

# The Eclipse Way

Szymon Brandys   Pawel Pogorzelski   Tomasz Zarna

Platform Workspace Team  
IBM Poland

2009-06-27

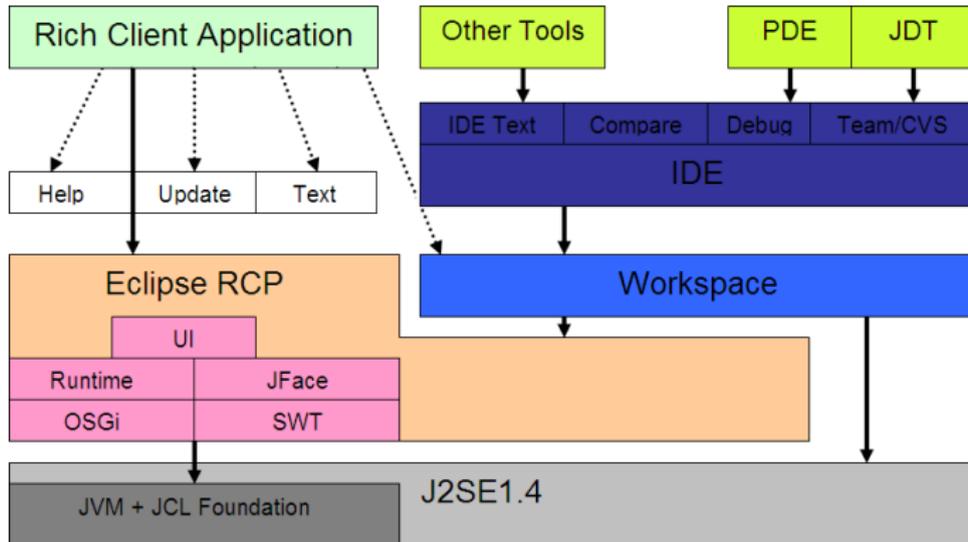
# Agenda

- 1 What Is Eclipse
- 2 Growth Path
- 3 How We Are Organized

## How We Got Here

- Java strategic technology for IBM
- Need to compete with MS VS and other Java IDE
- Created in 1998 by IBM/OTI teams responsible for VisualAge product family
- In 2001 given opened to open source to increase exposure and accelerate adoption
- In 2004 Eclipse Foundation was created
- Eclipse already well regarded tooling platform
- In mid 2004 Eclipse 3.0 ships, now based on OSGi
- Eclipse becomes more and more an RCP platform
- Thousands of Eclipse based products on the market, from ST to fully loaded IDEs

# Eclipse Architecture



# Eclipse 3.5

The screenshot displays the Eclipse IDE interface. On the left, the Package Explorer shows the project structure for 'org.eclipse.core.net'. The 'src' folder is expanded, showing the 'org.eclipse.core.internal.net' package with various Java files. 'ProxyManager.java' is selected and highlighted. The main editor window shows the source code of 'ProxyManager.java'. The code includes a package declaration, an import for 'java.net.Authenticator', and a class declaration 'public class ProxyManager implements IProxyService, IPreferenceListener'. It also shows several private static final String constants and a private static final IProxyService instance. A large, semi-transparent Eclipse logo is overlaid on the right side of the code editor. At the bottom right of the code editor, there is a small copyright notice: '(c) Copyright Eclipse contributors and others, 2000, 2009. All related trademarks and logos are trademarks or registered trademarks in the U.S., other countries, or both. Eclipse is a trademark'.

```
Java - org.eclipse.core.net/src/org/eclipse/core/internal/net/ProxyManager.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer  Ju JUnit  ProxyManager.java
org.eclipse.compare [dev.eclipse.org]
org.eclipse.compare.core [dev.eclipse.org]
org.eclipse.compare.examples
org.eclipse.compare.examples.xml
org.eclipse.compare.tests
org.eclipse.compare.win32
org.eclipse.core.net [dev.eclipse.org]
src
  org.eclipse.core.internal.net
    AbstractProxyProvider.java 1.
    Activator.java 1.6
    Messages.java 1.1
    Policy.java 1.2
    PreferenceInitializer.java 1.3
    PreferenceModifyListener.java
    ProxyChangeEvent.java 1.1
    ProxyData.java 1.9
    ProxyManager.java 1.24
    ProxyManager 1.24
    ProxySelector.java 1.6
    ProxyType.java 1.17
    StringMatcher.java 1.1
    StringUtil.java 1.1
    WindowsProxyProvider.java 1
    messages.properties 1.1
  org.eclipse.core.internal.net.proxy
  org.eclipse.core.net.proxy

ProxyManager.java
 * Copyright (c) 2007, 2009 IBM Corporation and others.
 package org.eclipse.core.internal.net;

 import java.net.Authenticator;

 public class ProxyManager implements IProxyService, IPreference

     private static final String PREF_HAS_MIGRATED = "org.eclip

 /**
  * Preference constants used by Update to record the HTTP
  */
 private static String
 private static String
 private static String

 private static final
 private static final
 private static final

 private static IProxy

 private AbstractProxy!

 ListenerList listeners;
 private String[] nonPr
 private final ProxyTyp
 new ProxyType
 new ProxyType
 new ProxyType
```

# IBM Rational Software Architect 7.0.0

The screenshot displays the IBM Rational Software Architect 7.0.0 IDE. The title bar reads "Resource - ProxyManager.java - Rational Software Architect". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Data, Run, Window, and Help. The Navigator on the left shows a project structure for "org.eclipse.core.net [dev.eclipse.org]". The main editor window shows the source code for "ProxyManager.java".

```
 * Copyright (c) 2007, 2009 IBM Corporation and others.
 package org.eclipse.core.internal.net;

 import java.net.Authenticator;

 public class ProxyManager implements IProxyService, IPreference

     private static final String PREF_HAS_MIGRATED = "org.eclipse

 /**
  * Preference constants used by Update to record the HTTP
  */
 private static String
 private static String
 private static String

 private static final §
 private static final §
 private static final §

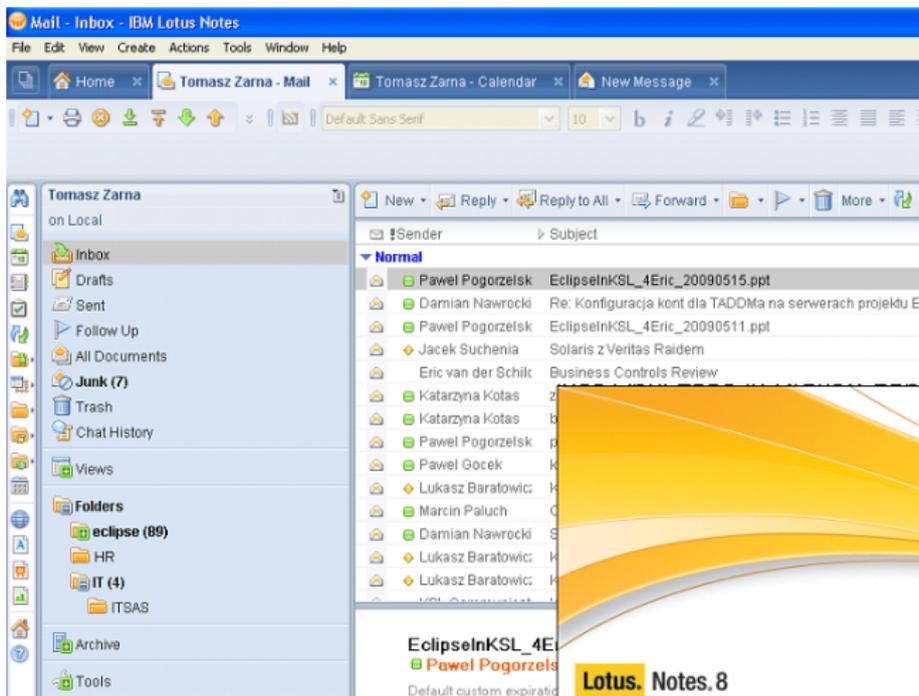
 private static IProxy§

 private AbstractProxy§

 ListenerList listeners
 private String[] nonF
 private final ProxyTyp
 new ProxyType
 new ProxyType
 new ProxyType
```

IBM Rational Software Delivery Pla  
Version 7.0

# IBM Lotus Notes 8



## Tivoli® Common Agent Services

```
berens:/opt/tivoli/ep/runtime/agent # ./agentcli.sh deployer list bundles state
Active System Bundle
Active initial@reference:file:plugins/org.eclipse.core.runtime_3.1.2.jar/
Active initial@reference:file:plugins/org.eclipse.update.configurator_3.1.0.jar/
Active initial@reference:file:plugins/com.ibm.pvc.wct.platform.autostart_6.1.0.0-20060201.jar/
Active update@plugins/org.eclipse.core.runtime.compatibility_3.1.0.jar
Active update@plugins/org.eclipse.osgi.services_3.1.2.jar
Active update@plugins/org.eclipse.osgi.util_3.1.1.jar
Installed update@plugins/org.eclipse.update.core.win32_3.1.0.jar
Resolved update@plugins/org.eclipse.update.core_3.1.1.jar
Active update@././agent/subagents/eclipse/plugins/CDSAxis.jar
Active update@././agent/subagents/eclipse/plugins/CDSClientAPIBundle.jar
Active update@././agent/subagents/eclipse/plugins/CDSDepotServer.jar
Active update@././agent/subagents/eclipse/plugins/COP-CommonAgent-TPM.jar
Active update@././agent/subagents/eclipse/plugins/CitScannerAgent_linux.jar
Active update@././agent/subagents/eclipse/plugins/EventAdmin.jar
Active update@././agent/subagents/eclipse/plugins/SCMCollectorAgent_linux.jar
Active update@././agent/subagents/eclipse/plugins/TPMAgentExt.jar
```

## Keeping It Big

How can you build something that can last 10 years and be:

- Industry leading
- Extendable
- Constantly evolving

Well, you need to have those:

- Modularity
- Scalability
- Stable APIs

# JVM Classloading

## Application wide classpath

```
java -classpath  
./a-1.5.0.jar:./b.jar:./c.jar:./a-2.1.0.jar./bin:  
com.vendor.App
```

- Not possible to use the same library in two versions
- Classnames conflicts
- All entries has to be searched which results in performance hit
- No need to declare dependencies explicitly

# OSGi

Classes and charts aren't enough, you need components. OSGi provide those in form of bundles:

- Explicit dependencies management
  - `Import-Package: org.osgi.framework; version=1.2`
  - `Export-Package: org.osgi.service.cm; version="1.2.1"`
- No sea of classes, no exhibitionism

Application wide classpath

**Modularity** that OSGi gives enables evolution.

## Declarative extensions

**Extension points** are the places where you expect functionality to be extended. **Extensions** are features that plug-in into extension points.

- Simple and powerful
- Don't load code until it is needed
- Explicit points where you can plug-in

### Lazy loading

Lazy loading given by Extension Registry gives **scalability**.

## Stable APIs

Stable APIs are critical to sustain growth. Clients can add features instead of updating to new API. So, it has to be:

- Consistent and wise
- Any decision made today will impact where you can go tomorrow

API compatibility is a huge commitment so we take a defensive approach:

- Don't add until there is at least one client
- Exhibit less rather than more
- Expose more if needed

## API Tension

API needs iteration and clients to work. But we need to have stable APIs for widespread adoption. So, we:

- Develop API and client at the same time
- Don't commit API before it's time
- API changes within release to accomodate new requirements and experience

It also gives us early feedback on API violation. Just because it works doesn't mean it's API compliant.

## API Layers

What if we want add new, more feature rich mechanizm?

- 1 Add a new functionality
- 2 An API layer that maps the old API to new implementation
- 3 Remove the old implementation
- 4 Deprecate the old API
- 5 After a few years we might drop it

### Binary compatibility

We tend to have more than stable API. Binary compatibility is what matters since users will not rebuild a plug-in.

## API Tools

API Baseline defines the state you want to compare your development against. Tools check:

- Usage problems
- Binary compatibility
- Bundle version numbers
- Maintenance tag @since

Other tags:

- @noimplement
- @noinstantiate
- @noextend

Known problems can be marked appropriately and filtered.

### Integration

Tools make API violations are perceived as natural as language constraints.

# Planning

- Community input
- Discuss propositions on bug reports
- Committed, proposed items
- We drop items to maintain schedule

## Continuous Integration

- Releases - e.g. R3.4; stable, tested, lack the latest features
- Stable - e.g. 3.5RC4; latest features, valuable and timely feedback
- Integration Builds - e.g. I20090611-1540, run weekly
- Nightly Builds - e.g. N20090426-1232; often major problems, useful to Eclipse Project developers

Always beta

We work on nightly builds so we try to keep them running.

# Milestones

- There is 7 milestones, each takes 6 weeks
- Shipping is hard, that's why we do it 7 times a release
- Customers can rebase more frequently
- Plan, develop, test, release, retrospective
- We play all the roles
- New and Noteworthy
  - Feed the community
  - Make people move to milestone builds
  - Fewer completed than more in progress

# The Convergence Process

- M6 - API freeze
- M7 - feature freeze
- RC1 - another commiter, PMC for API changes
- RC2 - two committers
- RC3 - two committers, component lead
- RC4 - two component leads, any lead can veto

## Committing Into RCs

Release Candidates time is when you never want to have a buildbrake.

# Community

- Initial investment
- Community grows and becomes self supporting, we don't have to grow
- Early feedback
- Open technical discussions, even more important than open bugs
- Transparency, good for distributed teams
- The village effect