inside eclipse

Erich Gamma
Eclipse Project Management Committee Member
IBM Distinguished Engineer
IBM Rational Software
what is eclipse?

- an IDE and more…
  - it’s a Java development environment (JDT)
  - it’s a general tools and integration platform
  - it’s a general application platform (RCP)
- an open source community
- an ecosystem to enable a total solution
  - including products by some major tool vendors
- a foundation to advance the eclipse platform
the full eclipse spectrum – eclipse has grown…

- **eclipse project**
  - Platform
  - JDT
  - PDE

- **eclipse tools project**
  - VE
  - UML2
  - Hyades
  - CDT
  - GEF
  - EMF

- **eclipse web tools project**
  - BIRT (Business Intelligence Report Tools)
  - TPTP (Test Performance Tools Platform)
  - STP (SOA Tools Platform)

- **eclipse technology project**
  - Mylar
  - AJDT
  - AspectJ
  - Equinox
  - Pollinate (proposal)

⇒ **callisto simultaneous release - ten major eclipse projects release at the same time at the end of June 2006**
outline

- more than a Java IDE
- more than a tools platform (RCP)
eclipse perception changes

3.2 “eclipse is an even better application and tools platform and Java IDE”

3.0/3.1 “eclipse is a general application platform”

2.0 “eclipse is a general tooling platform”

1.0 “eclipse is a Java IDE”

broader application improves the platform
built to last

- deliver on time, every time
  - decisions in this release impact what we can do next release
  - must preserve architectural integrity
how buildings last

• Stewart Brand: how buildings learn – what happens after they’re built
  stuff: furniture
  services: electrical, plumbing (7-15y)
  structure: foundation, load bearing walls (30-300y)
  site: geographical setting (forever)

• layers:
  • evolve at different rates during the life of a building
  • shear against each other as they change at different rates
  • an adaptive building must allow slippage
  ➢ a building that lasts is adaptive and can change over time
  ➢ lasts for generations without total rebuilding
example

built 1860  today
structure foundation

- the eclipse plug-in architecture
- everything is a plug-in
  - simple and consistent
platform vs. extensible application

- eclipse rich client platform
  - it has an open, extensible architecture
  - built out of layers of plug-ins
everything is a plug-in

- plug-in
  - set of contributions
  - smallest unit of eclipse functionality
  - declares its pre-requisites
everything is a plug-in

- extension point
  - named entity for collecting contributions
everything is a plug-in

- extension
  - a contribution
everything is a plug-in

- the platform run-time is making the connection
eclipse plug-in architecture

- **plug-in == component**
  - set of contributions
  - smallest unit of Eclipse function
  - details spelled out in plug-in manifest
  - big example: mail client
  - small example: action to calculate the number of lines of a mail
- **extension point** – named entity for collecting contributions
  - Example: extension point to add additional spam filtering tools
- **extension** – a contribution
  - Example: a specific spam filter tool
- **RCP - Platform** – set of standard plug-ins
- **Runtime** – controls and manages contributions
scalability

user visible appearance
<action
  toolbarPath="search"
  icon="icons/opentype.gif"
  toolTip="Open Type"
  class="org.eclipse.jdt.OpenTypeAction"/>

lazily instantiated using reflection

org/eclipse/jdt/OpenTypeAction.class

contribution implementation

Declarative Definition (manifest)

Procedural Implementation (Java JAR)
OSGi – Open System Gateway Initiative

- Brings modularity to Java
  - Named, versioned bundles
  - Dependency management
  - Explicit imports/exports
  - Built-in security
  - Dynamic
- Independent industry standard
- Has become popular in cell phone system management and other areas
- These are exactly the characteristics we want for advanced Smart Clients
eclipse plug-in architecture: describing a plug-in

- **3.1 separates plug-in description into two manifests**
  - prior to 3.1 a single manifest was used

- **plug-in manifest files**
  - to describe *component attributes*
    - OSGI META-INF/MANIFEST.MF
  - to describe *extensions*
    - plugin.xml

- **Plug-in development tools**
  - provide a combined editor
  - perform semantic checking of the contents
manifests – component model - OSGi

Bundle-Name: HP35-OSGi
Bundle-SymbolicName: HP35-OSGi; singleton:=true
Bundle-Version: 1.0.0
Bundle-Activator: HP35.HP35Plugin
Require-Bundle:
org.eclipse.ui.views,
org.eclipse.ui,
org.eclipse.core.runtime
Eclipse-AutoStart: true
Export-Package:
HP35,
HP35.views

Plug-in identification

Required Plug-ins

exported packages

Declare contribution this plug-in makes
manifests – extensions ⇒ plugin.xml

```xml
<plugin>
  
  <extension>
    <view
      name="HP35"
      icon="icons/calculator.jpg"
      category="HP35"
      class="HP35.views.HP35View"
      id="HP35.views.HP35View">
    
    </view>
  
  </extension>

  <extension-point id="operators" name="Operators"
    schema="schema/operators.exsd"/>

</plugin>
```

Declare contribution this plug-in makes

Define new extension point open for contributions
Contributing an extension

- **plug-in org.eclipse.ui**
  - declares extension point org.eclipse.ui.popupmenus
  - declares interface IActionDelegate
- **plug-in castcounter**
  - implements interface IActionDelegate
  - contributes class CastCountAction to extension point
- **plug-in org.eclipse.ui** instantiates CastCountAction
developing and deploying plug-ins

- **plug-in development environment** (PDE)
  - specialized tools for developing eclipse plug-ins
    - e.g. editor for plug-in manifest files
  - templates for new plug-ins
  - implemented as plug-ins atop eclipse platform and JDT
  - PDE runs and debugs another eclipse application

- **features** group plug-ins into installable chunks
  - features downloadable from url addressable location
  - obtain and install new or update existing plug-ins
contribution cycle...

- the *contribution cycle*
- publish the plug-in
  - create a feature
    - groups plug-ins into installable chunks
  - create an update site
    - contains zips for features and plug-ins
- enable extensions
  - define extension points
- extenders can extend your extensions
managing plug-ins: install/update

- **features** group plug-ins into installable chunks
  - feature manifest file

- plug-ins and features bear version identifiers
  - major . minor . service
  - multiple versions may co-exist on disk

- features downloadable from URL addressable location
  - using Eclipse Platform update manager
  - obtain and install new plug-ins
  - obtain and install patches & updates to existing plug-ins

- support for update site mirroring & shared installations
ready to use; easy to adopt

- component model
  - simple but consistent
- plug-ins have to be easy to develop ⇒ tool support
  - Java Development Tools (JDT) + Plug-in development environment (PDE)
- scale-up to thousands of installed plug-ins
  - the problem is start-up time
    - lazy loading
- plug-ins have to be easy to manage and discover
  - install, update, disable
  - built-in install/update support
services plumbing: APIs

- APIs matter
  - define consistent, concise API
    - define API conventions (*.internal.* in eclipse)
  - don’t expose the implementation
  - develop implementation and client at the same time
- define APIs for stability
  - binary compatibility is highest priority
  - we would rather provide less API than desired (and augment) than provide the wrong (or unnecessary) API and need to support it indefinitely
component centric

- component centric development
  - a team is responsible for one or more component dependencies through APIs
  - ensures high velocity development inside a component
    - *eclipse 3.1 provides tools support to check for API access violations*
      - *as you type*
- define producer/consumer relationships among components
  - tension among components is healthy for coming up with good component interfaces/APIs
APIs first

- APIs don’t just happen; we need to design them

- specifications with precisely defined behavior
  - what you can assume (and what you cannot)
  - it works ≠ API compliant
  - documented classes ≠ API
  - “provisional API” = API that didn’t make it for a release

- must have at least one client involved, preferably more

- need an API advocate
  - we all care about having sustainable APIs
  - need someone who lives and breathes APIs
example: API evolution in the Java Development Tools

- new APIs
  - AST (Abstract Syntax Tree)
  - AST rewriting
    - code manipulation
- open-up
  - contribute to quick fix/quick assist
  - contribute to code assist
- push-down
  - make JDT specific support available to other languages:
    - template processors
    - linked editing
stuff, furniture - UI

- eclipse extension architecture is contribution based
  - extensions contribute to the workbench
  - the workbench manages and presents the contributions
- enables UI evolution
  - 3.0 new look
  - 3.1 new preferences
keeping the house clean…

- blur the boundary between the Platform and contributions
  - plug-ins should fit in (and together) naturally
  - this is a shared responsibility for all plug-in developers

**housekeeping rules**

- **sharing rule**: Add, don’t replace
- **invitation Rule**: Whenever possible, let others contribute to your contributions
- **fair Play Rule**: All clients play by the same rules, even me.
- **Relevance Rule**: Contribute only when you can successfully operate
- **Integration Rule**: Integrate and don’t separate
outline

- more than a Java IDE
- more than a tools platform (RCP)
a brief history of RCP

- Eclipse 3.0 (2004): We made this real by:
  - factoring out the IDE-specific aspects from the workbench
  - you get a blank slate: an empty workbench
  - but with a very rich set of features to fill it with
  - RCP-specific APIs for configuration and lifecycle notification
  - integrating with the new OSGi runtime
  - cleaning up dependencies in other components (e.g. Help and Update)
- Eclipse 3.1 (2005): improved tooling, some new APIs
what is RCP?

while the Eclipse Platform is designed to serve as an open tools platform, it is architected so that its components can be used to build just about any client application. The minimal set of plug-ins needed to build a rich client application is collectively known as the **Rich Client Platform**.

- a subset of the full Platform
- there are other subsets
characteristics of a smart client application

- desktop app (not in web browser)
- run on multiple platforms, devices, and configurations
- rich UI with consistent metaphors
- tight integration with desktop OS (e.g. DnD, System Tray)
- easy deployment and server-centric management -> lower TCO
- makes use of local filesystem and other devices (printer, card reader)
- makes use of server-side resources and data too
- reduced round trips to the server
- responsive UI
- occasionally disconnected operation
  - securely caching data, knowing when to invalidate
  - indicating potentially stale data to user
Example RCP apps
IBM Workplace Client Technology, Client Administrator

- Dynamic, user-based provisioning
- Locked-down desktop
- Strong security requirements
- Restricted customizability
Maestro – NASA Space Mission Management

- Responsive UI
- Open Platform for space mission monitoring
- Bringing together different teams in NASA/JPL
eclipse rich client platform

A rich client platform needs a strong component model with the following major characteristics:

- Specified interfaces: A component must declare its public API and how it can be extended.
- Lazy loading: Components are loaded on demand not on startup.
- Versioning: Prerequisite components are referenced by name and version.
- Dynamic detection: Components are detected dynamically (no need to restart).

Additionally, the following issues must be addressed:

- Managing: Install, update, remove, and discover components.
- Development: IDE to develop components.
RCP components + optional components

- Help (optional)
- Update (optional)
- Text (optional)
- IDE Text
- Compare
- Debug
- Search
- Team/CVS
- IDE Views
- IDE (optional)
- Generic Workbench
- Resources (optional)
- JFace
- SWT
- Runtime (OSGi)
rcp in action: hp35
UI Components

- SWT - Standard Widget Toolkit
- JFace – Framework providing higher-level UI abstractions
- Workbench – Provides reusable and extensible UI metaphors
- Text - Framework(s) for building high-function text editors
- UI Forms - Framework for building forms-based views and editors
- GEF - Framework for building rich graphical editors
native integration can you tell?

RCP means you can’t tell
**SWT - Standard Widget Toolkit**

- Platform-independent native widget toolkit

UI Components
JFace

⇒ Framework on top of SWT providing higher-level UI abstractions

- Application window: menu bar, tool bar, content area & status line
- Viewers (MVC pattern)
- Actions, action bars (abstracts menu items, tool items)
- Preference and wizard framework
Workbench

- Defines reusable and extensible UI metaphors

Leverages extension point mechanism and JFace abstractions.

Provides:
- Views
- Editors
- Action sets
- Perspectives
- Wizards
- Preference pages
- Commands and Key Bindings
- Undo/Redo support
- Presentations and Themes
- Activities (aka Capabilities)

Is Dynamic-aware: responds to registry changes and adds/removes views, action sets, etc. accordingly
UI Components

UI Forms

⇒ Framework for building forms-based views and editors

- Form consisting of multiple FormParts
- Extra widgets:
  - FormText (marked-up text)
  - ScrolledForm
  - Section
  - MasterDetailsBlock
- Extra layouts:
  - TableWrapLayout (HTML-like)
  - ColumnLayout (newspaper-like)
- Flat look, lightweight borders
- Forms-based multi-page editor

- Used extensively in PDE
GEF (Graphical Editor Framework)

⇒ Framework for building rich graphical editors

- Draw2D - structured graphics drawing framework
- Graphical editor framework:
  - MVC architecture
  - Undo/Redo support
  - Palette and common tools for manipulating objects
  - Integration with Properties and Outline view
Eclipse Help

- HTML and XML based system
- Context-sensitive (F1) help
- Dynamic content generation
- Search engine
- Dynamic-aware
- Highly scalable (used on ibm.com)

Help UI
User interface and dialogs

Help Core
API to access the documents

plugin.xml

```xml
<extension point="org.eclipse.help.toc">
  <toc primary="true" file="doc/guide.xml" />
  <toc file="doc/tipsAndTricks.xml" />
</extension>
```

guide.xml

```xml
<toc label="Go wild user Guide">
  <topic label="Getting Started">
    <anchor id="gettingstarted" />
  </topic>
  <topic label="Commands" href="doc/cmds.html" />
</toc>
```
Building a help plug-in

In this example, we assume that a documentation author has already supplied you with the raw documentation in the form of HTML files. The granularity and structure of these files is completely up to the documentation team. Once the documentation is delivered, setting up the plug-in and topics can be done independently.

We start by assuming that the documentation has already been provided in the following tree:

```
html/  
    concepts/  
        concept1.html  
        concept1_1.html  
        concept1_2.html  
    tasks/  
        task1.html  
        task2.html  
        task3_1.html  
        task3_2.html  
    ref/  
        ref1.html  
        ref2.html
```

User Assistance

Intro support

- Provides the “welcome experience” for your product

- HTML / CSS or SWT / Forms based

- Can run actions to drive the UI
Cheat sheets

- Guides the user through a series of complex tasks to achieve a goal
- Content written in XML
- Can run actions to drive the UI
Deployment and Update

Updating your application – Update UI

![Diagram of the Eclipse IDE showing the Install/Update window and the Product Configuration window. The Install/Update window has options for feature updates and deployment sites to visit. The Product Configuration window shows the Eclipse Platform configuration.](image)
Deployment and Update

Eclipse and Java Web Start

- RCP apps can be Java Web Start’ed

- PDE Feature Export helps with JAR signing and creating JNLP manifests
EMF (Eclipse Modeling Framework)

- **Code generation from models**
- **Supports model-driven development**

- Input models can come from:
  - by hand (using EMF model editor and Properties view)
  - from some existing design (e.g. UML from RAD or EclipseUML)
  - from annotated Java interfaces
  - some serialization schema (e.g. XSD, WSDL)
- generates efficient model code (incremental, non-destructive)
- efficient dynamic API for manipulating EMF objects and their metadata
- notification of changes to model objects (great for MVC)
- change recording and summarization
- serialization, e.g. to/from XML or XMI
  - lazy-loading of objects linked across different resources
- understands: relationship arity, inverse relationships, strong containment vs simple reference
- framework for model validation
EMF (Eclipse Modeling Framework)

- can generate supporting adapters for:
  - showing EMF models in JFace viewers (with automatic update)
  - enabling command-based (undoable) editing
  - editing model objects in the Properties page
- can even generate a complete RCP app for editing instances of your model (good for rapid prototyping)
BIRT (Business Intelligence and Reporting Tools)

- Business Intelligence and Reporting Tools based on Eclipse
- Initially focused on embedded reporting for Java developers
- Proposal has 4 initial projects
PDE (Plug-in Development Environment)

- New plug-in project wizard with templates
PDE cont’d

- Product configuration and branding editor
- Can easily test and export from here
VE (Visual Editor)

→ GUI builder and framework for creating GUI builders
WTP (Web Tooling Project)

⇒ Tools for developing Web and J2EE applications
RCP summary

- rich set of functionality
- pervasive Plug-in architecture
- extensible
  - extensions and Extension Points
  - workbench provides many extension points
- scalable
  - supports large products (like RAD, Lotus Workplace)
  - scales down to embedded devices (eRCP, JCL/Foundation)
  - progressive exposure to functionality (perspectives, activities)
  - aggressive laziness
- customizable (see examples)
- dynamic
- interoperable: COM/OLE, AWT/Swing

Middleware for Applications
on the way to 3.2…
conclusions

eclipse perception changes

1.0 “eclipse is a Java IDE”

2.0 “eclipse is a general tooling platform”

3.0 “eclipse is a general application platform”
where can I find out more?

- [www.eclipse.org](http://www.eclipse.org)
- RCP UI page:
- Ed Burnette’s RCP tutorials
- Platform and RCP newsgroups:
  [news://news.eclipse.org/eclipse.platform.rcp](news://news.eclipse.org/eclipse.platform.rcp)
  [news://news.eclipse.org/eclipse.platform](news://news.eclipse.org/eclipse.platform)
- Gamma, Beck: Contributing to Eclipse – Principles, Patterns, and Plug-ins, Addison-Wesley, 2004
  [www.awprofessional.com/series/eclipse](www.awprofessional.com/series/eclipse)