Using Eclipse as a Tool-Integration Platform for Software Development

Zhihui Yang and Michael Jiang, Motorola

It’s the most powerful tool-integration platform ever in the software world, and it’s open source. Eclipse has achieved what many industry associations have tried since the 1980s—to provide an easy-to-use, robust, open, industry-level platform for plugging in tools so that they can interact and integrate seamlessly with each other. It’s a proof point for the business and spirit that drives the Free and Open Source Software movement—and will continue to grow it. This summary of Eclipse’s practical use comes from an industry giant, Motorola, and we thank the authors for illuminating the benefits and best practices from their experience. —Christof Ebert

Eclipse is an open source software project dedicated to providing a robust, full-featured, and commercial-quality platform for developing and supporting highly integrated software engineering tools. The Eclipse platform defines a set of frameworks and common services that collectively make up the “integrationware” required to support a comprehensive tool-integration platform. Excepting the small Eclipse runtime kernel, all the platform components are plug-in tools integrated seamlessly through pre-defined extension points.

As a universal tool-development platform, Eclipse supports deliverables throughout the development and maintenance life cycle (see figure 1). By using the plug-ins corresponding to each life-cycle phase, developers avoid having to switch between tools to work on different project phases and to manage the interactions of tools and the deliverables they produce.

Comparison with other platforms

Many software development tools use XML for tool integration. The XML standard lets developers integrate applications at the data level with a well-formed hierarchical representation. XML defines specifications for expressing application semantics, such as the I/O behavior and invocation method, to access services in a platform-independent manner. For instance, the Web Services Description Language and the SOAP object-invocation protocol are widely deployed technologies that rely on XML to support the integration of applications, services, and tools.

Various integrated software development environments enable development tools to work together seamlessly at different integration levels. Of the existing integration platforms, NetBeans (www.netbeans.org) and IntelliJ IDEA (www.jetbrains.com/idea) are the most comparable to Eclipse for Java developers. From a technical perspective, these platforms have far more similarities than differences, and all are extensible through plug-ins. However, Eclipse benefits from a larger and growing body of contributors worldwide. The number of Eclipse plug-ins and
Saving face. When you start up the workbench, implements the workbench user inter-
extend. For example, a group of plug-ins define new extension points for others to
Eclipse platform offers. Plug-ins can also predefined extension points that the
ation. Plug-ins add functionality through
framework to create a usable applica-
ponents. Other tools plug into this basic
bench, workspace, help, and team com-
ily using the export wizard. Eclipse also
provide a set of services that you can use
to extend graphical editors. GMF also
levers other Eclipse plug-ins, such as
EMF Technology-Transaction, Validation,
and Object Constraint Language, to enable model-level validation.

At Motorola, we experimented with integrating tools specific to different life-
cycle development phases into the Eclipse environment. The service-creation frame-
work is one tool-integration project using Eclipse. We designed it to facilitate the
creation and deployment of mobile data applications and services, such as text and
multimedia messaging and location- and presence-based services. We implemented
it through the set of integrated tools shown in figure 2. The GMF plug-in im-
plements the graphical user interface, letting developers use visual programming
to build applications. The framework maps a diagram element in the editor to
one or multiple predefined service com-
ponents at the component layer.

We developed the second tier, the do-
main model layer, on top of an EMF plug-in to specify the service-creation
domain rules and constraints. This tier also enables application validation at the
model level. We adopted the Web Tools Platform plug-ins to facilitate the imple-
mentation of our own plug-ins for
service components as J2EE components and Web services. These
ponents and services handle the in-
tegration with legacy systems and data
orage. We also used WTP to package
and deploy the final application.

Hints for practitioners

We also uncovered challenges. First,
the lack of good documentation and
guidelines for plug-in development results
in a rather steep learning curve. Although
Eclipse’s rich features can greatly reduce the amount of coding needed for developing plug-ins, figuring out where these features are and how to use them can take a lot of time. The most helpful resource we found to address these issues is the Eclipse plug-ins site (see the “Eclipse Resources” sidebar). The site organizes plug-ins into 61 categories and includes a functional description, developer comments and ratings, and location information for each plug-in.

Frequent updates of the Eclipse platform and installed plug-ins can also cause incompatibilities. You must have an engineering management process in place to keep the platform and various plug-in tools in sync and to make sure the same development environment is available to all team members.

Despite Eclipse’s wide range of plug-ins, you must often design your own to meet specific requirements and create unique solutions. Before you decide to develop a plug-in, it’s a good practice to decide the level of tool integration for the development environment (see the “Eclipse Integration Levels” sidebar). Different integration levels determine what tools to use and what extension points to extend for the integration. To check the extension points provided by a particular tool or plug-in, we often rely on the Eclipse framework’s help component. It gives extension points and descriptions for most plug-ins installed on the current Eclipse platform.

Our research leads us to believe that Eclipse is a cost-effective, productive development environment to support life-cycle software development through effective tool integration. The flexible plug-in architecture and resource management let organizations customize and extend Eclipse to meet their engineering processes, tool integration, and project management needs.

Zhihui Yang is a senior software engineer at Motorola’s Software and System Engineering Research Lab. Contact her at zhihui.yang@motorola.com.

Michael Jiang is a distinguished member of the technical staff at Motorola’s Software and System Engineering Research Lab. Contact him at michael.jiang@motorola.com.

**Eclipse Resources**

For basic concepts about the Eclipse plug-in architecture, extensions, and examples, see the Eclipse Foundation site (www.eclipse.org), especially www.eclipse.org/articles/article-Plug-inarchitecture/plugin_architecture.html.


Eclipse-related publications are available at www.eclipse.org/resources/index.jsp.

For detailed information about how to set up Tomcat and JBoss in Eclipse for J2EE development, see www.purposesolutions.com/Resources/EclipseJ2EE.html.

The Eclipse Project Resource Center offers many tutorials, articles, magazines, plug-ins, cheat sheets, conference lists, and the latest books at www.deitel.com/Eclipse.

**Eclipse Integration Levels**

Eclipse supports a number of integration levels for developers to target specific tool needs. Among them, the following levels are especially useful.

**Invocation integration** lets you configure the Eclipse platform to invoke a tool on a specific resource type in a separate window. It can’t integrate with other Eclipse views and editors.

**Data integration** allows data to be exchanged or manipulated by tools. Data integration provides state reuse, but not behavioral reuse.

**API integration** allows applications to be accessed through tool-specific APIs, which simplify client application development by providing data encapsulation and behavioral reuse.

**UI integration** lets a tool participate with other tools as if they were designed as a single application. It requires using the Eclipse UI framework to build the tool’s UI and integrating the result with the Eclipse workbench through its extension points.

**Reference**