Eclipse.NET:
An Integration Platform for ProjectIT-Studio

João de Sousa Saraiva, Alberto Rodrigues da Silva

INESC-ID, Instituto Superior Técnico,
Rua Alves Redol, nº 9 –1000-029 Lisboa, Portugal
joao.saraiva@inesc-id.pt, alberto.silva@acm.org

Abstract. The development of information systems is a complex task, involving technical, human and organizational issues. The results of this task are not always successful, which are apparent in the lack of software quality and in costs and schedules overruns. The ProjectIT research program addresses this specific task, however, taking different approaches comparing with the current mainstream (for instance, in the ProjectIT context we emphasize the requirement and test engineering, the system modelling, or the project management activities). From the software application point of view, ProjectIT consists in two tools: the Web-based collaborative ProjectIT-Enterprise tool and the Windows-based ProjectIT-Studio tool. This paper presents and discusses the motivation for the Eclipse.NET environment as an integration platform to support complex and rich-client Windows-based applications, particularly to support the ProjectIT-Studio tool. Eclipse.NET was ported from the homologous and original open source Eclipse project and provides a very flexible and extendible platform for .NET Windows applications.

Keywords: Eclipse, .NET, Windows-based application, Windows-based platform, ProjectIT.

1. Introduction

The activity of designing, implementing, managing and operating information systems presents obvious similarities with other engineering branches. All these projects involve the planning and management of financial resources, as well as the management of people and equipment, deadlines, quality, scope, etc. However, there are some important differences, specific and exclusive to the area of IT, and that can be pointed out:

- The **changing of requirements** by clients and/or users **is much more frequent and demanding** than in other engineering branches. For example, in civil engineering it is highly unusual that, after a bridge has been built, the citizens (i.e. the users) and the local government (i.e. the client) demand changes to it!
- **The same type of artefact representation is kept throughout the whole lifecycle**: a digital representation. This does not usually occur in other engineering branches. For example, a bridge is designed and conceived using a digital represen-

tation (i.e. according to several drawings and calculations stored somewhere in a computer system). However, in the construction phase itself, the bridge is implemented with bricks, cement, iron, etc. In most engineering branches, there is a clear distinction in artefact representation between the design phase and the implementation phase.

It is easy to realize that, while the first difference presents a disadvantage as well as a significant challenge, the second difference is a tremendous advantage of IT projects over projects from other engineering branches. It is in this scope that the research program "ProjectIT" has been promoted in the context of the Information Systems Group (GSI) of INESC-ID [8].

This paper first introduces ProjectIT and ProjectIT-Studio. Afterwards, the emerging problem of the lack of tool integration in ProjectIT is described, as well as the approach being taken to solve this problem, along with its underlying architecture. Following this, the auxiliary tools being used are presented and described. Finally, we present future work to be made using the integration platform.

2. ProjectIT and ProjectIT-Studio

In the Information Systems Group of INESC-ID [http://gsi.inesc-id.pt], we have been developing efforts to increase the productivity of the software development process. In the recent past, the group has been involved in two research projects in the area of software engineering:

- XIS, whose main contribution was the development of the XIS/UML profile intended to specify, simulate and develop information systems following a MDD approach [20] [16].
- ProjectPro, a Web based collaborative system to support the definition of basic concepts (projects, releases, sub-systems, requirements) and their relationships [15].

The current view of the functional architecture of the project is represented in Figure 1. ProjectIT-Workbench provides the basic and common infrastructure to all the other components (such as user definition, permissions, version management, and project definition). ProjectIT-Time is the component that supports, among other features, the definition of the activities to be performed throughout a project, their workflow, the artefacts involved. ProjectIT-Tests is an initiative in the area of tests engineering, closely related with requirements engineering. ProjectIT-MDD is the component that provides the analysis, design and code generation features of ProjectIT, following an approach according with the principles of model driven development. For historical reasons, it can be also known by "XIS", which stands for "eXtensible Interactive Systems".

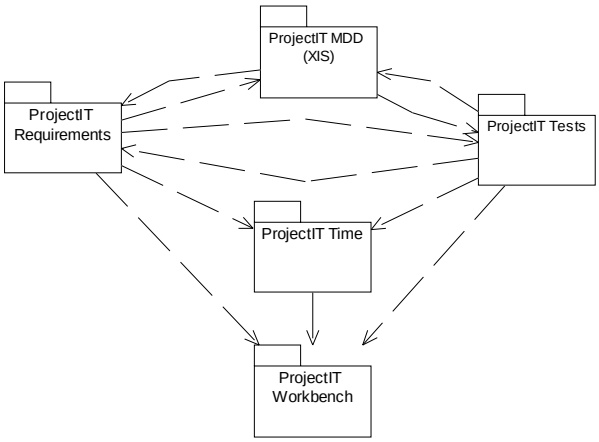

Fig. 1. ProjectIT functional architecture (extracted from [20]).

On the other hand, from an application perspective, ProjectIT involves two main tools: (1) the Web-based collaborative ProjectIT-Enterprise; and (2) the Windows-based ProjectIT-Studio tool (see Figure 2).

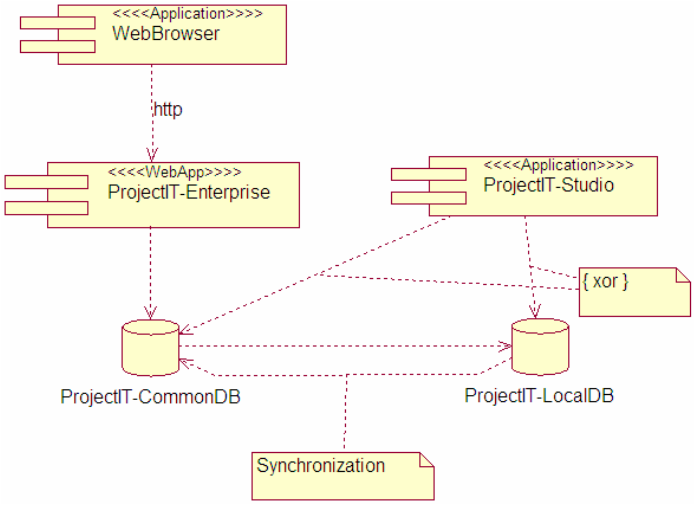

Fig. 2. Application architecture of ProjectIT (extracted from [20]).

ProjectIT-Studio's main objective is to provide designers and developers high productivity in tasks such as management and requirement specification, model design, automation code generation and software development. On the other hand, ProjectIT-Enterprise's goal is to deliver collaboration mechanisms to support development teams with considerable (medium and/or high) size, emphasizing tasks related to pro-

ject and document management, such as version control, and time, quality or risk control [20].

This paper focuses on ProjectIT-Studio, as it consists in the combination of multiple management and development tools. In theory, these tools are developed separately from each other, following a modular approach, still they all use Microsoft .NET technology [10]. This makes these tools prime candidates for integration.

3. The motivation for an integration platform

ProjectIT is a project of considerable size, with many tools being developed. Still, the various phases of information systems development are not isolated from each other. This means the tools developed should be able to interoperate with each other; otherwise, the sheer amount of trouble one would have when managing some of the stages of development would make even the most patient user turn away and run for it, for his sanity's sake…

As ProjectIT grew, it became clearer that its involved tools should present some degree of integration, modularity and interoperability amongst themselves. So, what was needed was an integration platform that provided the means for all these tools to communicate efficiently and also released the tool developer from most repetitive tasks, which usually lead to code duplication and, therefore, greater bug possibilities.

However, ProjectIT uses Microsoft's .NET Framework technology [10]. To the best of our knowledge, at the moment there was no such platform in the .NET environment. So, the decision was taken to try and bring such a platform, available in another environment, to the .NET world: the Eclipse Platform [3].

The Eclipse Platform is a well-known and stable platform, with wide-spread support in the Java community. Although commonly known to software developer community as an open-source IDE (Integrated Development Environment) for Java software development, Eclipse should be best described as "an open universal framework for building developer tools" [21].

It should be noted that the objective of the port of the Eclipse Platform is not to provide a new IDE for development in the .NET environment. The objective is to provide an integration platform for ProjectIT, which can (1) gather its tools; (2) present them to the user under a single front-end, instead of presenting multiple single tools; (3) promote better interoperability between these tools; and (4) provide building blocks for development of tools.

The Eclipse Platform's principal role is to supply tool providers with mechanisms to use, and rules to follow, that lead to seamlessly-integrated tools. These mechanisms are exposed via well-defined API interfaces, classes, and methods. The platform also provides useful building blocks and frameworks that facilitate developing new tools [21]. This level of integration is attained due to Eclipse's plug-in based architecture: all functionality is contained in plug-ins, which are controlled by the core Platform Runtime. The core Platform Runtime, however, cannot actually *do* anything that its user (i.e. the developer) will find useful.

This port of the Eclipse Platform has a good chance of succeeding, as most of its functionality is based on the application of design patterns, and there are not too many

dependencies on Java-specific mechanisms (among a few of them, one exception is the usage of Java's Class Loader mechanism, which is explained in [9]).

4. The Eclipse architecture

Eclipse's architecture is based on the following classic concepts: core platform runtime, plug-ins, and extension points.

4.1. Core Platform Runtime

The Platform Runtime is responsible for configuring the Platform, detecting available plug-ins, reading their manifest files and dynamically building a plug-in registry. The Platform Runtime also performs some validation tasks, such as detecting extensions to extension points that do not exist. After all these tasks have been completed, the Runtime makes the registry and the Platform's configuration available to plug-ins [21].

4.2. Plug-in

A plug-in (or a component) is the smallest unit of behaviour in the Eclipse Platform. Any tool can be developed as a plug-in or as a set of plug-ins. In fact, all of Eclipse's functionality is located in plug-ins, the exception to this being the functionality in the core Platform Runtime.

A plug-in also features a manifest file which contains information about the plug-in itself, such as extension points (explained next) that the plug-in provides, and what extensions the plug-in provides to extension points defined by other plug-ins. The reason this manifest file is located separately from the plug-in itself is so the Runtime can read the information about the plug-in without actually loading the plug-in into memory. This way, the Runtime can build the registry without loading every plug-in, saving resources [21].

Plug-ins can also depend on each other. These dependencies are expressed in the plug-in's manifest file. This way, a plug-in will only be used (and, of course, successfully loaded) if all other plug-ins on which it depends are also successfully loaded.

4.3. Extension point

An extension point is a way for plug-ins to interoperate with each other. This interconnection model is achieved through the classical process: (1) any plug-in can declare any number of extension points; (2) on the other hand, any plug-in can provide any number of extensions (also called "contributions" in the Eclipse community) for one or more extension points declared by other plug-ins. For example, the Workbench plug-in declares an extension point for user preferences. Any plug-in can contribute its own user preferences by defining extensions to this point.

An extension point can have a corresponding API interface, defined using another previous API or using a new API. Other plug-ins can provide extensions to an extension point by implementing the interface [21].

Figure 3 illustrates how plug-ins and extension points are related. Plug-in 1 declares three extension points (Extension point 1, Extension point 2 and Extension point 3). Extension point 1 and Extension point 2 receive the extension (or contribution) from Plug-in 2. On the other hand, Plug-in 2 also declares two other extension points (Extension point 4 and Extension point 5). These last two extension points may also receive contributions from other existing plug-ins. Plug-in 1 may also be contributing to other existing extension points, not represented in the figure.

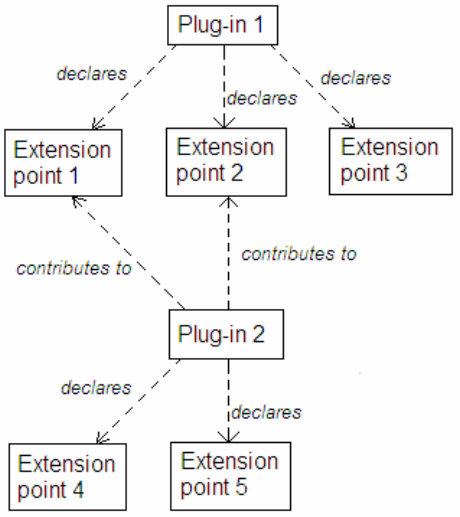

Fig. 3. Plug-ins can declare extension points, and can also contribute to already existing extension points, declared by other plug-ins.

4.4. Eclipse Platform's plug-in organization

The Eclipse Platform features several components (i.e. plug-ins), aside from the ones that developers can create. Figure 4 shows the major components in Eclipse. As an example of component usage, all Eclipse application that feature a graphical user interface will have dependencies to the WorkBench plug-in, which will also depend on the SWT (Standard Widget Toolkit) and on the JFace plug-ins. However, a plug-in with a text-only user interface will not need them [21].

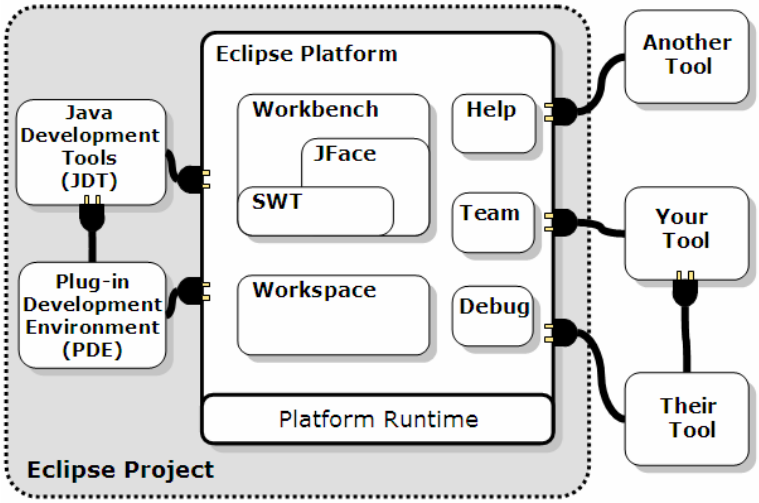

Fig. 4. Major components in the Eclipse Platform (extracted from [4]).

Figure 5 shows another view of the Eclipse Platform's organization, this one from a more layered point of view, and of the plug-ins used for providing Eclipse's most common uses. In this figure, note the Java Development Tools plug-in, which provides the Eclipse Platform with the ability to serve as a Java IDE.

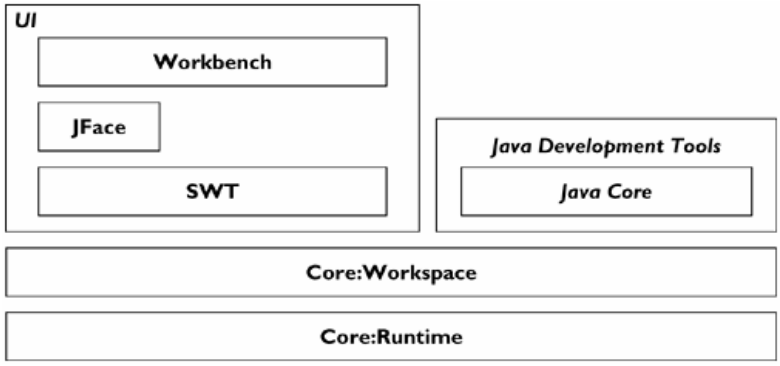

Fig. 5. The Eclipse Platform with some common plug-ins (extracted from [5]).

5. What tools are being used

The Eclipse Platform is available only for the Java platform, and the Platform Runtime is heavily based on the Java Class Loader mechanism. This mechanism is incompatible with .NET, which uses the Assembly mechanism [1], along with a Class Loader mechanism for fetching classes from within assemblies. Therefore, the port of

Eclipse to .NET has to deal with this difference between the Java platform and the .NET framework.

However, it is desirable that any Eclipse Platform code which does not use Java-specific mechanisms be automatically converted.

Both of these approaches have their advantages and disadvantages. A manual conversion makes the resulting code more efficient, as we can adapt the code to use the full potential of the .NET framework. On the other hand, this approach does bring the disadvantage of spending more time in the conversion. Also, manual conversion introduces a much greater chance of unwillingly introducing bugs in the code, as this conversion would be made by humans, who are error-prone. As for the automatic conversion, its advantages are the manual conversion's disadvantages, and vice-versa.

For this work, we have decided that the best approach would be a mix of the two approaches just presented: code that uses Java-specific mechanisms (or code that uses these mechanisms indirectly, through dependencies) will be manually converted, as the usage of these mechanisms will have to be adapted to the .NET framework. As for code that does not use such mechanisms, an automatic conversion is preferable, as there would be no reason (besides adapting the code to use the .NET framework to its full potential) for spending more time than necessary in its manual conversion.

Considering the approaches presented and their advantages and disadvantages, the following auxiliary tools are being used in this port of the Eclipse Platform: Microsoft's JLCA (Java Language Conversion Assistant) [11] and IKVM.NET [7].

However, it should be noted that the underlying architecture of the port of Eclipse will be just like the original Eclipse's architecture, which was presented in the previous section. The changes that must take place in the port are changes on the implementation, as the implementation must be adapted to the .NET framework.

5.1. Microsoft's JLCA (Java Language Conversion Assistant)

The JLCA tool is a free extension to Microsoft's Visual Studio .NET [12]. This tool allows the conversion of most Java source code, supplied by the user, to the C# language [13]. This tool has proven to be very useful in converting most of the large amount of Java code that constitutes the Eclipse Platform, and allows us to save a large amount of time in the conversion of Eclipse's code.

However, some time must still be spent in this manual conversion, as JLCA can not convert code that uses Java-specific mechanisms, such as the Java Class Loader mechanism, on which the Platform Runtime is based.

Figure 6 illustrates how the JLCA tool is used in the process of converting Eclipse.

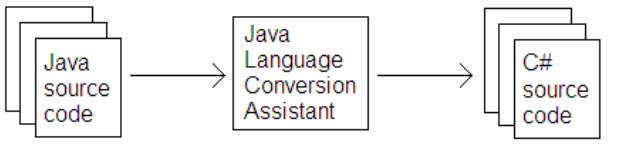

Fig. 6. The JLCA tool is used to convert Java source code to C# source code.

The JLCA tool will be used to convert most of the Eclipse source code, as nearly all plug-ins depend on the Platform Runtime and, therefore, on Java-specific mechanisms. One exception to this is the SWT component, which can be used in a stand-alone application (not necessarily Eclipse; there are other programs that use this component, such as Azureus [2]), and can be converted automatically, using IKVM.NET (shown next).

5.2. IKVM.NET

The IKVM.NET project aims to create a middle-ware platform that provides total interoperability between the Java and .NET platforms. To achieve this goal, it defines a middle tier, between the Java program and the .NET Framework. This tier receives calls to the Java virtual machine (VM) and invokes the .NET framework accordingly. In other words, it translates Java VM calls to .NET framework calls. Figure 7 illustrates how IKVM.NET operates and achieves total interoperability between Java programs and the .NET framework.

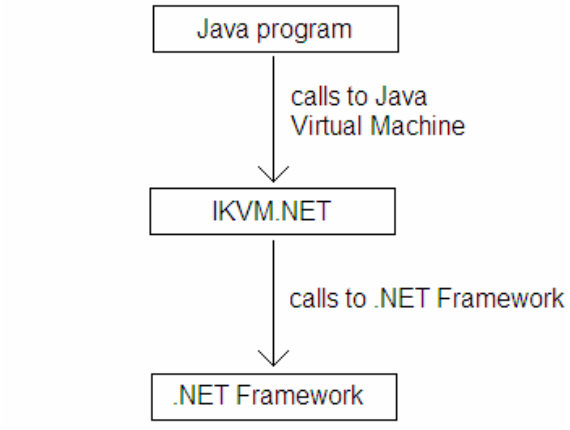

Fig. 7. IKVM.NET acts as Java Virtual Machine, but calls the .NET framework.

This project is in a quite advanced state, with numerous Java applications being correctly executed on top of the .NET framework. However, this tool is very useful for this work due to the library converter included with this tool. This converter takes a

Java class library (a .jar file) and returns a .NET class library (an .exe or .dll file, depending on whether the Java class library is executable or not, respectively).

The reader may be wondering, at this point, why not to simply use IKVM.NET together with Eclipse, instead of going through all this work. The answer to this question is: the objective of this effort is not to bring the Eclipse Platform to the .NET environment; the objective is to provide a platform, located at a higher level than the .NET framework, which provides a more efficient integration of the tools developed in the context of ProjectIT. The Eclipse Platform was chosen because of its advanced state of development, and because it has proven itself in the IT community [4].

However, the main disadvantage of Eclipse used in conjunction with this tool is the fact that, although the Java class files are converted to .NET libraries, they still expect to interact with other Java class files. This means that, should we use Eclipse with IKVM.NET, all plug-ins should be developed in Java. This would bring another added effort of converting all existing tools in ProjectIT-Studio to Java.

One of the few components that will be converted using this tool is SWT, as it does not depend on Java-specific mechanisms, nor does it depend on the Platform Runtime.

6. Other possible solutions

6.1. Using the Eclipse Platform and converting ProjectIT's tools to Java

Another possible solution for our integration problem would be to use the existing Java Eclipse Platform, and convert all our existing tools (in .NET code) to Java code, in the form of Eclipse plug-ins. This solution would certainly have its advantages: the Eclipse Platform is heavily tested and, therefore, very stable.

However, the tools that are in development for ProjectIT have also been tested. Even more important is the fact that, if we converted these tools to Java, each member of the team behind ProjectIT would have to adapt to this "new" language (by getting used to it, and by converting their tools to Java). This learning curve would not be so great, as the differences between Java and C# (ProjectIT's most commonly used language) are not significant. Nevertheless, ProjectIT's development effort would be held back for some time.

Given this scenario, the approach taken was to continue research and development of the ProjectIT tools and, in parallel to their development, create the port of the Eclipse Platform to the .NET Framework. This way, once the new Platform is stable and safe for deployment, the existing tools can be adapted to plug-ins with only a small effort.

6.2. Converting Java code to .NET versus a new implementation

The solution this article presents can raise some controversy, as this article has not yet presented whether it would be easier to simply create a new implementation of the Platform, based on the Eclipse Platform's design.

A new implementation would have its advantages, such as that the code would be originally created with the .NET Framework in mind, and would therefore be cleaner and would be based more heavily on the Framework. However, the sheer volume of the code of the Eclipse Platform would make this task a very long one. This fact, combined with the fact that a very significant part of the Eclipse Platform's code is the application of design patterns (some of which are described in [6]), does make a conversion of the existing Eclipse code much more appealing (and more time-efficient), as many best-practices are already present in it.

These facts, along with the existence of the JLCA and IKVM.NET tools (explained above), were the decisive factors in the decision to convert the existing Java code to .NET code.

7. Future work

When this port of Eclipse is completed, the creation of new tools (and the adaptation of already existing tools) will take place. For example, we will create a modeler for the ProjectIT/UML profile [17] [19] [20], supporting the Model Driven Development (MDD) based approach [16]. This modeler will provide input for another tool to be adapted, the ProjectIT Tool, an automatic code generator [18]. Another example of an already existing tool that will be adapted is the ProjectIT-RSL Language tool [22], which is currently a Microsoft Visual Studio .NET plug-in.

Figure 8 illustrates these future plans for the Eclipse.NET platform. The extension points (the mechanism through which the developed tools will communicate with each other) are not represent to maintain simplicity in the figure.

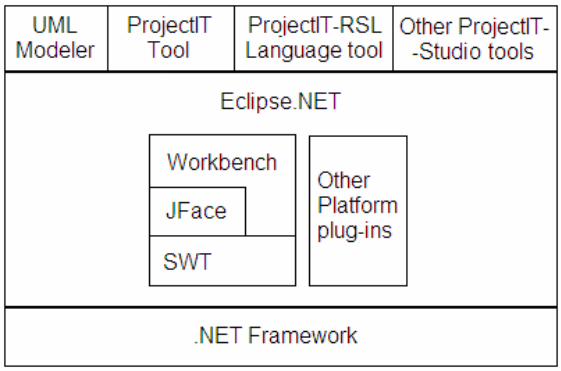

Fig. 8. ProjectIT-Studio tools will be adapted as plug-ins for the Eclipse.NET platform.

Orthogonally to the future conversion of these tools to Eclipse.NET plug-ins, the Eclipse.NET platform itself will continue evolving, by making some aspects better and by adding other aspects which are considered necessary.

However, this does not mean that Eclipse.NET will be completely synchronized with the original Eclipse Platform, as this type of synchronization would make Eclipse.NET a mere copy of Eclipse and would most likely force us to give up on exploring the .NET Framework to its fullest extent. We intend to include Eclipse's future important advancements and breakthroughs into Eclipse.NET, but we also plan to improve this new Platform with new ideas of our own. Also, best practices in the various fields of IT development (such as the best practices presented in [14], for building Windows applications) will be incorporated.

Hopefully, Eclipse.NET will make its contribution to the evolution of IT projects (and of information systems in general) in the .NET environment.

8. References

[1] Archer, T., *Inside C#*, Microsoft Press, 2001.
[2] Azureus web site: http://azureus.sourceforge.net
[3] Eclipse's web site: http://www.eclipse.org
[4] Eclipse Project's slide show presentation, http://www.eclipse.org/eclipse/presentation/eclipse-slides.pdf, 2003.
[5] Gamma, E., Beck, K., *Contributing to Eclipse: Principles, Patterns, and Plug-Ins*. Addison-Wesley, 2003.
[6] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
[7] IKVM.NET Project's web site: http://www.ikvm.net
[8] INESC-ID, Information Systems Group (GSI). ProjectIT web site: http://berlin.inesc-id.pt/alb/ProjectIT@81.aspx
[9] McGraw, G., Felten, E., *Securing Java*. John Wiley & Sons Inc., 1999. Web site: http://www.securingjava.com/chapter-two/chapter-two-7.html
[10] Microsoft .NET web site: http://www.microsoft.com/net
[11] Microsoft Java Language Conversion Assistant 3.0 beta's web site: http://msdn.microsoft.com/vstudio/downloads/tools/jlca/30beta/default.aspx
[12] Microsoft Visual Studio .NET Home web site: http://msdn.microsoft.com/vstudio
[13] Microsoft Visual C# Developer Center web site: http://msdn.microsoft.com/vcsharp
[14] Microsoft Windows Forms .NET web site: http://www.windowsforms.net
[15] Moreira, V., *ProjectPRO, Plataforma de Gestão de Requisitos, Relatório de Trabalho Final de Curso*. Instituto Superior Técnico (Portugal), July 2003.
[16] Model Driven Architecture web site: http://www.omg.org/mda.
[17] UML web site: http://www.uml.org
[18] Queiroga, R. B., *The XIS Case Tool, Relatório de Trabalho Final de Curso*. Instituto Superior Técnico (Portugal), October 2004.
[19] Silva, A., Videira, C., UML, Metodologias e Ferramentas CASE. Centro Atlântico (Portugal), Lisbon, 2001.
[20] Silva, A. R., *O Programa de Investigação ProjectIT*. INESC-ID & IST (Portugal), October 2004. Web site: http://berlin.inesc-id.pt/alb/uploads/1/193/pit-white-paper-v1.0.pdf
[21] Shah, R., *Working the Eclipse Platform*, http://www-106.ibm.com/developerworks/opensource/library/os-eclipse, March 2003

[22] Videira, C., Carmo, J. L., Silva, A. R., *The ProjectIT-RSL Language Overview*. UML Modeling Languages and Applications: UML 2004 Satellite Activities, Lisbon, 2004.