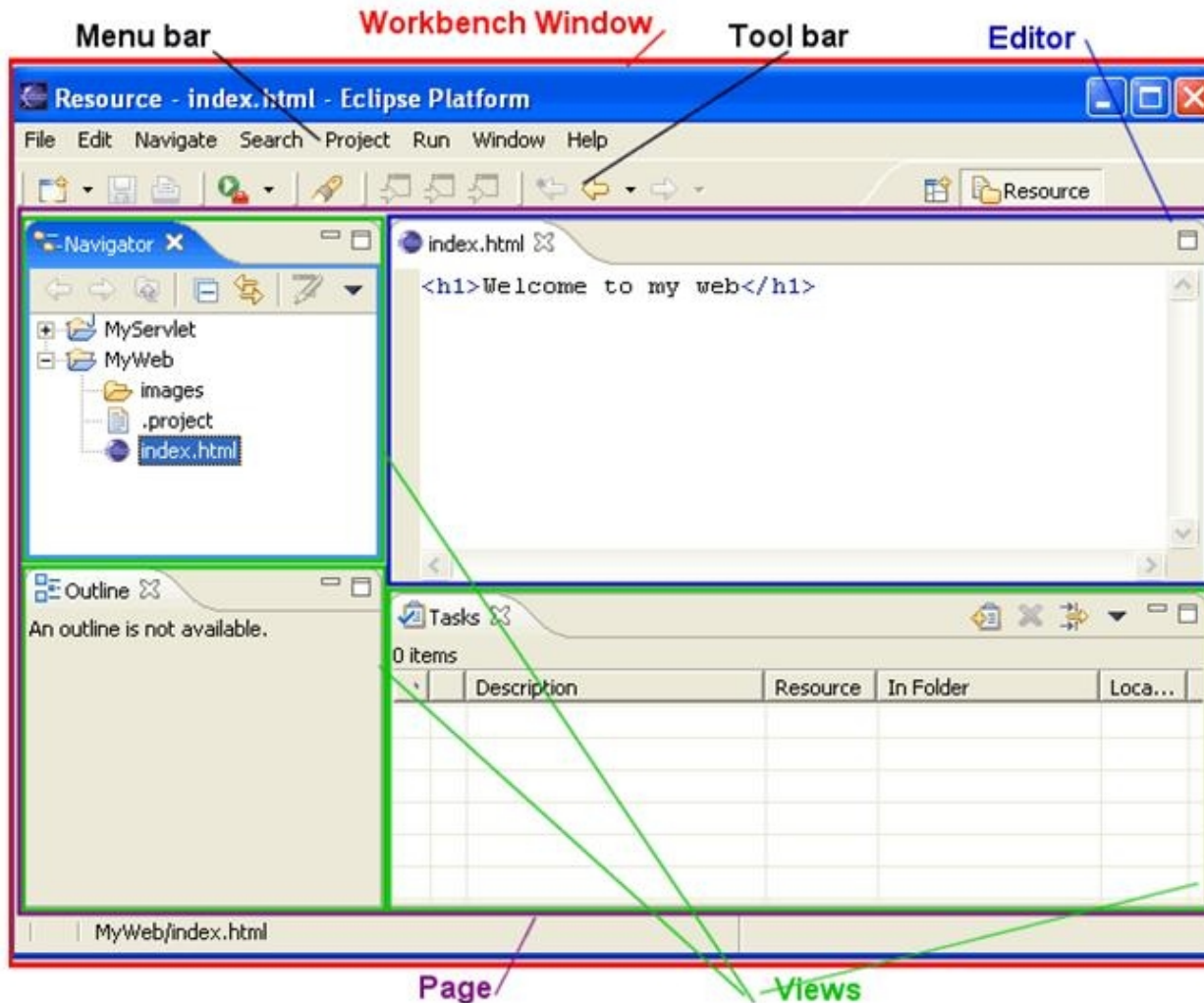


Eclipse Workbench (1/2)

- Fundamental component – appear as a collection of windows
- Provides contribution-based UI extensibility
- Defines a powerful UI paradigm with windows, perspectives, views, editors, and actions
- Allows you (developer) and user to works with those UI components

- You implement advisors which participate in:
 - Workbench startup and shutdown
 - Showing menu, toolbar, status line, configuring shown components
 - Programmatically define actions shown

Eclipse Workbench (2/2)





Views and Editors

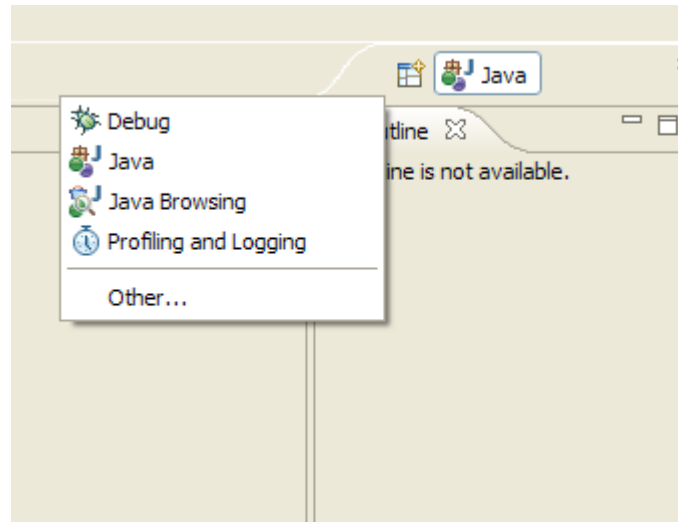
- == View Parts
- Views
 - There is usually one instance of a view
 - Views can appear all around Editor Area
 - Views can have local menus and toolbars
 - User can hide/show views from menu
- Editors
 - There could be several instances of the same type of editor
 - Editors can appear in only one region of the page
 - Editors can be in a dirty state
 - Editors don't have local menus and toolbars
 - User can open editor only by editing (creating new) resource
 - To open Editor you have to create EditorInput

Perspectives



- Group together and organize UI elements that relate to specific task
- Defines placement and size of views and editors
 - Stored in user's workspace
- Use can activate perspective from menu

•





Actions, Progress, Preferences

- Actions can be defined declaratively or programmatically
 - You can place action to menu from another plug-in
 - Action can react on selection change (even declaratively)
 - You can define shortcuts for actions
- Infrastructure for running background task
 - User can observe progress
 - User can stop task
- Preferences
 - You can define editable preferences (Preference Pages)
 - You can store UI parts' (plug-ins') properties into workspace programmatically (there is special folder for each plug-in)

Cooperation of plug-ins (1/3)



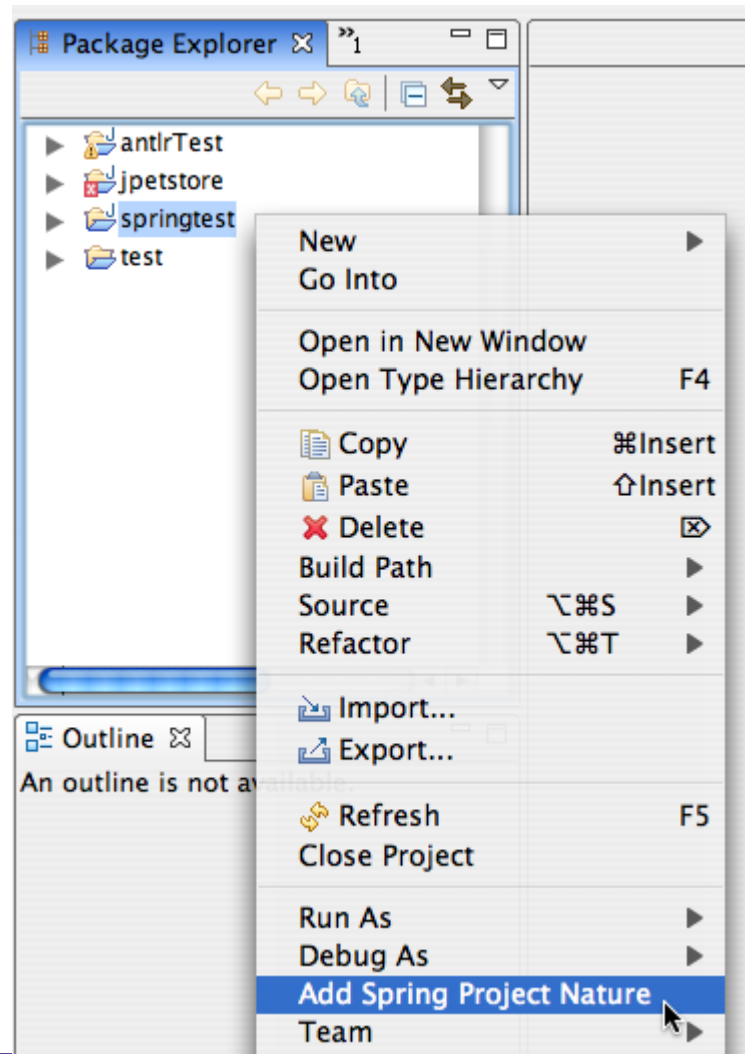
```
<requires>
  <import plugin="org.eclipse.core.runtime"/>
  <import plugin="org.eclipse.core.resources"/>
  <import plugin="org.eclipse.ui"/>
  <import plugin="org.eclipse.ui.ide"/>
  <import plugin="org.eclipse.jface.text"/>
  <import plugin="org.eclipse.ui"/>
  <import plugin="org.eclipse.ui.ide"/>
  <import plugin="org.eclipse.ui.views"/>
  <import plugin="org.eclipse.ui.workbench.texteditor"/>
  <import plugin="org.eclipse.ui.editors"/>
  <import plugin="org.eclipse.jdt.core"/>
  <import plugin="org.eclipse.jdt.ui"/>
  <import plugin="org.springframework.ide.eclipse.core"/>
</requires>
```

Cooperation of plug-ins (2/3)



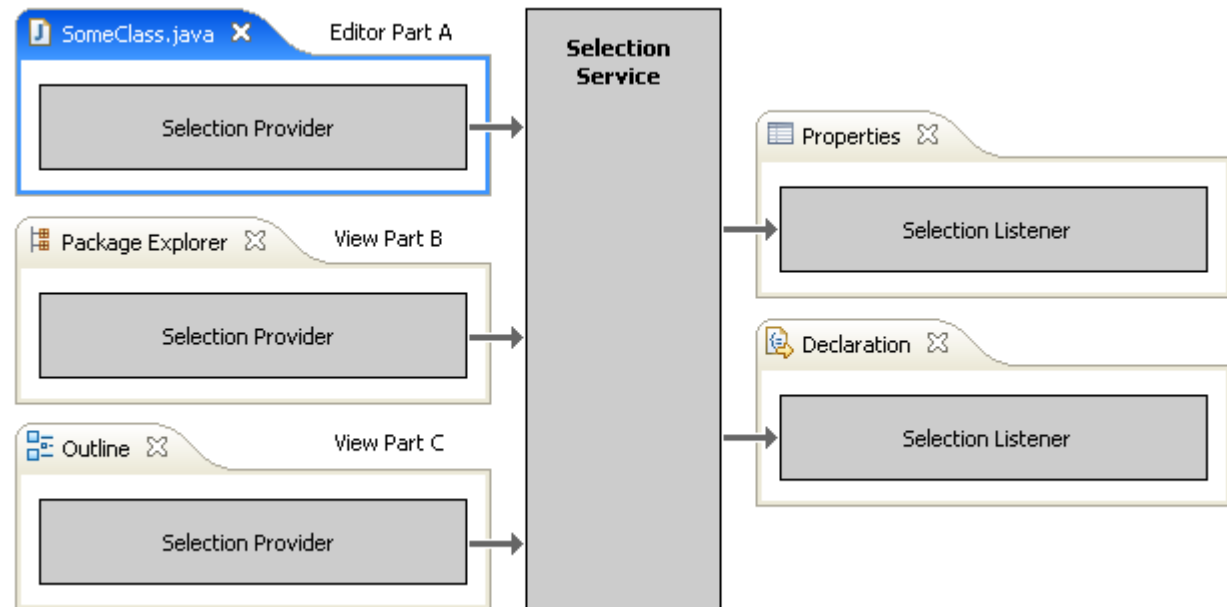
```
<extension
  point="org.eclipse.ui.popupMenus">
  <objectContribution adaptable="true"
    objectClass="org.eclipse.core.resources.IProject"
    id="org.springframework.ide.eclipse.ui.actions.nonSpringProject">
    <visibility>
      <and>
        <not>
          <objectState name="nature"
            value="org.springframework.ide.eclipse.core.springnature" />
        </not>
        <not>
          <objectState name="nature"
            value="org.springframework.ide.eclipse.beans.core.beansnature" />
        </not>
      </and>
    </visibility>
    <action label="%popupMenus.addNature.label"
      class="org.springframework.ide.eclipse.ui.internal.actions.AddRemoveNature"
      enablesFor="+>
      id="org.springframework.ide.eclipse.ui.actions.addNature">
    </action>
  </objectContribution>
```

Cooperation of plug-ins (3/3)



Connecting View Parts Together (1/2)

- Using the selection
 - Provider extends ISelectionInterface (Structured selection, Text selection), when provider is JFace UI component you are done
 - Listener must understand to emitted events, ignore unknown events





Connecting View Parts Together (2/2)

- Part Listeners
 - Listen on event specific to View Parts – close, open, hide, ...
- Direct communication
 - Callback methods
 - Register listener via extension point

Sample Application: V-Builder

The screenshot displays the V-Builder 3.0 application interface. The main window is titled "V-Builder 3.0" and contains several panes:

- Explorer [Project]:** Shows a tree view of design files including "GetDelivTime.design", "GetPhoneNum.design", "GetPizza.design", and "main.design". Below it, "Design Pages" lists "GetPhoneNum" and "PlayAddress".
- GetPhoneNum [GetPhoneNum]:** A flowchart diagram with nodes: "GetPhone" (start), "ConfirmPhone", "LookupAddress", and "PlayAddress". A "No" path loops back from "ConfirmPhone" to "GetPhone", and a "Yes" path leads from "ConfirmPhone" to "LookupAddress", which then leads to "PlayAddress".
- GSL Editor [pizza.gsl]:** A configuration panel for a "Recognition State" named "GetPhone". The description is "Get the caller's phone number." and "Use N-Best Scripting?" is unchecked.
- main [GetPizzaSize_help.wav]:** A window showing an audio waveform with playback controls (play, stop, pause, TTS) and a status bar indicating "Length: 5.778 Position: 0.0".
- Right Panel:** A list of pizza toppings: ground beef, beef, hamburger, jalapeno, jalapeeno, peppers, jalapenos, halapeenios, halapeenios, lotsa garlic, linguica, lingueesa, mushrooms, black, olives, onions, pastrami, pepperoni, pineapple, red onions, salami, sausage, shrimp, and pepperoni.

Cooperation of Modules

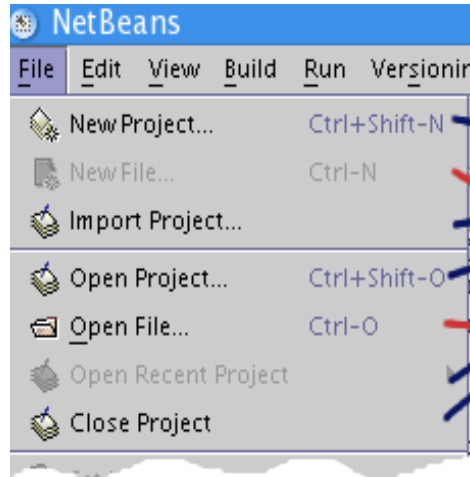
Composition of UI Elements

- Menu, toolbar elements
 - Get merged together by the NetBeans framework
- Windows Layout
- Registration solved by Layers

```
<folder name="Menu" >  
  <folder name="File" >  
    <file name="Open.instance" >  
      <attr instanceCreate=  
        "org.openide.actions.OpenAction" />  
    </file>  
  </folder>  
</folder>
```

Cooperation of Modules

Composition of Menu



```
folder name="Menu/File">
  <file name="NewProject.instance" />
  <file name="ImportProject.instance"
/>
  <file name="Separator.instance" />
  <file name="OpenProject.instance" />
  <file name="OpenRecent.instance" />
  <file name="CloseProject.instance" />
</folder>
```

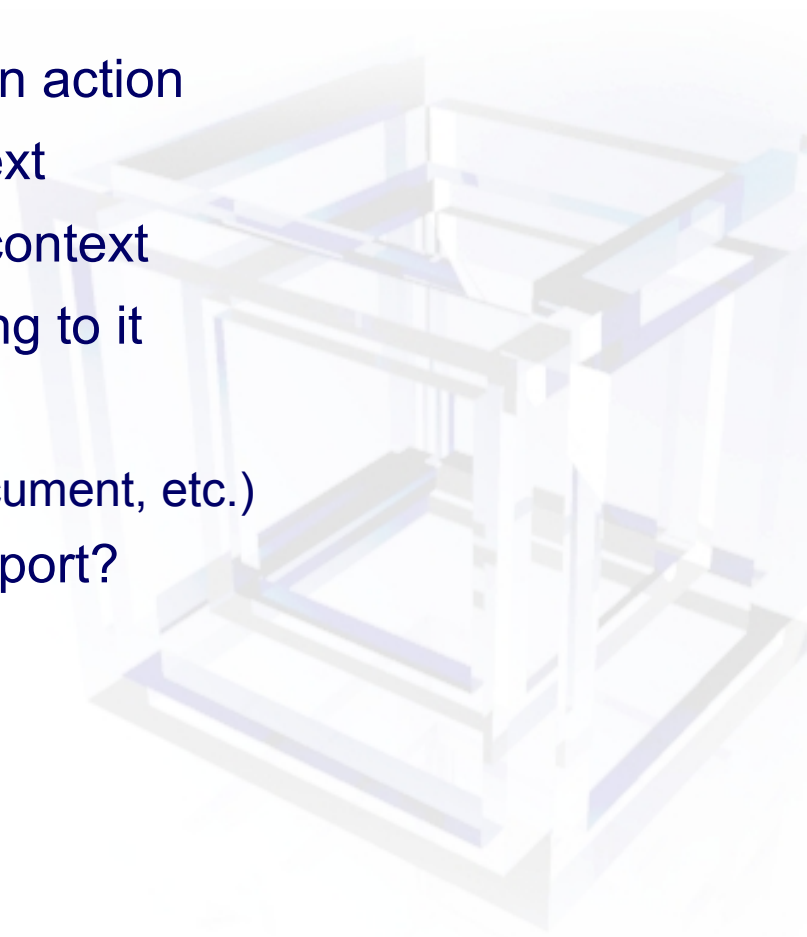
```
<folder name="Menu/File">
  <file name="NewFile.instance" />
  <file name="Open.instance" />
</folder>
```

```
<folder name="Menu/File">
  <attr name="NewProject.instance/NewFile.instance" boolvalue="true" />
  <attr name="NewFile.instance/ImportProject.instance" boolvalue="true"
/>
  <attr name="OpenProject.instance/Open.instance" />
  <attr name="Open.instance/OpenRecent.instance" />
</folder>
```

Cooperation of Modules

Exposing module state

- A module owns a Window, another an action
- Windows can have associated Context
- Selected window defines the global context
- UI Elements update its state according to it
- What is inside context?
 - Anything important (javax.swing.Document, etc.)
- What operations context need to support?
 - Changes of its content
 - Observability

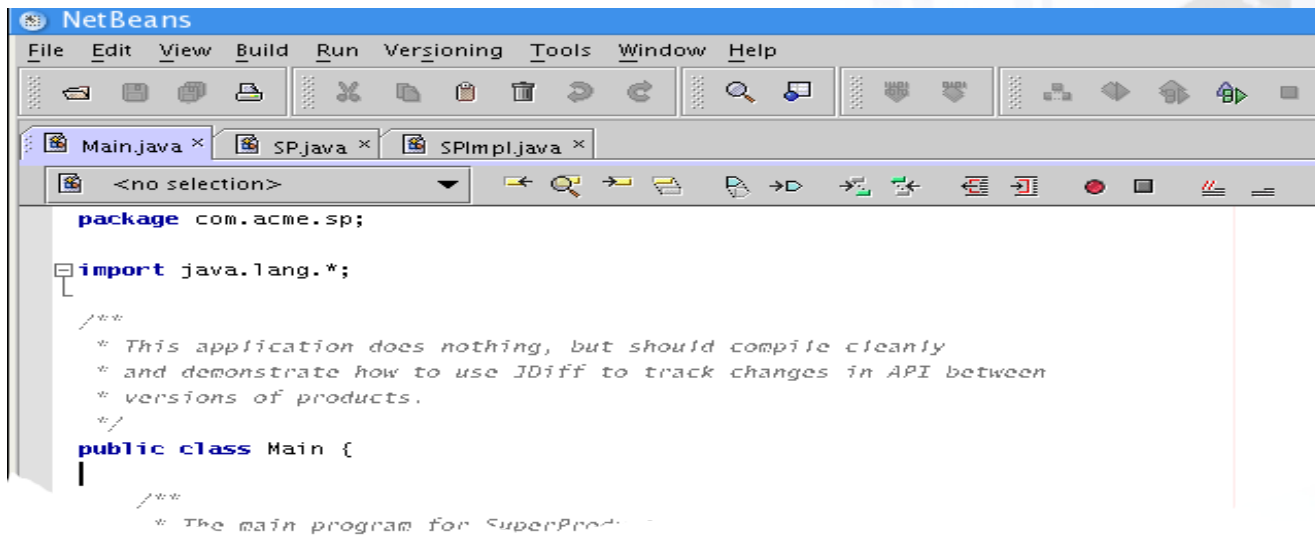


Cooperation of Modules

Exposing Window State

```
class MyWindow
extends org.openide.windows.TopComponent {
    private JEditorPane pane;

    public org.openide.util.Lookup getLookup () {
        return Lookups.singleton (pane.getDocument ())
    }
}
```



Cooperation of Modules

Querying and Listening to State

```
import org.openide.util.Utilities;

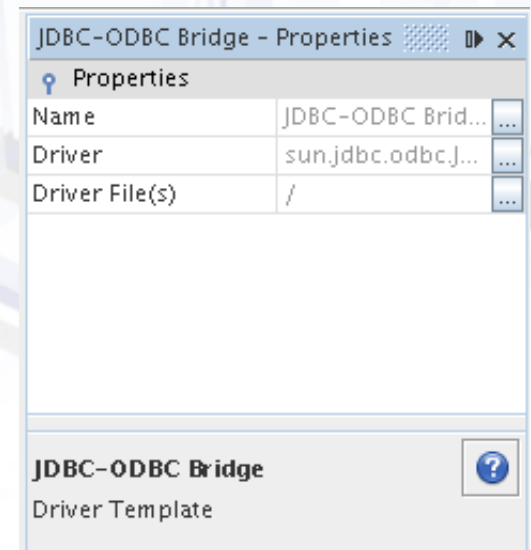
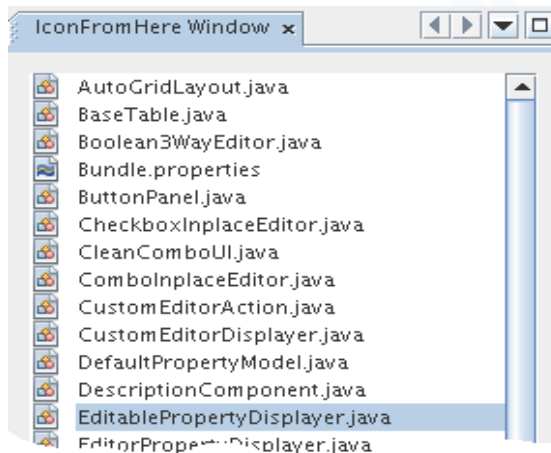
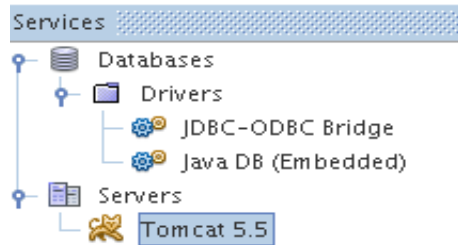
class MyInsertAction extends AbstractAction {
    public void actionPerformed (ActionEvent ev)
        Lookup lkp = Utilities.actionGlobalContext();
        Document doc = lkp.lookup (Document.class);
        if (doc != null) {
            doc.insertString ("Hello world!", null, 0);
        }
    }
}
```

In similar way one can listen to changes in Lookup using `LookupListener`.

Univesal Bean Tree Viewer

Nodes, Explorer and Property Sheet

```
class Node extends FeatureDescriptor {
    public Children getChildren();
    public Lookup getLookup();
    public void addNodeListener(NodeListener);
    public void removeNodeListener(NodeListener);
}
class Children {
    public Node[] getNodes();
}
```



Welcome screen



Welcome screen

NetBeans Gives You Free Stuff



The screenshot shows the NetBeans Welcome screen in a window titled "Welcome x". The interface is organized into several sections:

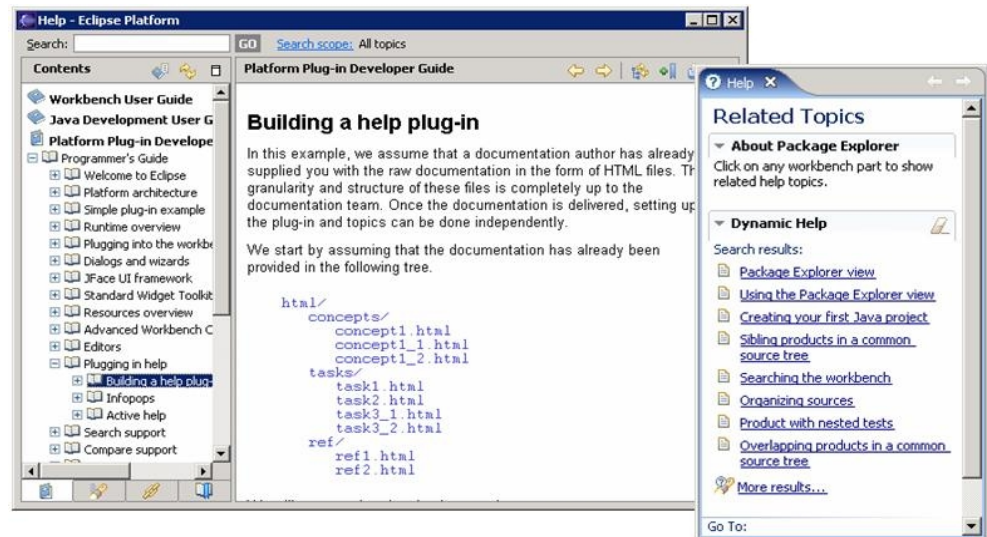
- GETTING STARTED**: A sidebar menu with "NetBeans Resources" and sub-items: "Docs and Support", "Mailing Lists", "NetBeans Platform", "Quick Start Guides", and "What's New".
- ARTICLES & NEWS**: A list of articles with titles and brief descriptions, including "NetBeans Google Toolbar Module Tutorial (2)", "Resolving Java ME Device Fragmentation Issues", and "Introduction to the Spring Framework in NetBeans IDE".
- SAMPLE PROJECTS**: A sidebar menu with "General", "Web", and "Enterprise" options.
- BLOGS**: A list of blog posts, including "Geertjan's Weblog: Tip of the Day Functionality for NetBeans Platform Applications".

At the bottom left is the Sun Microsystems logo, and at the bottom right is a checkbox labeled "SHOW ON STARTUP" which is checked.

Help System

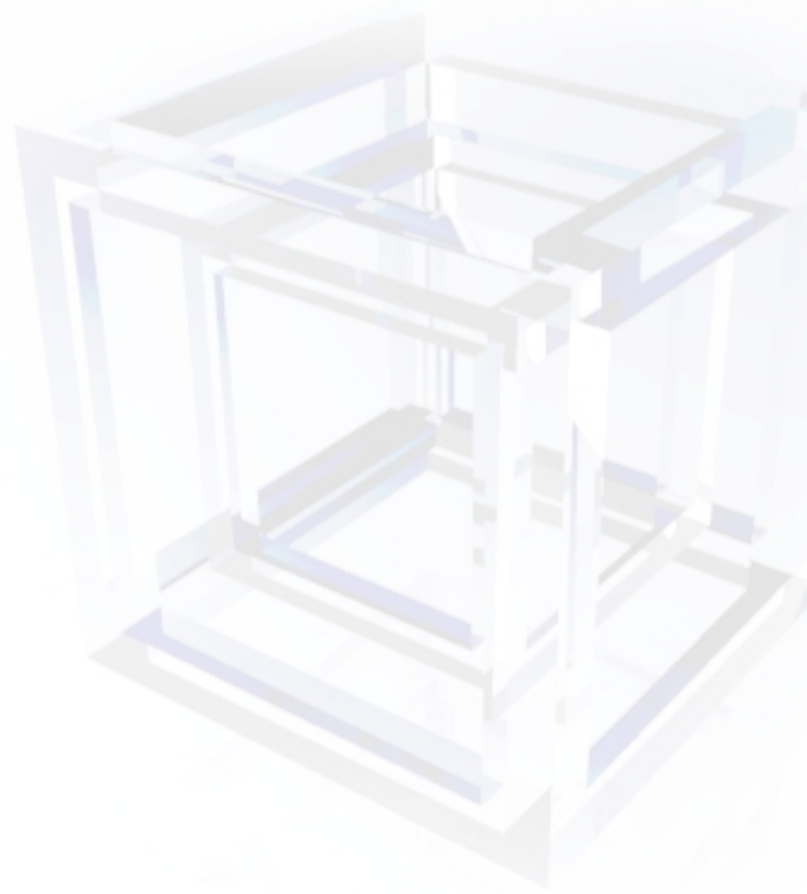


- Eclipse provides functionality for creating help plug-ins
- Eclipse provides capability for viewing and searching such help
- You use your help plug-in in RCP application
- Eclipse support linking SWT components and help items (i.e. Context Help)
 - `PlatformUI.getWorkbench().getHelpSystem().setHelp(composite, "net.bioclipse.plugins.bc_jmol.scriptInputField");`



Command Line API and Server Side NetBeans

Demo



Sample Eclipse RCP Application



Demo

Branding



- You can define:
 - Launcher name and icon for each operating system
 - Window icons
 - Splash screen
 - Progress bar placement
 - About dialog

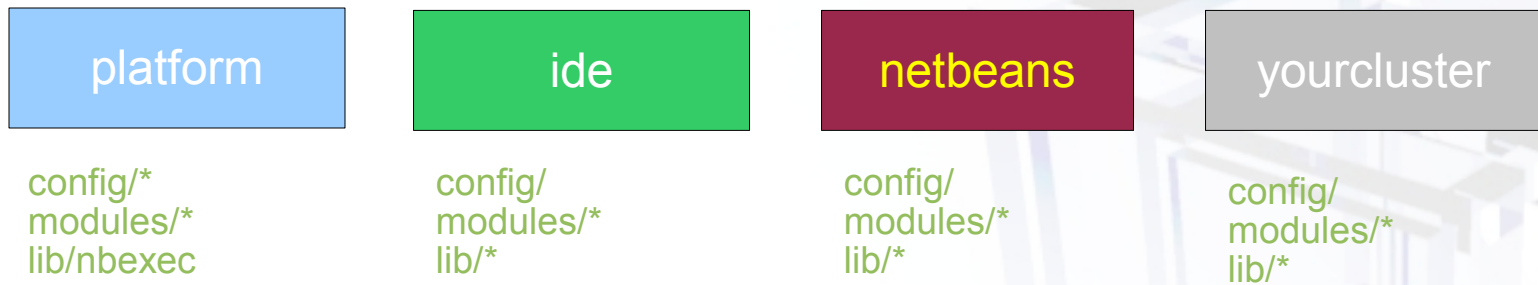


Deployment Scenarios

- What is stored on file system
 - Install location
 - Configuration location
 - Workspace
- Independent installs
- Shared Installs
 - Read only install
 - Configuration is stored in user home directory
- Shared Configurations
 - Configuration precreated on server
- Multiple Configurations
 - Sharing common plugins

Installation Structure

Clusters & Launcher



- Each product consists of a sets of clusters
- Has its own launcher
- **nbexec –cluster platform:ide:netbeans**
- Can provide branding of shared resources

Future

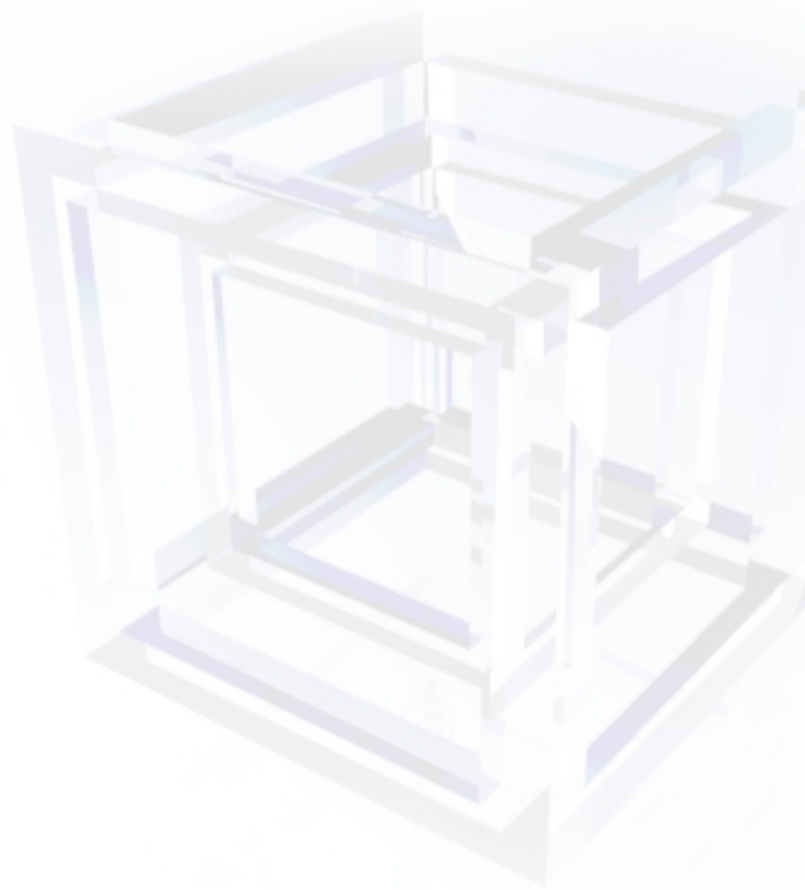


- Embedded RCP
- JFace
 - Databinding in Eclipse 3.3
 - Nebula project
- Many interesting Eclipse project – i.e you can reuse plugins
 - Business Intelligence and Reporting Tools
 - Eclipse Communication Framework
 - Eclipse Process Framework, Eclipse Java Workflow Tooling
 - Maya - installations
 - Open Healthcare Framework

-

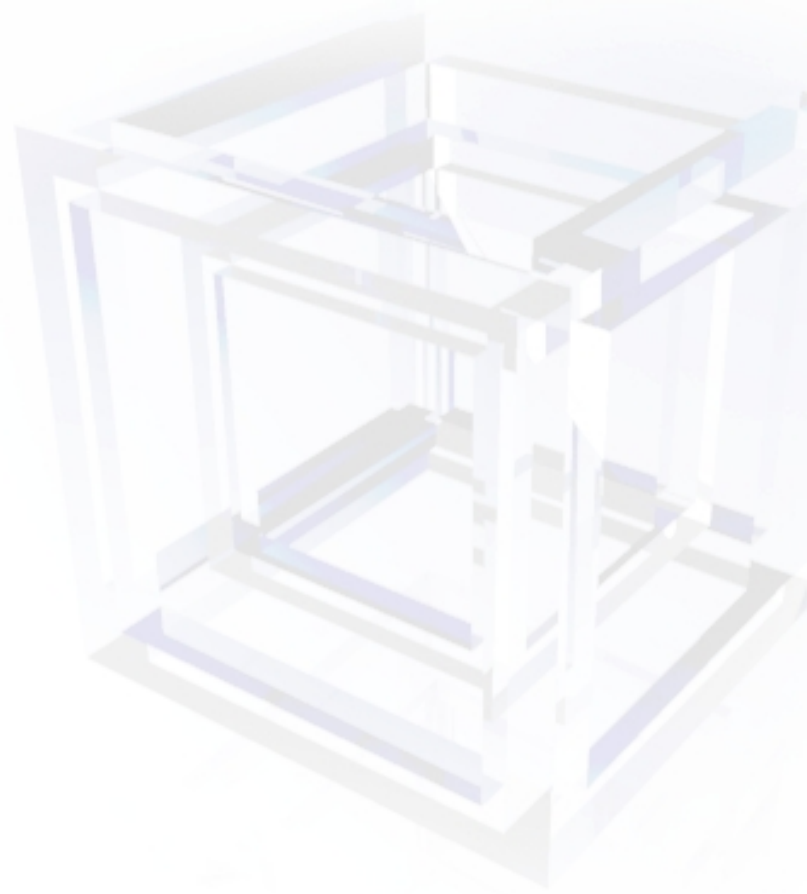
Future of Swing and NetBeans

- JSR 277
 - Java Module System
- JSR 295
 - Beans Binding
- JSR 296
 - Swing Application Framework
- JSR 269
 - Retouche
- Linux Packaging
 - ubuntu, rpm



Convert Swing Application to NetBeans

Demo



Branding Eclipse RCP Application

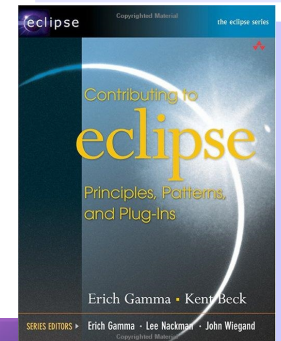
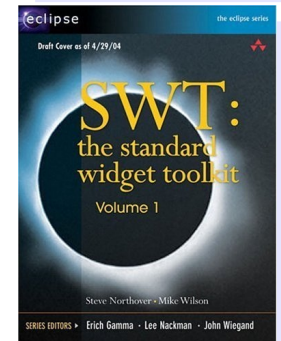
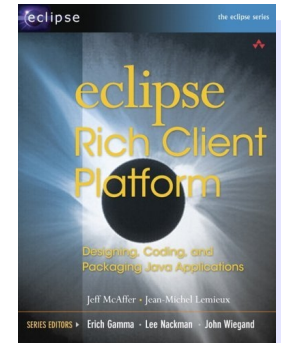


Demo

Recommended Reading

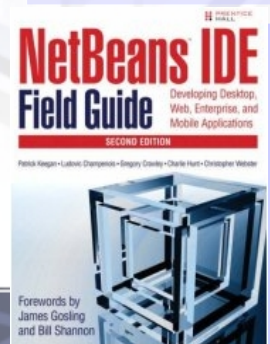
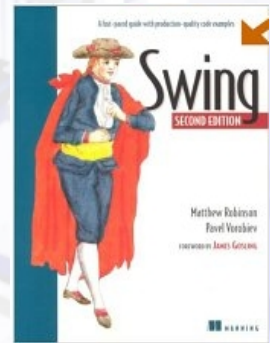
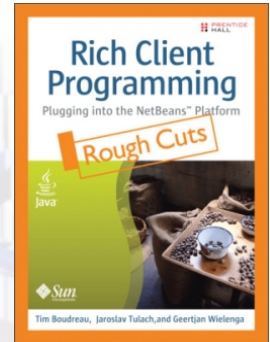


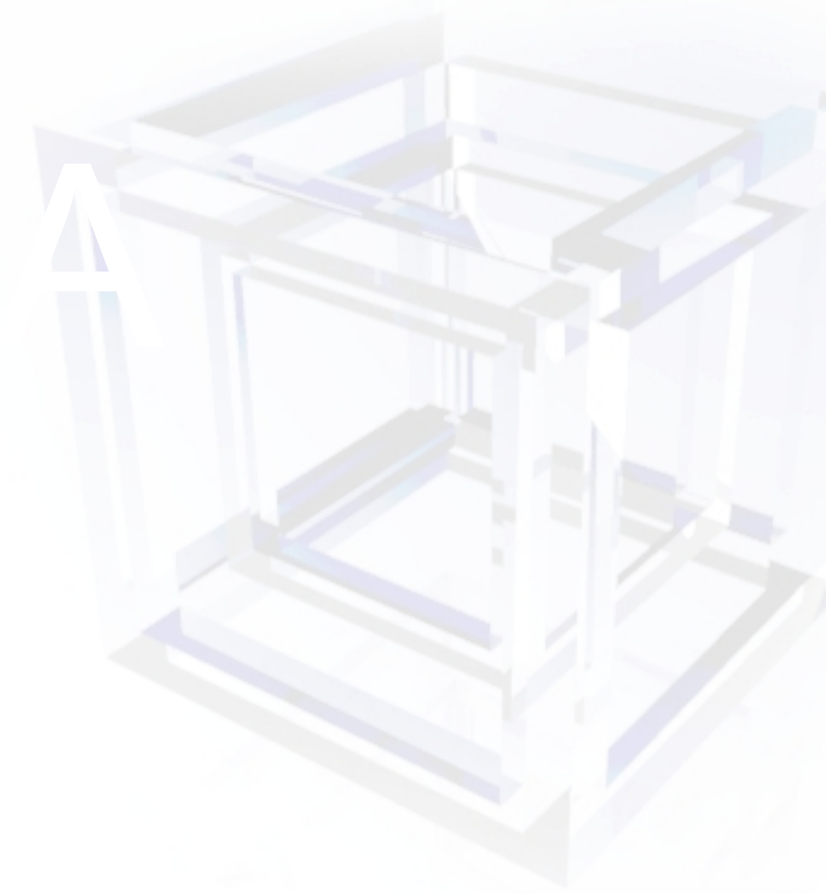
- Eclipse Rich Client Platform
 - By Jeff McAffer and Jean-Michel Lemieux
 - Addison-Wesley Professional
 - ISBN: 0321334612
- SWT : The Standard Widget Toolkit, Volume 1
 - By Steve Northover, Mike Wilson
 - Addison-Wesley Professional
 - ISBN: 0321256638
- Contributing to Eclipse: Principles, Patterns, and Plugins
 - By Erich Gamma, Kent Beck
 - Addison-Wesley Professional
 - ISBN: 0321205758



Recommended Reading

- Rich Client Programming, plug-in
 - By Tim Boudreau, Jaroslav Tulach, Geertjan Wielenga
 - Prentice Hall
 - ISBN: 0132354802
- Swing, Second Edition
 - By Matthew Robinson, Pavel Vorobiev
 - Manning Publications
 - ISBN: 193011088X
- NetBeansField Guide: Developing Desktop, Web, Enterprise, and Mobile Applications
 - By P. Keegan, L. Champenois, G. Crawley, Ch. Hunt, Ch. Webster
 - Prentice Hall
 - ISBN: 0321205758







Resources for Eclipse presentation

- Above mentioned books
- Presentations at <http://www.eclipse.org/resources/>
- Source code and documentation to Spring IDE - <http://springide.org>