Eclipse Plug-In Development

Keyvan Nayyeri

http://nayyeri.net

Overview

- Fast-growing and fast-changing requirements in software
- New methodologies to respond to these changes
- Demand for better ways and tools to handle frequent updates in the codebase
- Expandability as a common need for software
- Dynamic Extensibility as the solution
- Abstraction as the basis of extensibility
- New extensibility features in programming languages

Eclipse

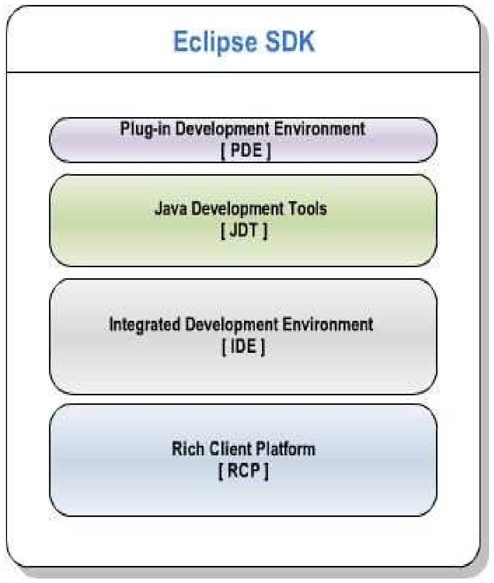
- One of the best development environments
- Multilingual support (Java, C, C++, COBOL, Python, Perl, PHP, Scala)
- Primarily for Java development
- Multi-platform (Linux, Windows, OpenSolaris, Mac)
- Originated from VisualAge
- Open Source and free
- Developed by the Open Source community
- Licensed under Eclipse Public License
- Written in Java

Eclipse and Extensibility

- Extensibility as a main goal in initial design
- The core is Eclipse SDK
- Plug-in as the main extensibility feature
- Many plug-ins added to the core
- Plug-in Development Environment (PDE) provides tools for plug-in development
- Different SDK levels consist of various plug-ins
- All the Eclipse functionality is located in different plug-ins

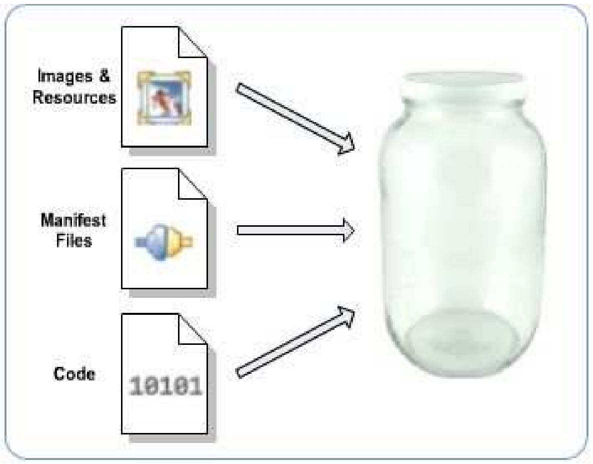
Eclipse SDK Structure

- RCP: Framework for building rich application
- IDE: Tools platform for building forms of tooling
- JDT: A complete Java IDE
- PDE: Tools necessary to develop plug-ins and RCP applications


Eclipse SDK
Plug-in Development Environment [PDE]
Java Development Tools [JDT]
Integrated Development Environment [IDE]
Rich Client Platform [RCP]

Eclipse Plug-In

- A small unit of Eclipse platform developed separately
- A JAR file
- It bundles all the code and resources
- Self-describing
- Either extends another plug-in or provides extensibility points for other plug-ins



Structure of a Plug-In

- A set of Java classes implementing the behavior
- The OSGi Bundle manifest file (Manifest.FM) explaining the expectations from the outside environment
- The plug-in manifest file (plugin.xml) describing the plug-in extensions and extension points

Plug-In Class

- Each plug-in has as a class representing it
- A plug-in class must extend org.eclipse.core.runtime.Plugin abstract class directly or indirectly
- org.eclipse.core.runtime.Plugin class provides facilities for managing plug-ins
- The Eclipse plug-in wizard names it Activator by default
- The class has start and stop methods
- It also has initialization and cleanup methods

OSGi Bundle Manifest

- Located in META-INF directory
- Deals with the runtime details of a plug-in
- Can be edited manually or using an Eclipse editor
- Implements the complete OSGi R4 Framework specifications
- Helps specifying the runtime dependencies of the plug-in
- Consists of a set of headers with values

Plug-In Manifest

- Named plugin.xml
- Deals with extensions and extension points
- Includes definitions for extension points
- Each extension point has an XML schema

Extension Points

- Eclipse provides several extension points
- Extension points provide additional functionality
- For example, extensions for runtime, builders, editors, menus, themes, and views
- You can build your own extension points
- You can extend any of the existing extension points

Lazy Loading

- Plug-ins load whenever they are needed
- They’re not loaded until it is required
- Lazy loading improves the start up speed and performance of Eclipse
- You can make your plug-in to start when Eclipse launches
- org.eclipse.ui.startup is an extension point to use to load a plug-in at start up
- The startup class must implement org.eclipse.ui.IStartup interface with *earlyStartup* method

Bundle Context

- Eclipse associates a *BundleContext* with each plug-in when it is started
- Provides information about plug-in
- Also provides information about other plug-ins
- *BundleEvent* provides events for different steps in plug-in lifecycle

Bundle

- Bundle and Plug-In may be considered the same entities in Eclipse
- *Bundle* class provides OSGi unit of modularity
- Six states for bundles (Uninstalled, Installed, Resolved, Starting, Stopping, Active)

Deployment

- Bundle the files and copy them to plug-in folder of Eclipse installation
- Use the Lunch button in Overview tab of Manifest editor
- P2 update site for online distribution

Demo

- Learn the steps to create a new plug-in project
- Understand the project structure
- Understand the code structure
- Understand the OSGi manifest file
- Understand the plugin.xml manifest
- Test the plug-in

References

- Eclipse plug-in development RefCard by DZone (http://refcardz.dzone.com/refcardz/eclipse-plug-development)
- Eclipse Plugin Development (http://www.eclipsepluginsite.com)

Resources

- PDE Does Plug-ins (http://www.eclipse.org/articles/Article-PDE-does-plugins/PDE-intro.html)
- Eclipse plug-in central (http://www.eclipseplugincentral.com)
- Developing Eclipse plug-ins (http://www.ibm.com/developerworks/library/os-ecplug)

Summary

- Extensibility as a growing requirement
- Java and Dynamic Extensibility
- Eclipse is extensible
- Plug-ins as the primary source of extensibility in Eclipse
- Java as the language for developing Eclipse plug-ins
- Manifest and source code as the main components of an Eclipse plug-in

Questions

Do you have any questions?