Maija Mustonen

Eclipse RCP and Eclipse Java IDE

Bachelor's Thesis
Computer Science and
Information Systems
16th March 2007

University of Jyväskylä
Department of Computer Science and Information Systems
Jyväskylä

ABSTRACT

Mustonen, Maija Elina Loviisa
Eclipse RCP and Eclipse Java IDE/ Maija Mustonen
Jyväskylä: University of Jyväskylä, 2007.
25 pages
Bachelor's Thesis

After it was first released in 2001, Eclipse has evolved into the most popular professional Integrated Development Environment (IDE) for the Java programming language. This thesis is a literature review of Eclipse both as a plug-in based Rich Client Platform (RCP) and as a Java IDE.

Data have been collected from books, scientific articles, official websites, and technical manuals. Furthermore, a simple Eclipse plug-in is introduced and the code explained. The main contribution of the thesis is to form an extensive and up-to-date description of the Eclipse phenomenon.

KEYWORDS: Eclipse, IDE, plug-in architecture

TABLE OF CONTENTS

1 INTRODUCTION

A few decades ago, a text editor, a simple command-line compiler, and possibly a debugger were sufficient to develop software. Programs were composed offline and debugging meant simply inserting output statements to the code.

Today, these tools have been combined in increasingly sophisticated and graphical integrated development environments (IDEs), which usually come with features such as syntax highlighting, code auto-completion, refactoring, unit testing support, and visual editors. IDEs for object-oriented languages offer tools for managing classes and objects, such as class hierarchy views. Needless to say, IDEs are making software development increasingly productive.

Java developers can choose a commercial product, such as JBuilder or IntelliJ IDEA, or a free and open source equivalent, such as Eclipse or NetBeans. This thesis is about Eclipse, which continues to gain popularity (Evans Data Corporation, 2006).

For many people, Eclipse is simply a Java IDE, but the phenomenon actually has three aspects. According to D'Anjou, FairBrother et al (2004), Eclipse is:

1) a rich client platform (RCP), for developing rich clients, as opposed to thin clients, such as web pages
2) a Java IDE
3) an open source community, Eclipse Foundation, that coordinates the Eclipse development. Members include companies such as IBM, Borland, Intel, Motorola, Nokia, and SAP (Eclipse Foundation, 2007a).

In Figure 1, Eclipse is pictured as an onion to illustrate the three aspects. The Java IDE is in the core and the RCP is wrapped around it, because the platform is a larger entity and includes not only the Java IDE, but other applications, such as the Eclipse C++ IDE. The largest entity of all is the Eclipse community.

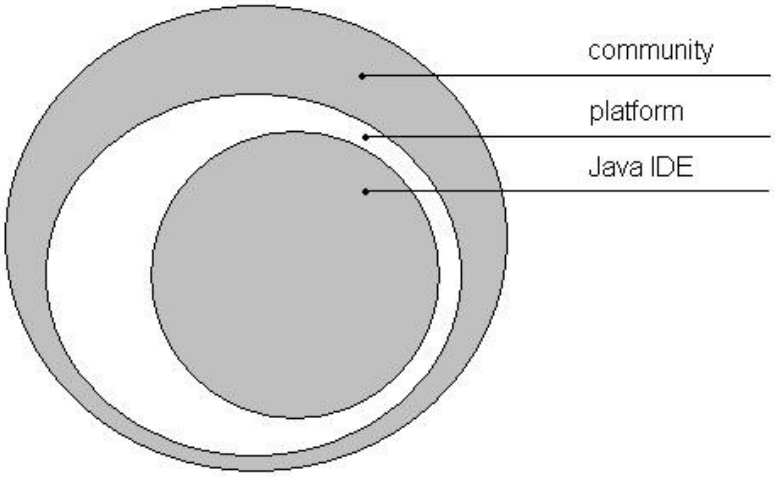

Image 1: The three aspects of the Eclipse phenomenon

Eclipse was born in 2001, when IBM released Eclipse 1.0 under the Common Public License (CPL). The current version is 3.2.1. Here is a brief list of the most important phases of the history and future Eclipse (Eclipse Foundation, 2007a).

- November 2001, Eclipse 1.0 release
- July 2002, Eclipse 2.0 release
- June 2004, Eclipse 3.0 release
- January 2005, license switch from CPL to EPL (Eclipse Public License) complete
- June 2006, Callisto simultaneous release, including Eclipse 3.2
- June 2007, Europa simultaneous release, including Eclipse 3.3

Already in 2005, Eclipse was cited as the most popular professional Java IDE (Goth, 2005 and Geer, 2005), and its growth continues. Evans Data Corporation (2006) estimates the use of the Eclipse RCP to triple within two years. Eclipse is an open source success story, having moved from anonymity to the spotlight even faster than projects such as OpenOffice and Mozilla (Ingo, 2005).

1.1 Research Strategy and the Aim of the Study

Like many other phenomena in the area of Information Technology, Eclipse is relatively new and constantly evolving. Even though version 1.0 was released more than five years ago, the latest important release, 3.2, was introduced as recently as June 2006. These two can hardly be juxtaposed.

Consequently, there is a shortage of up-to-date academic material concerning Eclipse. The aim of the study is to describe the phenomenon thoroughly from several points of view. This way, we aim to form an extensive and up-to-date description of Eclipse.

Data were collected from a variety of sources: Books and scientific journals, but also from official web sites and technical manuals. Moreover, a simple plug-in, with code examples, will be introduced.

1.2 Limitations

Because of its extensible nature and the growing number of plug-ins available, Eclipse is increasingly used for developing with other languages, such as C++ and PHP, as well. In this thesis, however, only the aspect of Java IDE is considered in detail.

The community nature of Eclipse is interesting, yet describing it further would make this thesis inappropriately lengthy. Hence, it has been left out.

1.3 Structure of the Thesis

In the next chapter, Eclipse is described as a rich client platform. First, we will have a closer look at its plug-in based architecture and secondly, a simple plug-in is described. In chapter 3, we will zoom closer into one of the Eclipse-based rich clients: the Eclipse Java IDE and cover its features and users. In addition, we will have a look at its competitors, particularly the NetBeans IDE. The final chapter is for the conclusions.

2 ECLIPSE RCP

In this chapter, we take a software architect's point of view on Eclipse. We will have a look at the plug-in based architecture of the Eclipse RCP and introduce a Hello World plug-in.

Historically, the Eclipse RCP was marketed as a "Universal IDE", "an IDE for anything and nothing in particular" (Vaughan-Nichols, 2003). That is, a platform for building and integrating development tools. Yet increasingly, it is a platform for arbitrary rich clients that have little or nothing to do with software development (des Rivieres & Beaton, 2006). A rich client is the opposite of clients like command line tools or web pages. Rich clients run locally and feature advanced UI functions, such as clipboard, drag-and-drop, and navigation (McAffer, Lemieux, 2005).

Rich clients have many desirable qualities, but designing, implementing, testing, and maintaining them requires considerable effort. Therefore, a suitable existing framework would save time and money. Williams (2007) has listed reasons why Eclipse is such a framework:

- Implements a clear, consistent, and cohesive architecture
- Supports development and execution on all the major desktop platforms (Windows, Mac OS X, Linux, QNX Photon, Pocket PC, HP-UX, AIX, Solaris)
- A snappy UI response that maintains the platform's native look-and-feel
- Provides a large variety of widgets, both standard (i.e., button and checkbox) and extended (i.e., toolbar, tree view, and progress meter)
- Provides extensive text processing that includes editors, position/change management, rule-based styling, content completion, formatting, searching, and hover help
- Supports using platform-specific features (i.e., ActiveX) and legacy software, if desired
- Enables branding the application

- Contains an integrated help system
- Manages user configuration and preferences
- Supports remote discovery and installation of application updates
- Created and backed by respected software companies experienced in creating object oriented frameworks
- Supports internationalization and national language translation
- Designed for flexibility with natural features for adding new functionality
- "Pay" only for what you need - base frameworks can be easily reduced as well as extended to tailor capabilities to precise requirements

2.1 Architecture

Eclipse is written in Java and developed with the Eclipse Java IDE. Except for the kernel, Platform Runtime, everything in Eclipse is a plug-in (des Rivieres & Beaton, 2006). This kind of architecture could be called "pure plug-in architecture", as opposed to traditional plug-in architecture (Birsan, 2005). Image 2 represents the Eclipse architecture. We will now cover SWT, JFace, plug-ins, and the Platform Runtime in more detail.

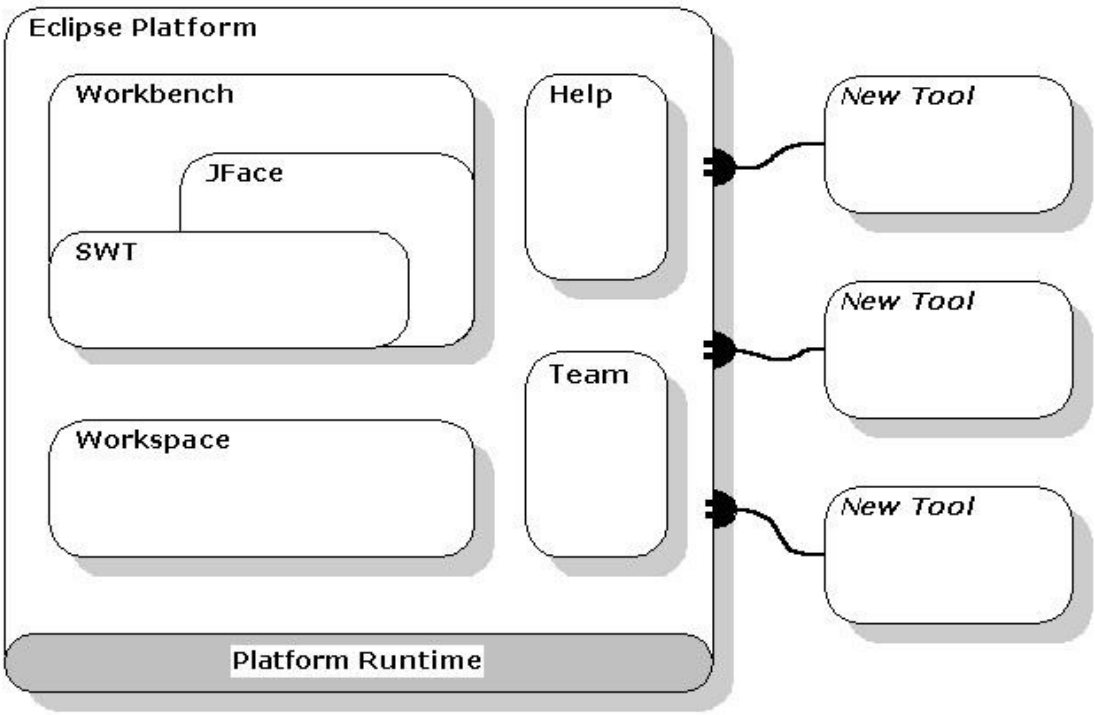

Image 2: The Eclipse architecture (des Rivieres & Beaton, 2006)

2.1.1 SWT

The Java language has two official GUI toolkits: the Abstract Window Toolkit (AWT) and Swing. AWT is tied to the underlying platform's windowing system and thus AWT components are fast yet look different on different operating systems. Unfortunately, pure AWT programming means programming to the least common denominator of all window systems (des Rivieres & Beaton, 2006).

Swing was developed later on to make up for AWT's performance problems. It is pure Java. Consequently, Swing is slower, but looks consistently the same everywhere. In addition, Swing is more sophisticated and offers high-level widgets such as trees and rich text.

Eclipse has however developed its own GUI toolkit called SWT (Standard Widget Toolkit). The decision was made with the intention to produce applications that shared the same look and feel with the underlying platform. Native widgets are used when possible. If no native widget is available, SWT provides an emulation. (D'Anjou et al, 2005)

In principle, it is possible to develop AWT/Swing applications with Eclipse, but for the plug-in development, SWT is essential.

2.1.2 JFace

JFace is a UI toolkit that simplifies common UI programming tasks. It is layered on top of SWT and works together with it without hiding it. JFace includes components such as font registries, dialogs, and frameworks for preferences. (McAffer. Lemieux, 2005)

2.1.3 Plug-ins

In Eclipse, a plug-in is the basic unit of functionality. Complex applications consist of several plug-ins. An Eclipse plug-in usually consists of a manifest file and Java classes in a JAR archive. They can also contain non-code contents, such as images and documentation. (D'Anjou, Fairbrother et al, 2005)

The manifest file declares the possible plug-in contributions: extension points and extensions. Extension points can be extended by other plug-ins whereas extensions extend these extension points. Gamma and Beck (2004) compare extension point to a power strip. Extensions are are the "power plugs" that are plugged into the strip.

According to Kestler (2004), the Eclipse architecture is like a neighbourhood, where one bad neighbour can affect all residents in a negative way. Concretely: If one plug-in uses resources such as memory wastefully, the whole application suffers. Milinkovich and Freeman-Benson (2005) present the following Eclipse architecture "housekeeping rules":

- Sharing: Add, don't replace.
- Invitation: Whenever possible, let others contribute to your contributions
- Fair Play: All clients use the same rules, even me
- Safe Platform: As the provider of an extension point, you must protect yourself against misbehaviour on the part of extenders
- Stability: Once you invite someone to contribute, don't change the rules => API compatibility

2.1.4 Platform Runtime

The Eclipse Platform Runtime is the small kernel of Eclipse. It is responsible for managing the plug-ins. Each plug-in has a manifest file and in start-up, the Platform Runtime reads them and generates an in-memory plug-in registry. However, the plug-ins are not loaded before they are actually needed, which

reduces the start-up time and overall memory footprint of the application. Eclipse has a principle to postpone code execution when possible. Once activated, the plug-in remains active until the Platform shuts down, unless it is explicitly deactivated (des Rivieres & Beaton, 2006).

2.2 Hello World Plug-in

We will now introduce an Eclipse plug-in, probably the simplest one possible. Our plug-in adds a new menu item, "Hello World", at the bottom of the existing Help menu. When it is selected, a "Hello, World." message dialog is displayed.

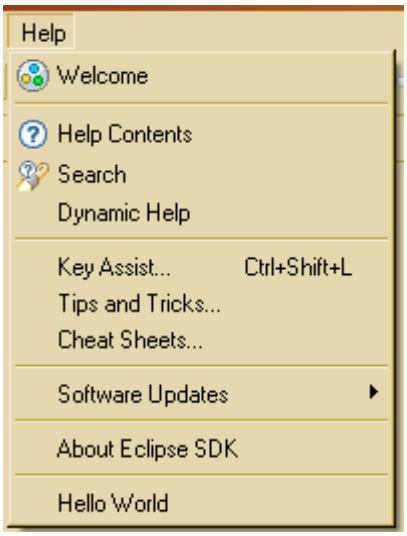

Image 3: The "Hello World" menu item in the Eclipse Help menu

Each plug-in is exported to a JAR file and resides in directory "plugins", under the Eclipse installation directory. To avoid name clashes, Sun's Java package naming convention is used. This means that the prefix is written in lowercase and should be one of the top-level domain names. Subsequent components depend on the organisation's internal naming conventions (Sun Microsystems, Inc., 1999). For example, all JDT package names start with "org.eclipse.jdt".

Next, we will have a closer look at the two manifest files and a class file.

2.2.1 Plug-in Manifest File

The plug-in manifest file uses XML format and is named plugin.xml. It describes how the plug-in should be integrated into the framework. In the early versions of Eclipse, this file was used by the Platform Runtime to access the execution-related information. Today, this information is stored in the bundle manifest file instead. Plugin.xml has remained the home for declaring extensions and extension points. (McAffer, Lemieux, 2006)

Example 1 shows the code for our plugin.xml file. The action tag declares an action that the user can access via the UI. When the action is clicked, Eclipse creates an instance of the class referenced in the class attribute (in bold) and calls its run() method. MenubarPath (in bold) defines where in the menubar the action should be placed. There are other attributes available as well. Icon, for example, defines the path to an icon file displayed with the action. ToolbarPath defines where in the toolbar the action is placed and tooltip defines the tooltip text which is displayed when the action in the toolbar is hovered with a mouse. (Eclipse Foundation, 2007b)

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.2"?>
<plugin>

  <extension
      point="org.eclipse.ui.actionSets">
    <actionSet
        label="Hello Action Set"
        visible="true"
        id="helloWorld.actionSet">
      <menu
          label="Help"
          id="help">
        <separator
            name="helloGroup">
        </separator>
      </menu>
      <action
          label="Hello World"
          class="helloworld.actions.HelloAction"
          menubarPath="help/helloGroup"
```

```
            id="helloworld.actions.HelloAction">
        </action>
      </actionSet>
    </extension>

</plugin>
```

Example 1: plugin.xml

2.2.2 Bundle Manifest File

The bundle manifest file, named MANIFEST.MF, follows the OSGi R4 Framework specification (OSGi Alliance, 2007). The term "bundle" comes from the OSGi world and essentially equals to plug-in (McAffer, Lemieux, 2006). The bundle manifest file is what the Platform Runtime first looks for in order to create the plug-in registry. In case the bundle manifest file is not found, it will be automatically generated based on the plugin.xml file (D'Anjou et al, 2004).

In Example 2, "Require-Bundle" (in bold) indicates that in order to function, this plug-in requires plug-ins org.ecipse.ui and org.eclipse.core.runtime. These are all that is needed, for they in their turn require SWT, JFace and other core components (McAffer, Lemieux, 2006).

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Helloworld Plug-in
Bundle-SymbolicName: helloworld; singleton:=true
Bundle-Version: 1.0.0
Bundle-Activator: helloworld.Activator
Bundle-Localization: plugin
Require-Bundle: org.eclipse.ui,
 org.eclipse.core.runtime
Eclipse-LazyStart: true
```

Example 2: MANIFEST.MF

2.2.3 HelloAction.java

This class (example 3) implements the IWorkbenchWindowActionDelegate interface. The run() method (in bold) of this class is where the action happens. MessageDialog is a JFace class and OpenInformation() is a convenience method to

open a standard information dialog (Eclipse Foundation, 2007b).

```
package helloworld.actions;

import org.eclipse.jface.action.IAction;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.eclipse.jface.dialogs.MessageDialog;

public class HelloAction implements IWorkbenchWindowActionDelegate {
        private IWorkbenchWindow window;

        public HelloAction() {
        }

        public void run(IAction action) {
                MessageDialog.openInformation(
                        window.getShell(),
                        "Hello World Title",
                        "Hello, World.");
        }

        public void selectionChanged(IAction action, ISelection selection) {
        }

        public void dispose() {
        }

        public void init(IWorkbenchWindow window) {
                this.window = window;
        }
}
```

Example 3: HelloAction.java

3 ECLIPSE JAVA IDE

For many, Eclipse is primarily a Java IDE. Yet to be specific, the Eclipse Java IDE refers to a group of plug-ins, named JDT (Java Development Tools). JDT is just one, albeit the most prominent, of the rich clients developed on the Eclipse RCP. Atop of JDT, there is PDE (Plug-in Development Environment), which provides tools for building Eclipse plug-ins.

In this chapter, we take a Java developer's point of view on Eclipse. We will have a look on JDT features and different kinds of users of JDT and PDE. Finally, we will find out more about the competing Java IDEs and how the competition influenced IBM's original decision to open source Eclipse. We will focus on NetBeans Java IDE, which is developed by Java's creator, Sun Microsystems, and is also open source.

2.1 Features

This chapter is based on “The Java Developer's Guide to Eclipse” (D'Anjou et al, 2005). JDT has many features that support developing Java code and the purpose of this chapter is to name just a few. For instance, JDT analyses the code on-the-fly and is able to suggest ways of completing the current phrase. This feature can be used with try/catch expressions, import statements, getters and setters, and default constructors, just to mention a few.

With refactoring tools, it is possible to reorganise the code globally in a quick and flawless way. For example, the user can rename all instances of a class, variable, or method. They can also pull or push methods up or down in the class hierarchy. The Encapsulate Field tool generates getters and setters for a field and replaces all direct references with these methods.

Eclipse also supports Javadoc, a tool that takes Java source files and generates documentation in HTML. Furthermore, it has Ant, a Java-based build tool, CVS for concurrent version management, and JUnit, a unit testing tool, built in.

2.2 Usage

Eclipse is the most popular Java IDE for professional users. Developers writing Eclipse plug-ins, including the ones developing Eclipse itself, are naturally a large user group of JDT and PDE. The most famous developers using Eclipse are probably in NASA, which is using Eclipse to develop Maestro, a software for managing remote vehicles (McAffer, Lemieux, 2005).

Many software companies have a long-term strategy to work together with educators, who in turn teach their students to use their tools. According to Goldman and Gabriel (2005), Eclipse has succeeded in providing technology that the academic world are getting exited about and hence Eclipse is used in Java pedagogy and in academic research projects.

2.2.1 Java Pedagogy

According to studies (de Raadt, Watson et al, 2004, Schulte and Bennedsen, 2006), the first programming language a computing student world-wide learns is - more often than not - Java.

In the Schulte and Bennedsen study, Eclipse was the second most used IDE in introductory programming courses. The most popular one was BlueJ, a freeware Java IDE specifically designed for introductory teaching (BlueJ, 2007). The sample consisted of more than 200 answers from university, college, and high school teachers mainly in Denmark, Germany, and the US.

Using an IDE might be tempting, but there is evidence that it is not ideal while learning programming. Research concerning using Eclipse for teaching Java programming has concluded that it is too complicated for a novice programmer. Essential tasks, such as starting a project, setting up the class paths, and running the application are easier in simpler tools, such as BlueJ. (Chen & Marx, 2005 and Mueller & Hosking, 2003)

Chen & Marx (2005) found out that it was better to use a more simplistic

development environment for several weeks before introducing Eclipse. This helped students to concentrate on basic Java concepts first. Given choice, many students were reluctant to switch to Eclipse, yet its status as an industrial-strength IDE was a formidable motivator.

Another way to solve the novice problem is to utilise Eclipse's flexible architecture and develop a set of simplifying plug-ins. Penumbra (Mueller & Hosking, 2003), Gild (Storey, Michaud, Mindel et al, 2003) and DrJava plug-in (Reis & Cartwright, 2003) are examples of such extensions.

For more advanced students however, the simplistic development environments might not suit as well. The features that benefit a novice, such as hiding some Java functionality, become a burden. Furthermore, simple IDEs do not scale well to larger software projects. Neither do they teach the importance of efficient software development practices. For these reasons, Hanks (2005) has found Eclipse an excellent IDE for teaching more experienced students. Besides, open source tools have the price and the licensing advantage: Students can easily set up identical environments at home.

2.2.2 Academic Research

An increasing amount of research projects have embraced the opportunity to customise a fully-featured open source IDE with their own plug-ins. An open source platform brings two benefits: feedback about the environment and the infrastructure to share with related research groups (Lintern, Michaud, Storey et al., 2003). Lintern, Michaud, Storey et al. (2003) integrated Eclipse with ShriMP, their own visualization tool, in order to improve program navigation, assist in program understanding, and support team collaboration and project management.

IBM has been supporting academic research around Eclipse with Eclipse Innovation Awards since 2003 (IBM, 2007)

2.3 NetBeans and Other Competitors

According to Ingo (2005), IBM did not have much other choice but to open source Eclipse, because at the beginning of the decade, there were already several reasonably good Java IDEs on the market, some of them open source. It was important to develop an independent Java IDE to work together with IBM's WebSphere products, yet competing with existing IDEs would require considerable investments. As IBM's business model had already successfully mixed proprietary and open code (Samuelson, 2006), it was only natural to give it yet another try.

Today, Eclipse Java IDE's biggest competitors are IntelliJ IDEA, a proprietary and commercial IDE by JetBrains and NetBeans, an open source effort by Sun Microsystems. Borland, a member of the Eclipse Foundation, has its own Java IDE, JBuilder. However, JBuilder 2007 has been built on Eclipse RCP (Kaster, 2006). There are also other commercial IDEs built on Eclipse, such as MyEclipse and IBM WebSphere Studio.

NetBeans is a plug-in based Java IDE, just like Eclipse. It went open source in 2000 and is published under Common Development and Distribution License (CDDL) (Sun Microsystems, 2007). Even though NetBeans was open sourced before Eclipse, it has now been overshadowed. According to Goldman and Gabriel (2005), this is partly because the latter is based on a superior design. NetBeans started as a student project in 1996, whereas Eclipse was designed by experienced professionals some years later. The Eclipse team was, for example, able to see the growing success of XML.

Sun is the developer of the Java language itself. For Eclipse, this setting can be both a strength and a weakness. The advantage is that IBM has faded in the background and Eclipse is seen primarily as an open source community project. Meanwhile, NetBeans is regarded as the Sun-specific IDE. The downside of this unofficial position is the fear that Eclipse is creating Java technology without Sun, resulting in impure Java and internal conflicts. (Vaughan-Nichols, 2003).

However, in January 2007, the Eclipse Foundation finally joined the JCP (Java Community Process), Sun's participative program to develop Java. This can be regarded an act of goodwill towards Sun (Eclipse Developer's Journal, 2007).

According to Harold (2007), NetBeans has currently two advantages over Eclipse: Superior native look and better tools for designing GUIs. He estimates that by the end of 2007, Eclipse and NetBeans will each take a half of the Java IDE market, while IntelliJ IDEA's share will stay around 5%.

4 CONCLUSION

This thesis has created an extensive and up-to-date description of the Eclipse phenomenon. For many, Eclipse is just a Java IDE, yet it can also be seen as a rich client platform and an open source community. In this thesis, the two first mentioned aspects were studied more closely.

In 2001, IBM gave a press release, announcing that it is donating a 40 million dollar software, codenamed "Eclipse", to open source. Back then, even the most optimistic spectators had trouble foreseeing the immense success of Eclipse. Being open source, Eclipse requires no registration and estimating the amount of users is not straightforward. However, it has clearly developed into a noteworthy RCP and the dominant Java IDE. It is an excellent product backed not only by IBM, but other influential companies such as Borland, Intel, and SAP.

From a "Universal IDE", Eclipse RCP has evolved into a platform for rich clients of various purposes. Rich clients integrate seamlessly with the native system and offer more advanced UI functions and thus a better user experience than thin clients, such as web pages.

Eclipse has a well-designed object-oriented architecture, where everything is a plug-in. The kernel of the system is the Platform Runtime, which creates a registry of available plug-ins and their interaction, and only loads the code, if needed. In this thesis, we introduced a simple plug-in. Furthermore, we glanced at some core plug-ins, such as SWT, Eclipse's own GUI toolkit.

IDEs are making programming increasingly productive. Not only is the Eclipse Java IDE used by professional developers, but by educators to teach programming. However, industrial-strength IDEs most likely are not suitable for novice students, as they might distract them from learning the basics. Eclipse has also been used in several academic research projects.

It is hardly incorrect to claim that Eclipse has, in a short time, all but destroyed the

market of proprietary Java IDEs. There are several reasons for Eclipse's success: It can be downloaded for free, it is fast, and easily customised with plug-ins. However, its competitor NetBeans is all this as well and - what is more - the first one of the two to go open source. The difference is that Eclipse has managed to forge strong partnerships and is perceived more as a community effort, whereas NetBeans is seen as the IDE by Sun Microsystems. This might be specious, as IBM still sits in the Eclipse Foundation Board and grants annual awards for academic projects involving Eclipse software. This ensures that Eclipse keeps supporting IBM middleware, such as the WebSphere Application Server.

It is curious that the Eclipse project has decided to develop SWT, instead of using the Java standard, Swing. Native looking SWT is without doubt an advantage when competing with Microsoft's Visual Studio, but complicates Eclipse's relationship with Sun. Both SWT and Swing have their ardent supporters and internal conflicts hardly strengthen Java's position against competitors, such as Microsoft's C#. Recently, however, there have been good news in the Java community: Java has been open sourced and the Eclipse Foundation has got closer to Sun by joining the JCP.

Eclipse keeps evolving and version 3.3 is due June 2007. Research suggests that the adoption of Eclipse will increase, yet only the future will tell whether it will manage to hold on to its assets: speed, partnerships, and the community image.

REFERENCES

Birsan D. 2005. On Plug-ins and Extensible Architectures. Queue, Volume 3, Issue 2, March 2005, 40-46

BlueJ, 2007. The BlueJ Homepage. http://www.bluej.org [Checked 15.3.2007]

Chen Z., Marx C. 2005. Experiences with Eclipse IDE in Programming Courses. JCSC, Volume 21, Issue 2, December, 104-112

D'Anjou, J., Fairbrother, S., Kehn, D., Kellerman, J., McCarthy, P., 2005. The Java Developer's Guide to Eclipse, 2nd Ed. Addison-Wesley.

de Raadt M., Watson R., Toleman, M. 2004. Introductory programming: what's Happening Today and Will There Be any Students to Teach Tomorrow? Proceedings of the sixth conference on Australasian computing education, Volume 30, 277-282

des Rivieres J. & Beaton W., 2006. Eclipse Platform Technical Overview 3.1. http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html [Checked: 18.2.2007]

Eclipse Developer's Journal, 2007. Eclipse Foundation Joins Three Industry Specification Groups. http://eclipse.sys-con.com/read/322765.htm [Checked 15.3.2007]

Eclipse Foundation, 2007a. The Eclipse homepage. http://www.eclipse.org [Checked 15.3.2007]

Eclipse Foundation, 2007b. Eclipse help. http://help.eclipse.org/help31

Evans Data Corporation, 2006. Eclipse Challenges Visual Studio .NET in EMEA Region, Latest Evans Data Reports Shows.

http://www.evansdata.com/n2/pr/releases/EMEA%20Winter%2006.shtml [Checked: Checked 15.3.2007]

Gamma E., Beck, K. 2004. Contributing to Eclipse: Principles, Patterns, and Plug-ins. Addison-Wesley.

Geer D. 2005. Eclipse Becomes the Dominant Java IDE. Computer, Volume 38, Issue 7, July 2005, 16-18

Goldman, R., Gabriel, R. P., 2005. Innovation Happens Elsewhere. Morgan Kaufmann. Available online: http://dreamsongs.com/IHE/IHE.html

Goth G. 2005. Beware the March of this IDE: Eclipse is Overshadowing Other Tool Technologies. IEEE Software, Volume 22, Issue 4, July-Aug., 108-111

Hanks, B., 2006. Using Eclipse in the Classroom. Journal of Computing Sciences in Colleges, Volume 21, Issue 3, February

Harold E., 2007. Java 2007: The Year in Preview. http://www-128.ibm.com/developerworks/java/library/j-java2007.html [Checked 15.3.2007]

IBM 2007. Eclipse Innovation Awards. http://www-304.ibm.com/jct09002c/us/en/university/scholars/products/eclipse/eig.html [Checked: 8.2.2007]

Ingo, H., 2005. Avoin elämä: Näin toimii Open Source. Ingo, Henrik 2005. Available online: http://www.avoinelama.fi/html/AvoinElama-sisalto.html

Kaster J. 2006. Developer Tool Groups – Product Roadmaps. http://bdn.borland.com/article/33519 [Checked: 13.11.2006]

Kestler M., 2004. Factoring for Eclipse. Dr Dobb's Portal.

http://www.ddj.com/dept/opensource/184405901 [Checked: 12.1.2007]

Lintern R., Michaud J., Storey M. & Wu X., 2003. Plugging-in Visualization: Experiences Integrating a Visualization Tool with Eclipse. Proceedings of the 2003 ACM symposium on Software visualization, 47-56

McAffer, J., Lemieux J-M., 2005. Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications. Addison-Wesley.

Milinkovich, M., Freeman-Benson, B., 2005. What New in Eclipse? A Java Technology IDE and a Whole Lot More. 2005 JavaOne Conference. http://developers.sun.com/learning/javaoneonline/2005/tools/TS-7654.pdf

Mueller F. & Hosking L. H, 2003. Penumbra: An Eclipse plugin for introductory programming. Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange, 65-68

OSGi Alliance, 2007. The OSGi Alliance homepage. http://www.osgi.org/ [Checked: Checked 15.3.2007]

Reis C., Cartwright R., 2003. A Friendly Face for Eclipse. Proceedings of the 2003 OOPSLA workshop on eclipse technoloy eXchange, 25-29

Samuelson P. 2006. IBM's Pragmatic Embrace of Open Source. Communications of the ACM. Volume 49, Issue 10, October 2006, 21-25

Schulte C., Bennedsen J. 2006. What do Teachers Teach in Introductory Programming? Proceedings of the 2006 international workshop on Computing education research, 17-28

Storey M.-A., Michaud J., Mindel M. et al. 2003. Improving the Usability of Eclipse for Novice Programmers. Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange, 35-39

Sun Microsystems, 2007. The NetBeans homepage. http://www.netbeans.com [Checked 15.3.2007]

Sun Microsystems, Inc., 1999. Code Conventions for the Java Programming Language. http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html [Checked 15.3.2007]

Vaughan-Nichols S. J. 2003. The Battle over the Universal Java IDE. Computer, Volume 36, Issue 4, April 2003, 21-23.

Williams Todd. 2007. Eclipse: A Solid Desktop, Rich-Client, or Embedded Application Framework. Eclipse Developer's Journal. http://eclipse.sys-con.com/read/321978.htm [Checked: 8.2.2007]