How to Develop Eclipse Plug-ins

Randall M. Hauch

VP Development, Chief Metadata Architect

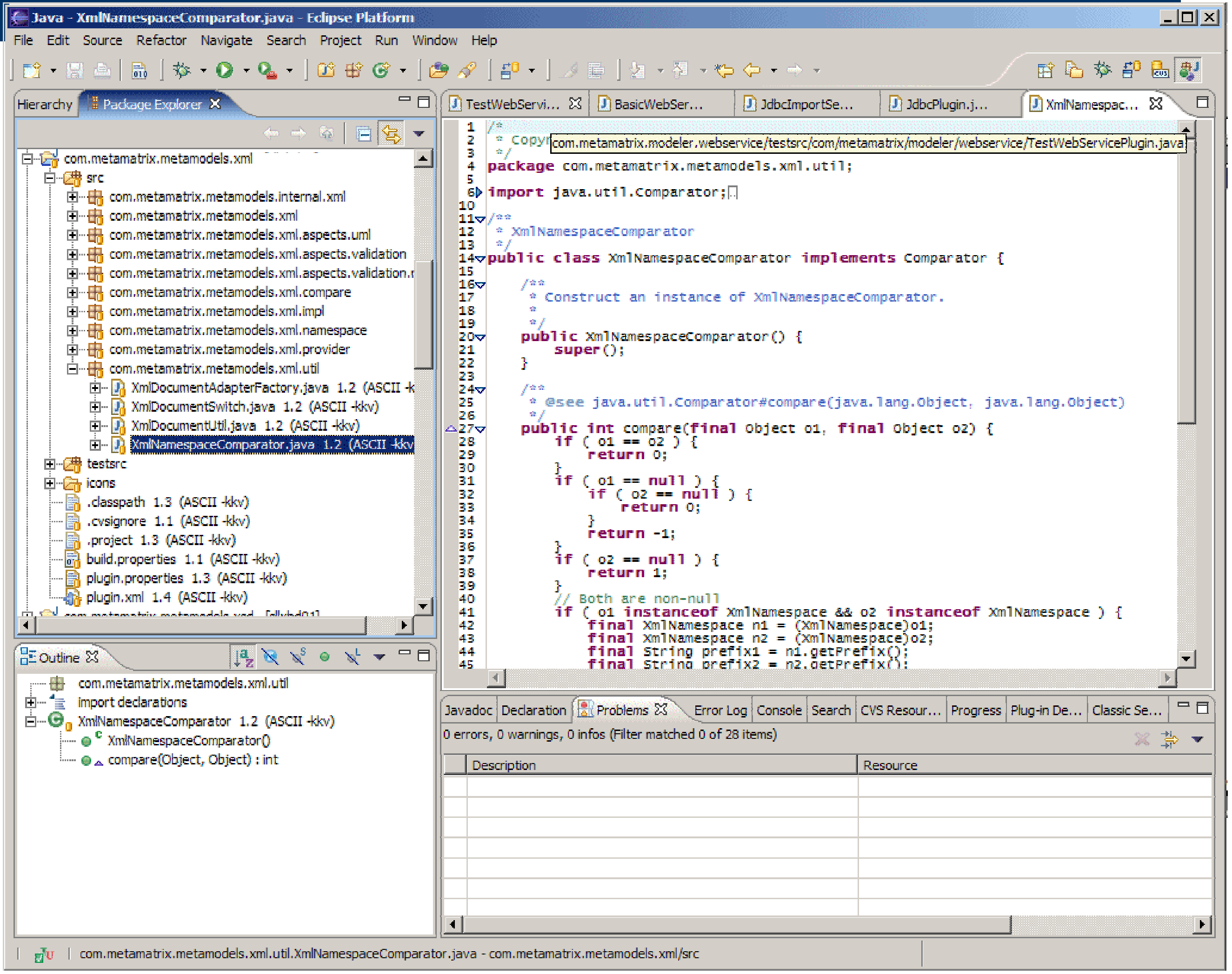
August 12, 2004

metamatrix®

Real Enterprise Information Integration

So Exactly What is Eclipse?

It’s an Outstanding and Free IDE …



Java - XmlNamespaceComparator.java - Eclipse Platform

File Edit Source Refactor Navigate Search Project Run Window Help

Hierarchy | Package Explorer

```
com.metamatrix.metamodels.xml
  src
    com.metamatrix.metamodels.internal.xml
    com.metamatrix.metamodels.xml
    com.metamatrix.metamodels.xml.aspects.uml
    com.metamatrix.metamodels.xml.aspects.validation
    com.metamatrix.metamodels.xml.aspects.validation.r
    com.metamatrix.metamodels.xml.compare
    com.metamatrix.metamodels.xml.impl
    com.metamatrix.metamodels.xml.namespace
    com.metamatrix.metamodels.xml.provider
    com.metamatrix.metamodels.xml.util
      XmlDocumentAdapterFactory.java  1.2  (ASCII -k
      XmlDocumentSwitch.java  1.2  (ASCII -kkv)
      XmlDocumentUtil.java  1.2  (ASCII -kkv)
      XmlNamespaceComparator.java  1.2  (ASCII -kkv
  testsrc
  icons
  .classpath  1.3  (ASCII -kkv)
  .cvsignore  1.1  (ASCII -kkv)
  .project  1.3  (ASCII -kkv)
  build.properties  1.1  (ASCII -kkv)
  plugin.properties  1.3  (ASCII -kkv)
  plugin.xml  1.4  (ASCII -kkv)
```

Outline

```
com.metamatrix.metamodels.xml.util
import declarations
XmlNamespaceComparator  1.2  (ASCII -kkv)
  XmlNamespaceComparator()
  compare(Object, Object) : int
```

TestWebServi… | BasicWebSer… | JdbcImportSe… | JdbcPlugin.j… | XmlNamespac…

com.metamatrix.modeler.webservice/testsrc/com/metamatrix/modeler/webservice/TestWebServicePlugin.java

```java
/*
 * Copyr
 */
package com.metamatrix.metamodels.xml.util;

import java.util.Comparator;

/**
 * XmlNamespaceComparator
 */
public class XmlNamespaceComparator implements Comparator {

    /**
     * Construct an instance of XmlNamespaceComparator.
     *
     */
    public XmlNamespaceComparator() {
        super();
    }

    /**
     * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
     */
    public int compare(final Object o1, final Object o2) {
        if ( o1 == o2 ) {
            return 0;
        }
        if ( o1 == null ) {
            if ( o2 == null ) {
                return 0;
            }
            return -1;
        }
        if ( o2 == null ) {
            return 1;
        }
        // Both are non-null
        if ( o1 instanceof XmlNamespace && o2 instanceof XmlNamespace ) {
            final XmlNamespace n1 = (XmlNamespace)o1;
            final XmlNamespace n2 = (XmlNamespace)o2;
            final String prefix1 = n1.getPrefix();
            final String prefix2 = n2.getPrefix();
```

Javadoc | Declaration | Problems | Error Log | Console | Search | CVS Resour… | Progress | Plug-in De… | Classic Se…

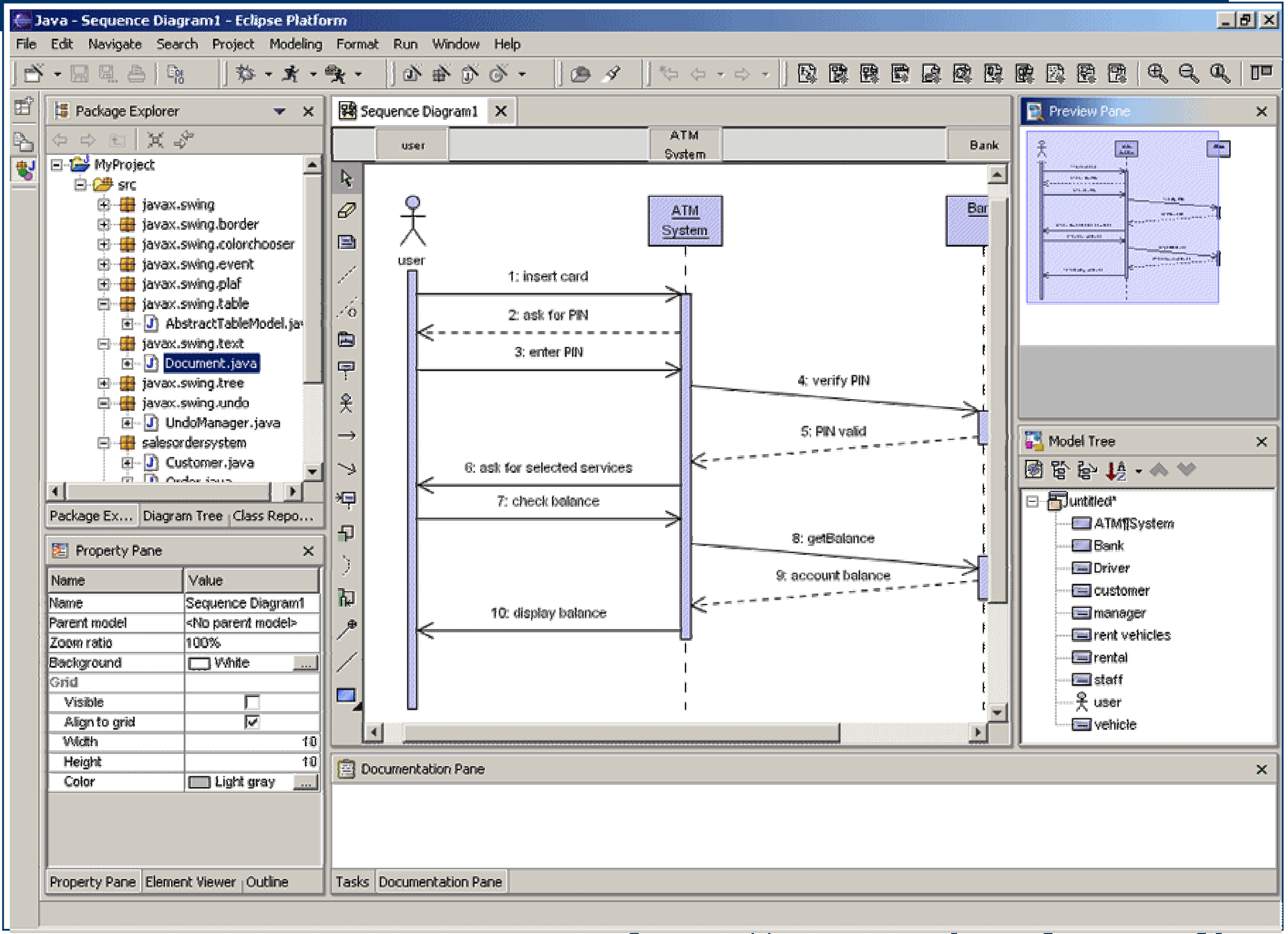
0 errors, 0 warnings, 0 infos (Filter matched 0 of 28 items)

Description	Resource

com.metamatrix.metamodels.xml.util.XmlNamespaceComparator.java - com.metamatrix.metamodels.xml/src

www.metamatrix.com

… Commercial Development Tools …

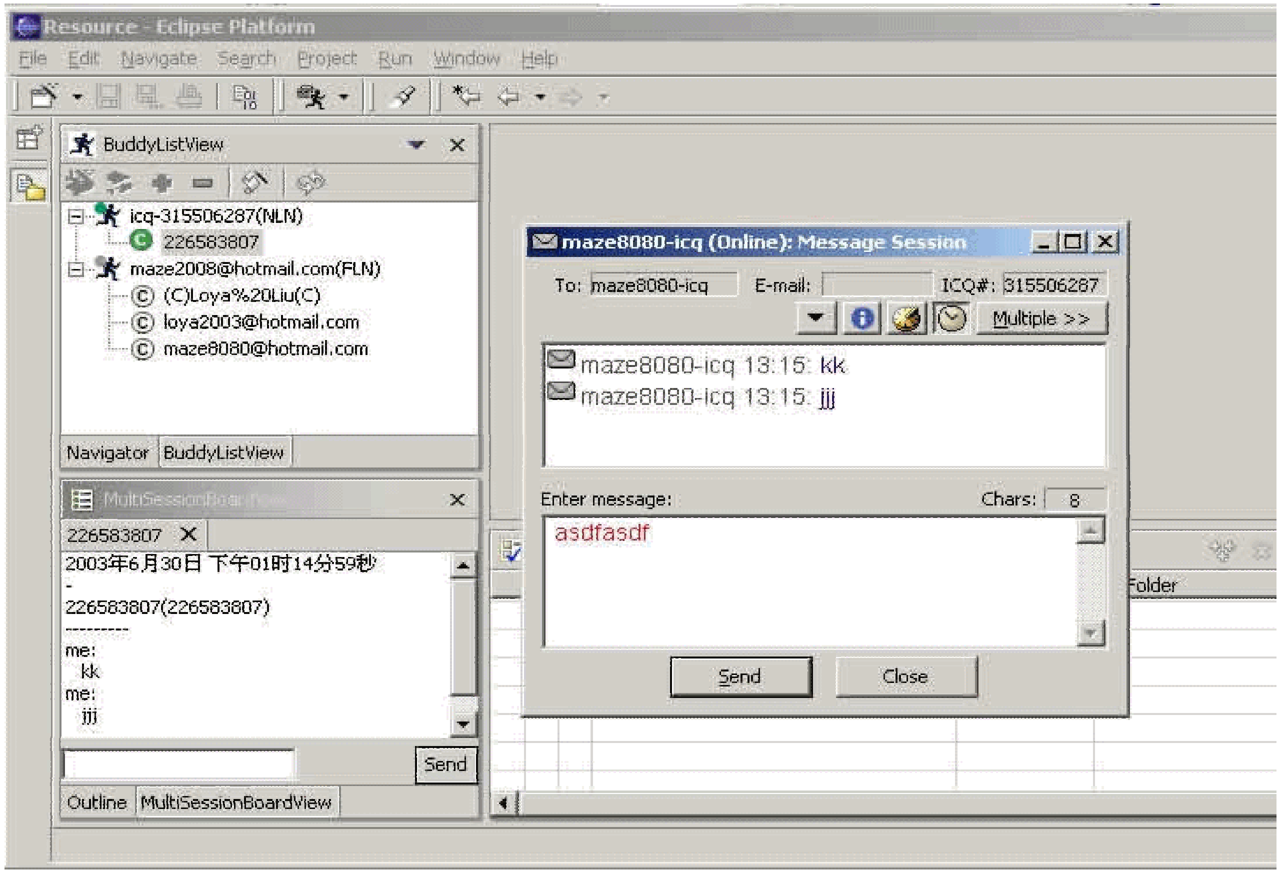


Property Pane

Name	Value
Name	Sequence Diagram1
Parent model	<No parent model>
Zoom ratio	100%
Background	White
Grid	
Visible	☐
Align to grid	✓
Width	10
Height	10
Color	Light gray

http://www.visual-paradigm.com

… Instant Messaging …


Resource - Eclipse Platform
File Edit Navigate Search Project Run Window Help
BuddyListView
Navigator BuddyListView
MultiSessionBoardView
Send
Outline MultiSessionBoardView
maze8080-icq (Online): Message Session
Multiple >>
Send
Close
Folder

icq-315506287(NLN)
226583807
maze2008@hotmail.com(FLN)
(C)Loya%20Liu(C)
loya2003@hotmail.com
maze8080@hotmail.com

226583807
2003年6月30日 下午01时14分59秒
-
226583807(226583807)
----
me:
kk
me:
jjj

To: maze8080-icq E-mail: ICQ#: 315506287

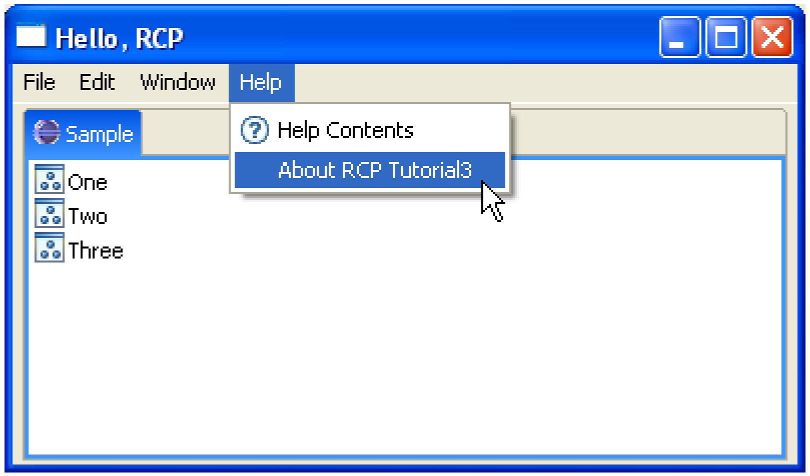
maze8080-icq 13:15: kk
maze8080-icq 13:15: jjj

Enter message: Chars: 8

asdfasdf

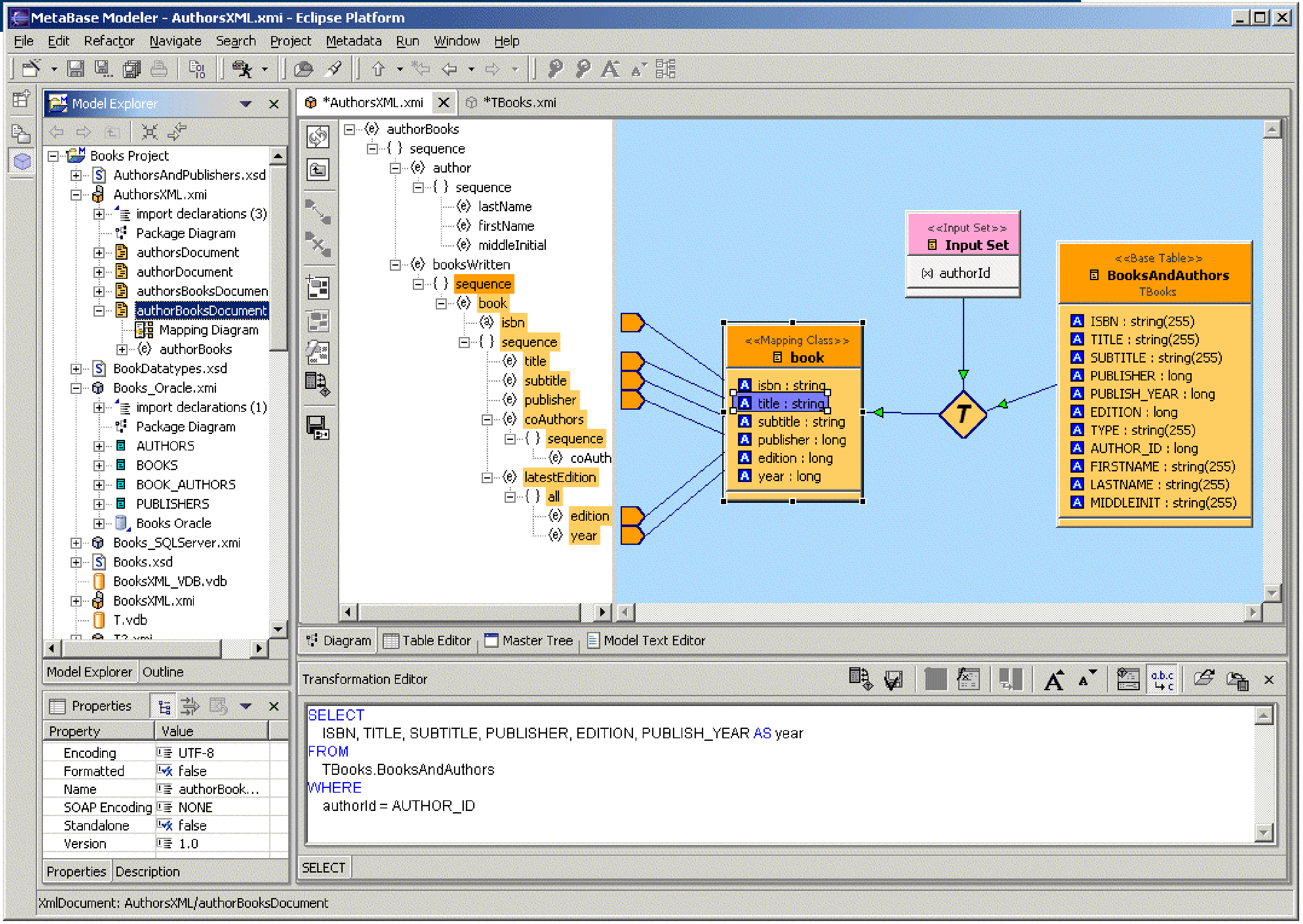
http://eimp.sourceforge.net

… Rich Clients …



http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/org.eclipse.ui.tutorials.rcp.part1/html/tutorial1.html

… Branded Applications …



Transformation Editor

```
SELECT
  ISBN, TITLE, SUBTITLE, PUBLISHER, EDITION, PUBLISH_YEAR AS year
FROM
  TBooks.BooksAndAuthors
WHERE
  authorId = AUTHOR_ID
```

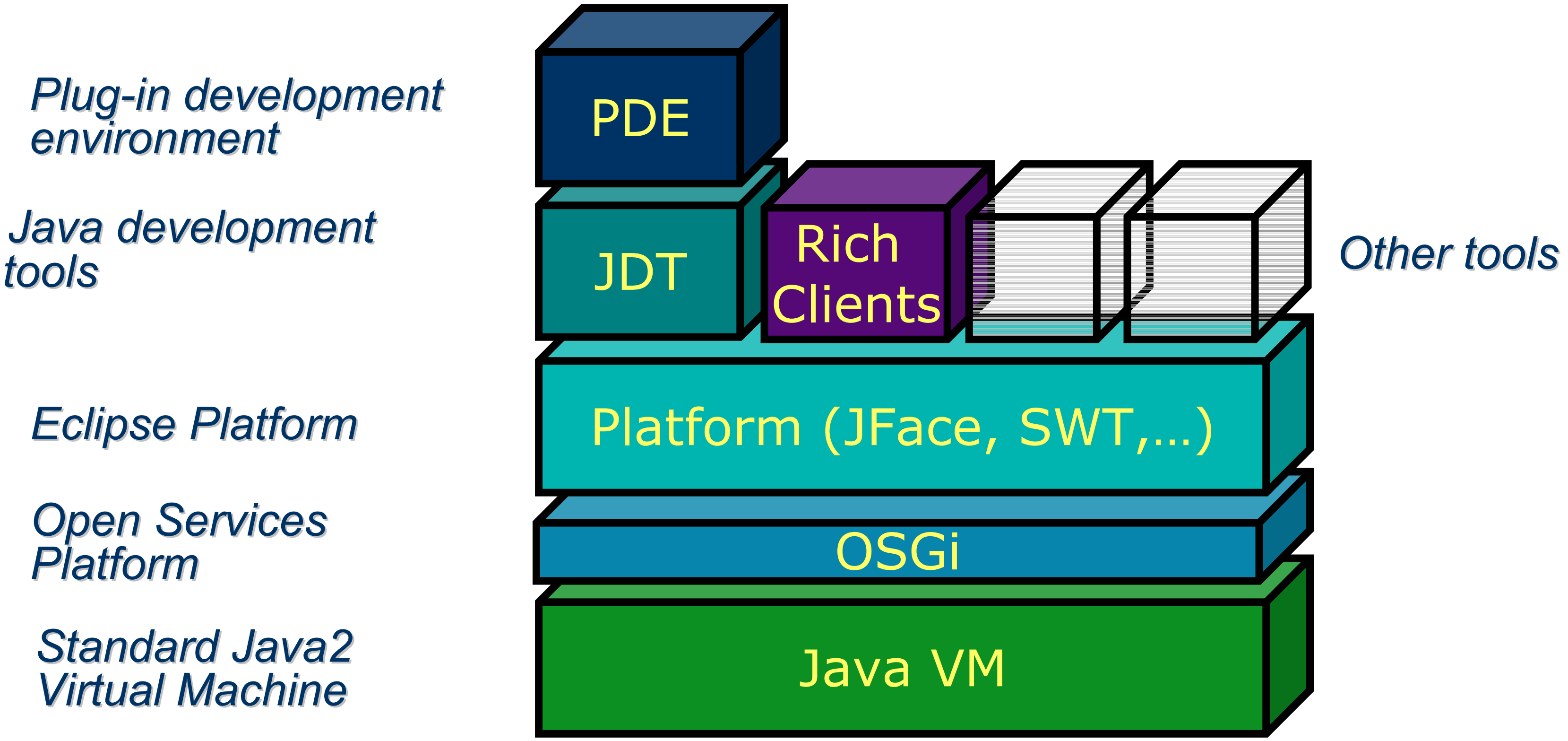
Properties

Property	Value
Encoding	UTF-8
Formatted	false
Name	authorBook…
SOAP Encoding	NONE
Standalone	false
Version	1.0

XmlDocument: AuthorsXML/authorBooksDocument

No, Really – What is Eclipse?

- Universal platform for integrating tools
- Platform for functionally-rich applications ("rich client")
- Architecutre that is open, extensible, and based on plug-ins



A Bit of Eclipse History

- In 1990's, tools weren't integrated & didn't work together
  - Many types of resources (e.g., JSPs, XML, HTML, Java …)
  - Didn't understand each others data
- Meanwhile, IBM recognized the success function had changed:
  - No longer was: "Who can build the best IDE?"
  - Was now: "How do you provide a truly integrated set of tools?"
- So Eclipse was born – a project to create a universal tool platform
  - Started in 1999
  - Open sourced in 2001
  - 2.0 shipped June 2002
  - 3.0 shipped June 2004

So? What makes Eclipse Different?

Eclipse is "***platform-centric***", not "tool-centric"

- Tool boundaries (as visible to user) disappear
- Platform has many standard and reusable interface components for performing common functionality
  - file management, repository integration, editors, view management, etc.
- Tools can be added at any time – and (with 3.0) they can be added even while running!
- Tool developers focus on their domain rather than "plumbing", and rely upon other tools built by experts in other domains

metamatrix®

How does Eclipse Work?

- When you want to build a new tool, you “teach” Eclipse about your tool rather than bolt on a monolithic “thing” on top of Eclipse
- You then write plug-ins that hook into plug-in points
- The result is users don’t see a new tool added to their environment, they instead see new capabilities that the platform is now able to perform
  - These new capabilities appear in places that make sense
  - Categories of capabilities can be turned off and on by user

Essential Eclipse Terminology

- **plug-in** – *[noun]* The smallest unit of functionality that describes itself and its capabilities using a manifest file; the unit of packaging and management.
- **extension point** – *[noun]* A well-defined point in the system that can be extended (implemented) by plug-ins.
- **extension** – *[noun]* A component in a plug-in that provides the contracted functionality defined by an extension point, usually by providing a class that implements the interface.
- **registry** – *[noun]* A list of installed and enabled plug-ins, the extension points they define, and the extensions they provide; in 3.0, this is OSGi.

Eclipse Plug-in Architecture

- Each plug-in
  - Contributes extensions to 1 or more extension points
  - Optionally declares new extension points
  - Depends on a set of other plug-ins
  - Contains Java code libraries and other files
  - May export Java-based APIs for downstream plug-ins
  - Typically lives in its own plug-in subdirectory
- Details spelled out in the **plug-in manifest**
  - Manifest declares contributions
  - Code implements contributions and provides API
  - plugin.xml file in root of plug-in subdirectory

Eclipse Plug-in Manifest (plugin.xml)

```
<plugin
    id = “com.example.tool"
    name = “Example Plug-in Tool"
    class = "com.example.tool.ToolPlugin">
  <requires>
    <import plugin = "org.eclipse.core.resources"/>
    <import plugin = "org.eclipse.ui"/>
  </requires>
  <runtime>
    <library name = “tool.jar"/>
  </runtime>
  <extension
    point = "org.eclipse.ui.preferencepages">
   <page id = "com.example.tool.preferences"
      icon = "icons/knob.gif"
      title = “Tool Knobs"
      class = "com.example.tool.ToolPreferenceWizard“/>
  </extension>
  <extension-point
    name = “Frob Providers“
    id = "com.example.tool.frobProvider"/>
</plugin>
```

Plug-in identification

Other plug-ins needed

Location of plug-in’s code

Declare contribution this plug-in makes

Declare new extension point open to contributions from this & other plug-ins

Using Eclipse to Develop Plug-ins

- The Plug-in Development Environment (PDE) is comprised of additional plug-ins (on top of Java Development Tools, or JDT):
  - ***Editors*** for plug-in related files
  - ***Views*** to find dependencies, extensions, extension points, etc.
  - ***Wizards*** to help create various files and associated projects
  - ***Builders*** to “compile” plug-in artifacts
  - ***Self-Hosting*** to run/debug the Eclipse platform
  - ***Tools*** to assist in packaging and deployment

Let’s make some plug-ins …

Part 1: Create a simple plug-in

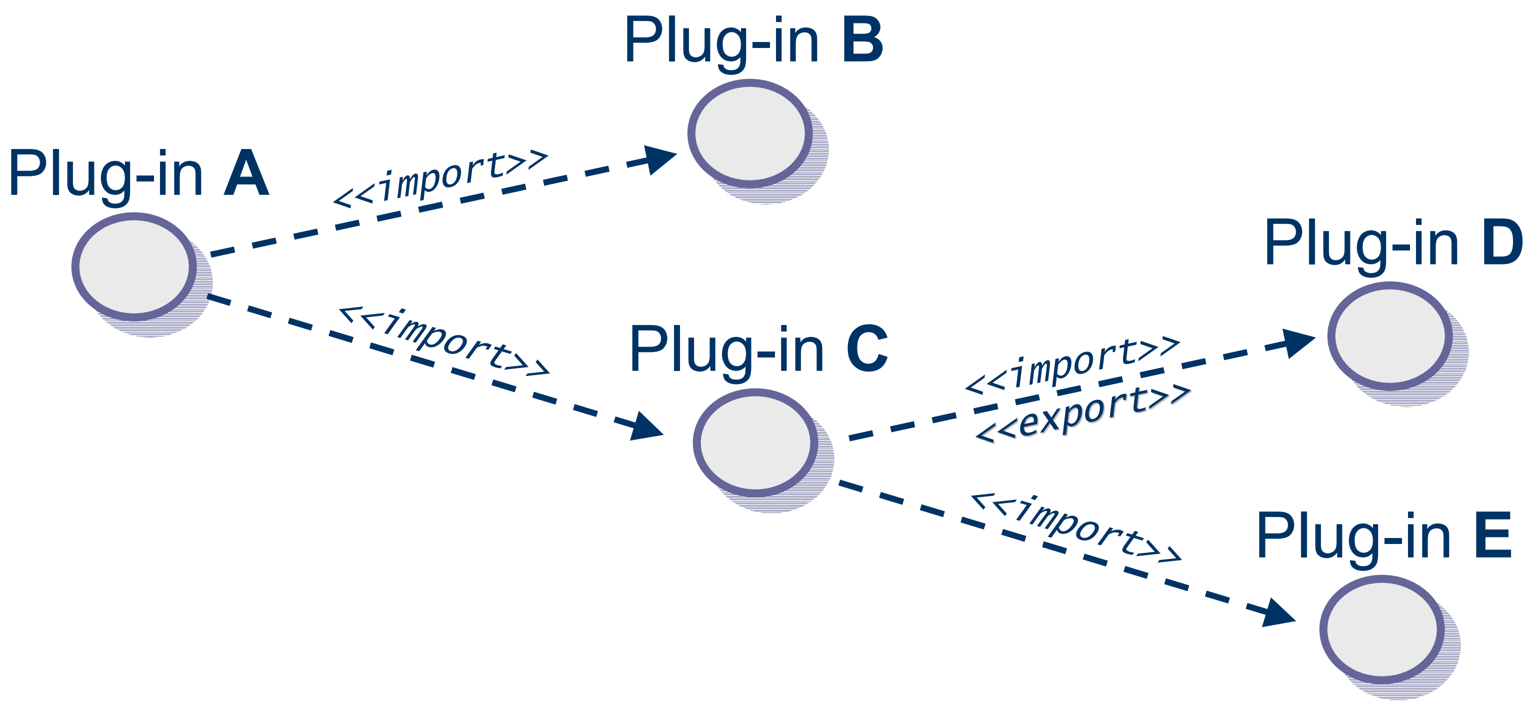
- Single action placed on toolbar and menu
- Second action (ordering)
- Actions on views
- Add a New File wizard
- Add a custom perspective
- Work with multiple plug-ins
  - Separate functionality into non-UI and UI plug-ins
  - Add dependencies (via imports)
  - Change view action to execute non-UI logic

Working with multiple plug-ins …

Eclipse Plug-in Activation

- Contributions processed without plug-in activation
  - Example: Menu constructed from manifest info for contributed items
- Plug-ins are activated *only as needed*
  - Example: Plug-in activated only when user selects its menu item
  - Scalable for large base of installed plug-ins
  - Helps avoid long start up times
- Each plug-in gets its own Java class loader
  - Loads its own resources
  - Delegates to required plug-ins for their resources
  - Restricts class visibility to exported APIs
- Each plug-in can have a single Java class that will be called by the platform to signal lifecycle changes (e.g., startup, shutdown)

Plug-in Dependencies



- If a plug-in (C) ***exports*** a plug-in that it imports (D), then the exported plug-in (D) is available to any plug-in (A) that imports the first (C)

So, what's the Catch?

- A plug-ins doesn't always explicitly "know" about other plug-ins at compile-time
- A plug-in can *dynamically* discover extensions of an extension point it knows about and knows how to use
  - The plug-in must import the plug-in with the extension point
  - The plug-in does *not* have to import plug-ins with the extensions

This decoupling is one of the architectural characteristics that help to make possible real tool integration, even when the tools don't know about each other!

- But this is when the classloader nightmares can happen!
  - Error messages usually don't describe the real problem, since usually fail to load a class that needs the missing class

Part 2: Multiple plug-ins

- Separate functionality into non-UI and UI plug-ins
- Add dependencies (via imports)
- Change view action to execute non-UI logic

Building and Deploying …

Deploying Eclipse Plug-ins

- A **feature** groups plug-ins into installable chunks
  - Feature manifest references the plug-ins
- Features have version identifiers
  - major.minor.service
  - Multiple feature versions can be installed at same time
- Features are downloadable and installable via web
  - Use the Platform’s Update Manager
  - Find and install new features (and their plug-ins)
  - Find and install updates to already existing features

Part 3: Deploying and Installing

- Create deployable artifacts
  - Feature plug-in(s)
  - Update site plug-in
  - Export plug-in projects
  - Export feature project(s)
  - Copy site.xml to location

- Install via Update Manager
  - Start new Eclipse
  - Find and install plug-ins (will prompt to restart)

Rich Client Platform …

Rich Client Platform

- Enables use of the Platform to create feature-rich client applications
  - Complete freedom in what the UI looks like – you do _not_ have to start with the “basic IDE”
  - Still able (but not required) to use the parts of the Platform you need – e.g., SWT, JFace, etc.
- Eclipse 3.0 uses OSGi
  - Plug-ins can be downloaded before they are run
  - Plug-ins can be selectively loaded and unloaded
  - Plug-ins can even be added after startup without having to restart

Part 4: Rich Client Platform

- The “smallest possible” Eclipse application …

Other Topics …

Internationalization

- Eclipse has integrated support for internationalization
  - Patterns for managing resource bundles (for plugin.xml and for code)
  - Wizards to externalize strings
  - Builder options to identify non-externalized strings
  - Can create plug-in fragments that add new locale-specific bundles to an installation

metamatrix®

Branding

- Eclipse licensing allows for unlimited distribution
  - See http://www.eclipse.org/legal for all the details
- Built-in capabilities to create branded products with custom
  - Splash screen
  - About dialog
  - Program executable

Questions?

Resources

1. http://www.eclipse.org
2. http://www.eclipseplugincentral.org
3. Shavor, et al., “The Java Developer’s Guide to Eclipse”, Addison-Wesley, NY, 2003

References

1. “A Different Shade of Blue”, Dave Thompson, IBM, presentation given at EclipseCon 2004
2. http://www.eclipse.org/eclipse/presentation/eclipse-slides.html
3. Shavor, et al., “The Java Developer’s Guide to Eclipse”, Addison-Wesley, NY, 2003