# The ProjectIT-Studio UMLModeler: A tool for the design and transformation of UML models

João de Sousa Saraiva [1], Alberto Rodrigues da Silva [1]

**joao.saraiva@inesc-id.pt, alberto.silva@acm.org**

[1] INESC-ID/Instituto Superior Técnico, Rua Alves Redol, 9, 1000-029 Lisboa, Portugal

**Abstract:** The emerging MDE paradigm requires modeling tools, such as UML-supporting tools. However, many current UML modeling tools still do not address some issues, such as defining and applying UML profiles, or the integration with requirements specification tools and code-generators. ProjectIT is a collaborative research project which aims to provide a software development workbench, with support for activities such as requirements engineering, analysis, design, and code generation. This is achieved through the implementation of a plugin-based CASE tool, called ProjectIT-Studio.
This paper presents ProjectIT-Studio/UMLModeler, a UML 2.0 visual modeling plugin for ProjectIT-Studio with features such as easy profile definition, "model-to-model" transformations, and integration with the requirements specification and source-code generation tools of ProjectIT-Studio.

**Keywords**: ProjectIT; MDE; UML; design; transformation.

## 1. Introduction

Ever since the appearance of computers, researchers have been trying to raise the abstraction level at which software developers write computer programs. Currently, this abstraction level is being raised into the model-driven engineering (MDE) approach (Schmidt, 2006), in which models are considered first-class entities and become the backbone of the entire MDE-oriented software development process. Other important artifacts, such as code and documentation, can be produced automatically from those models, relieving developers from issues such as underlying platform complexity or inability of third-generation languages to express domain concepts (Schmidt, 2006).

MDE is not a new idea. Already in the 1980s and 1990s, CASE tools were focused on supplying developers with methods and tools to express software systems using graphical general-purpose language representations. The developer would then be able to perform different tasks over those representations, such as correction

analysis or transformations to/from code. However, these CASE tools failed, due to issues such as those pointed out in (Schmidt, 2006): (1) poor mapping of general-purpose languages into underlying platforms; and (2) code was still the first-class entity in the development process, while models were used only as documentation. Currently, there are better conditions for such modeling tools to appear. Software systems are reaching such a high degree of complexity that third-generation languages aren't sufficient anymore; another abstraction level is needed. This need, combined with the choices of middleware technology currently available, to which models can be easily mapped, is the motivation for the adoption of MDE.

From the developer's point of view, a key issue for acceptance of any approach is good tool support, so that software programs can be created in an easy and efficient manner. There is a wide variety of modeling tools available today, covering most modeling standards and approaches in existence today, such as Poseidon (Gentleware, n.d.), or Enterprise Architect (SparxSystems, n.d.), from a long list of tools that support UML (UML, n.d.) modeling. However, most of these UML modeling tools do not address a number of issues essential to MDE, such as: (1) integration with requirements specification tools; (2) flexible and easy specification of "model-to-model" or "model-to-code" transformations; (3) easy definition and application of UML profiles; and (4) easy maintenance of model consistency according to the semantics defined in UML and in the UML profiles applied to the model.

This paper presents **ProjectIT-Studio/UMLModeler**, a UML modeling tool which tries to address the major issues referred above. This paper is structured as follows. Section 2 overviews the ProjectIT initiative and the ProjectIT-Studio CASE tool. Section 3 presents related work. Section 4 describes the ProjectIT-Studio UMLModeler tool. Section 5 concludes this paper.

## 2. ProjectIT and ProjectIT-Studio

Software development is a complex activity, involving issues in areas such as technology and human resources. Despite the efforts made to overcome those issues, they are still frequently found in software development projects, with negative consequences in terms of project scope, time, budget and quality. One of the causes for this situation is the fact that many projects do not follow a structured, standard and systematic approach, like the methodologies and best practices proposed by the Software Engineering community.

It was in this context that an initiative in the area of requirements engineering, project management, and model-driven development, named **ProjectIT** (Silva, 2004), was started. Its main goal is to minimize the negative consequences mentioned above, by researching new approaches to accelerate the underlying development process by automating its repetitive activities, thus globally improving the quality and productivity of the development process. So far, this research project

has produced the following results: (1) the "eXtreme modeling Interactive Systems" (XIS) UML profile, currently in its second version (Silva et al., 2007b); (2) a new requirements specification language, called ProjectIT-RSL; (3) the ProjectIT approach, a software development approach that essentially follows the Model-Driven Architecture (MDA) (OMG, n.d.a) philosophy combined with the use of ProjectIT-RSL; and (4) a CASE tool with a modular plugin-based architecture, called ProjectIT-Studio (Silva et al., 2006), covering the activities of the software development life cycle, from requirements specification to artifact generation through generative programming techniques.

The ProjectIT-Studio tool supports the ProjectIT approach, and consists of an application based on Eclipse.NET (SourceForge.net: Eclipse.NET, n.d.), which features a modular plugin-based architecture and runs over the .NET Framework. ProjectIT-Studio can also be seen as an orchestration of three different plugins (see Figure 1): ProjectIT-Studio/Requirements, ProjectIT-Studio/UMLModeler, and ProjectIT-Studio/MDDGenerator.

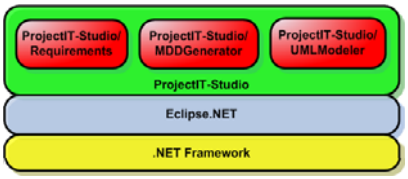


Figure 1 – The plugin architecture of ProjectIT-Studio.

**ProjectIT-Studio/Requirements** (Requirements for short) is responsible for requirements specification, and provides a requirements text editor that supports typical IDE features, such as on-the-fly syntactic verification, syntax highlighting, and auto-complete. **ProjectIT-Studio/MDDGenerator** (MDDGenerator for short) is responsible for the generation of system artifacts based on models; it establishes a mapping between a system model and a software platform, and generates the corresponding source-code and documentation for the specified platform, by using a template-based generative approach. Finally, **ProjectIT-Studio/UMLModeler** (UMLModeler for short) (Saraiva, 2006), presented in this paper, is responsible for standard UML modeling; the Designer creates models based on a given UML profile, which are then used by MDDGenerator to create the system artifacts according to specific software architectures.

## 3. Related Work

ProjectIT-Studio/UMLModeler is the result of only one of several research initiatives in the area of MDE. This section presents a brief overview of some related work in the area of UML modeling and model transformation tools.

### 3.1. UML modeling tools

To determine the set of relevant features to include in UMLModeler, some popular UML modeling tools were analyzed, which are now referred. Although there are many other tools available, these tools were analyzed because we consider that they constitute a good representation of the current status of this area.

*ArgoUML 0.24* (ArgoUML, n.d.) is an open-source UML modeling tool, which currently only supports UML 1.4. Although the tool does not other many of the typical functionalities of UML modeling tools, it differentiates itself due to its use of "cognitive psychology" (ArgoUML, n.d.) to detect model inconsistencies and promote modeling best-practices. It supports import/export to XMI, as its models are stored natively in that format; however, it stores graphical information separately from the model. Another interesting feature is that it enforces user-defined OCL constraints, although only for Classes and Features. The tool offers code generation capabilities, but no model transformations to support MDA.

*Enterprise Architect 7.0* (SparxSystems, n.d.) is a commercial tool that is easy to use and supports UML 2.0, offering a range of functionalities that considerably accelerate modeling tasks. It also supports the UML profile mechanism, and it makes the definition of a profile an easy task. However, profile definition is limited to defining stereotypes and what metaclasses they extend, without any structuring possibility as the Profile Packages provide. Constraints entered in the profile definition (using OCL) are not enforced; the only enforced validation is the application of a stereotype to an instance of a metaclass. EA supports import/export to XMI and diagrams can be exchanged between different instances of EA. EA offers code generation capabilities and some MDA-oriented PIM-to-PSM model transformation templates, and users can define model transformation templates by using the same template constructs as for code generation.

*Poseidon for UML 6.0* (Gentleware, n.d.) is a commercial UML modeling tool based on ArgoUML, but supporting UML 2.0. It supports import/export to XMI, because its models are stored in this format; diagrams are also stored in XMI, as the tool supports UML 2.0 and its Diagram Interchange specification. The tool allows users to specify model constraints using OCL 2.0, but does not enforce those constraints, unlike ArgoUML. It offers code generation capabilities, but no model transformations to support MDA. Although the tool does support the UML Profile, it does not allow typical users to define their own profiles, and the available profiles are provided by code generation plugins.

*Rational Rose 2003* (IBM, n.d.) is a commercial UML modeling tool which supports UML 1.5. It features a wide range of functionalities that promote modeling good-practices and accelerate the modeling tasks. Although it includes UML's standard stereotypes, it does not provide an easy way to add user-defined stereotypes, and there is no support for the definition of a profile within the tool. The tool does not natively support import/export to XMI, although a plugin by UniSys adds

this functionality. The tool offers code generation capabilities, but no model transformations to support MDA. It was recently replaced by Rational Software Modeler, in the scope of IBM's Rational Software Development Platform.

### 4.2. Model transformation languages and tools

The area of model transformation languages and tools is currently the target of multiple research initiatives, some of them discussed in this subsection.

*QVT (Queries/Views/Transformations)* (OMG, 2005) is a standard for performing model transformations, defined by the OMG. This standard provides a framework to transform source models into target models, where both types of models conform to any MOF 2.0 metamodel. Currently, QVT only addresses "model-to-model" transformations. There are some implementations available, such as SmartQVT (SmartQVT, n.d.) or MTF (alphaWorks MTF, n.d.), although it most available implementations address only parts of the QVT specification.

*ATL (ATLAS Transformation Language)* (ATL, n.d.) is a model transformation language developed at INRIA to answer the QVT Request For Proposal. It is implemented as an open-source Eclipse plugin, the ATL Integrated Environment, with a number of standard development tools (such as syntax highlighting and a debugger) that aim to ease development of ATL transformations.

*VIATRA (VIsual Automated model TRAnsformations)* framework (VIATRA2, n.d.) is the core of a transformation-based verification and validation environment for improving the quality of systems designed using UML, by automatically checking consistency, completeness, and dependability requirements (VIATRA2, n.d.). These verifications are done by using invisible formal methods to support the designing of model transformations; these formal methods are hidden by automated model transformations, allowing VIATRA to complement other approaches.

*SiTra (Simple Transformer)* (SiTra, n.d.) is a simple library designed to provide a pragmatic approach to implementing model transformations. In this approach, transformations are defined by using a standard programming language (e.g., Java, C#) to write transformation rules, supported by two interfaces. Transformations can then be applied to a model simply by applying all defined transformation rules on the model; these rule applications are done by a class provided by SiTra.

## 4. ProjectIT-Studio UMLModeler

The **ProjectIT-Studio/UMLModeler** provides a UML modeling environment to support the ProjectIT approach, as well as other generic approaches. Beyond the common UML modeling features, the Designer can create and refine UML models by using features such as: (1) UML Profile support; (2) "model-to-model" transfor-

mations; and (3) integration with the ProjectIT-Studio plugins for requirements specification and source-code generation. This section presents these features.

### 4.1. UML Profile support

An aspect that differentiates UMLModeler from typical UML modeling tools is its support for the graphical definition and application of an UML profile, inspired on Enterprise Architect (SparxSystems, n.d.). Support for UML profiles was one of the main concerns when developing UMLModeler, because although most tools also support it, it is often complicated to define and/or use a Profile in those tools.

#### Defining a UML Profile

The process of defining a profile in UMLModeler involves only the creation of a profile, profile packages (which allow the segmentation of the contents of a profile into smaller packages), and stereotypes. To improve the reusability of profiles, it is typically a good idea to define each profile in a separate "blank" UML model file, and at a later time apply the profile to another UML model file.

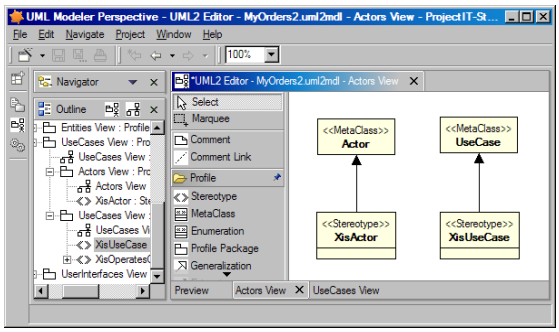


Figure 2 – Some stereotypes defined in UMLModeler.

After creating the profile, the user can add elements (such as diagrams, profile packages, or stereotypes) to it. A profile package acts only as a container of stereotypes and other profile packages, and does not carry any particular semantics. To add stereotypes to the profile, the user needs to create the stereotype, add the relevant attributes in the stereotype's Property Form, specify inheritance relationships between stereotypes, and specify the metaclass(es) that the stereotype extends. The user can also specify a set of images for each stereotype. Figure 2 presents a screenshot with two stereotypes, *XisActor* and *XisUseCase*, extending the UML metaclasses *Actor* and *UseCase* respectively.

Profiles, diagrams, and all other modeled UML elements are stored in a XML document (known as a "UML model file"), which allows other tools to create and/or manipulate UML models to be used in ProjectIT-Studio. Figure 3 presents an example of such a file (for simplicity, most of the details have been removed). Another advantage of this mechanism is that profiles (as well as "regular" models) can be easily exchanged between different instances of ProjectIT-Studio, simply by copying the file that contains the profile's definition.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <Root version="2">
3    <Profiles>
4      <Profile id="..." name="XIS2" visibility="
           Public">
5        <OwnedMembers>
6          <ProfilePackage id="..." name="UseCases
               View" visibility="Public">
7            <OwnedMembers>
8              <ProfilePackage id="..." name="
                   Actors View" visibility="Public
                   ">
9                <OwnedMembers>
10                 <Stereotype id="..." isAbstract
                       ="False" name="XisActor"
                       visibility="Public" isLeaf
                       ="False">
11                   <MetaclassExtensions>
12                     <Extends id="..." metaClass
                           ="..." metaClassType="
                           Actor" />
13                   </MetaclassExtensions>
14                   <OwnedAttributes />
15                 </Stereotype>
16               </OwnedMembers>
17             </ProfilePackage>
18           </OwnedMembers>
19         </ProfilePackage>
20       </OwnedMembers>
21     </Profile>
22   </Profiles>
23   ...
24 </Root>
```

Figure 3 – Example of a UML profile serialized as XML.

## Applying a UML Profile and Stereotypes

After applying the profile to the model, the user can then apply stereotypes to UML elements in diagrams. The tool only allows the application of a stereotype to an UML element if that stereotype extends that element's metaclass; otherwise, the stereotype application will not be allowed, and the user will be notified of this through visual feedback. Figure 4 presents a screenshot of a diagram with some UML Actors to which the *XisActor* stereotype (shown in Figure 2) has been applied.

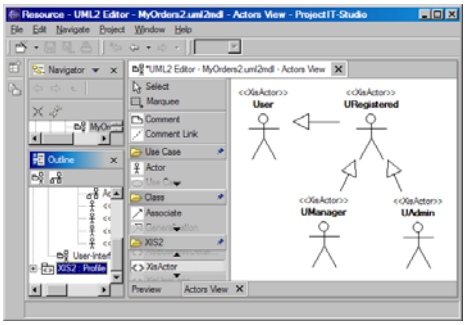


Figure 4 – Screenshot of a diagram with stereotypes applied to UML Actors.

Stereotypes can be applied to elements by one of two ways: (1) by selecting the stereotype in the toolbox and clicking on the target element; or (2) by using the element's Property Form. Although the first way is meant to significantly accelerate the (frequent) task of applying stereotypes to elements, the second way allows the application of stereotypes, editing of tagged values, or removal of stereotype applications from the current element.

A difference between UMLModeler and other UML modeling tools is the fact that UMLModeler allows the application of multiple profiles to a UML model, enabling MDA-oriented (OMG, n.d.a) scenarios in which a software designer creates a generic model and annotates it using various UML profiles. A typical example would be: (1) a software engineer creates a PIM model using UML; (2) for each target language (e.g., Java, C#, C++), a corresponding profile is applied to the model; and (3) for each of the desired target languages, PSM models are obtained by using PIM-to-PSM transformations that recognize those UML profiles.

### 4.2. Support for "model-to-model" transformations

Another aspect that differentiates UMLModeler from other tools is its support for *model manipulation by external plugins*. This mechanism allows developers to easily provide transformation operations, which can process and manipulate UML models for any number of purposes (e.g., the generation of the UserInterfaces View, in the case of XIS2 (Silva et al., 2007b)). It also allows developers to provide *models validation*, which contributes to avoid potential errors in transformations (because they only become available to the user after certain conditions are met by the UML model). It should be noted that this mechanism can be used by any plugins, for any type of model interpretation and/or manipulation purposes.

These transformations are currently implemented through C# code which receives a model, parses the model, and possibly alters the model. Figure 5 presents an overview of such a transformation (most of the source-code has been removed for text simplicity). Any changes to the model will be immediately reflected in the UMLModeler views, as the Observer pattern (Gamma et al., 1995) is used to allow the tool to be notified of any changes that occur to the model. Any "model-to-model" transformations that can be applied to the model may be invoked by the user, through the appropriate item from the model's context menu.

### 4.3. Integration with Requirements and MDDGenerator

UMLModeler also stands apart from other modeling tools because it is well integrated with the other ProjectIT-Studio plugins, allowing users to go from requirements specification to source-code generation without requiring several different tools. This integration, illustrated in Figure 6, is done by using the import and extension point mechanisms provided by Eclipse.NET, and enables UMLModeler to

receive models from an external source (in this case, Requirements) and to provide models to external plugins which can process such models (in this case, MDDGenerator). Further details about this integration can be found at (Silva et al., 2006).

```
1  public class PIM_To_PSM_Transformation {
2    public static bool ValidateTransformation(
         ModelContents model) { ... }
3
4    public static void TransformModel(
         ModelContents umlModel) {
5      // Perform validation, just in case
6      if(!ValidateTransformation(umlModel)) {
7        return;
8      }
9
10     // Process and manipulate umlModel
11   }
12 }
```

Figure 5 – Example of a "model-to-model" transformation.

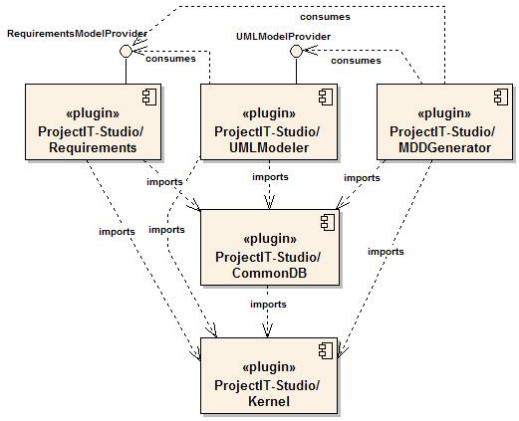


Figure 6 – Integration between ProjectIT-Studio's plugins.

### 4.4. Comparison with other tools

One of the concerns during the development of UMLModeler was to provide an intuitive interface for creating UML models and profiles. Enterprise Architect (EA) features an easy-to-use UI, and so UMLModeler's UI was inspired on Enterprise Architect's. However, we also identified some usability issues in EA when dealing with profiles, such as: (1) the need to export a profile to XML and subsequently import it again in order to apply the profile; (2) the editing of the tagged values in stereotype applications; or (3) stereotypes could not be organized in packages. This

prompted us to define a different approach to dealing with UML profiles from within the tool. Table 1 presents a comparison between UMLModeler and the other UML modeling tools previously described in Section 3.

Table 1 – Comparison between UMLModeler and other UML modeling tools.

| | ArgoUML 0.24 | Enterprise Architect 7.0 | Poseidon for UML 6.0 | Rational Rose 2003 | UMLModeler |
|---|---|---|---|---|---|
| Supports UML 2.0 | No | Yes | Yes | No | Yes |
| Supports "model-to-model" transformations | No | Yes | No | No | Yes |
| Users can add their own transformations | --- | Yes | --- | --- | Yes |
| Supports code-generation | Yes | Yes | Yes | Yes | Yes |
| Users can define their own profiles | No | Yes | No | No (tool does not provide support) | Yes |
| Users can apply multiple profiles | --- | Yes | --- | --- | Yes |
| Stereotypes can be structured in "profile packages" | --- | No | --- | --- | Yes |
| Supports import/export to XMI | Yes | Yes | Yes | No (only by UniSys plugin) | No (to be added in the future) |
| Supports OCL | Yes | No | No | No | No (to be added in the future) |
| Supports Model Patterns | No | Yes | No | Yes (users cannot define their own patterns) | No (to be added in the future) |

Additionally, UMLModeler provides a flexible framework for defining “model-to-model” transformations (although it does not provide transformation validation mechanisms such as those in VIATRA). This framework is conceptually similar to SiTra, as UMLModeler provides a set of classes and interfaces to allow the manipulation of an UML model. However, UMLModeler does not provide base interfaces for defining “model-to-model” transformation rules as SiTra does; instead, “model-to-model” transformations are defined as methods which receive an UML model. Although we plan to provide such base interfaces in the future as a way to accelerate the definition of transformation rules, we will still provide this “low-level” mechanism because it does not impose any restrictions on the definition of ”model-to-model” transformation rules, allowing maximum flexibility.

### 4.5. Results achieved

UMLModeler has been used in the context of ProjectIT-Studio, in the moderately-complex MyOrders2 case study (Silva et al, 2007a). Because of its profile definition features and its integration with the ProjectIT-Studio requirements and code-generation tools, UMLModeler was able to address all the issues that arose from that case study (e.g., definition of the XIS2 profile within the tool, receiving the prototype model from the requirements tools, refining the model, and supplying the final model to the code-generation tool).

## 5. Conclusions and Future Work

This paper presented the current status of the UMLModeler, a plugin for ProjectIT-Studio that supports UML modeling in the context of the ProjectIT approach. This plugin provides a graphical UML diagram editor and an easy-to-extend framework that allows the editing of any aspect of an UML model. It also offers a range of features that can usually be found in UML modeling tools, and many more features are expected in future versions. Some of UMLModeler's innovative features relatively to the common ground of UML modeling tools, currently available, were also presented, such as its profile definition mechanism and the support for model manipulation by other plugins. Although these features will be improved in the future, they already differentiate UMLModeler.

As for future work, one of the issues to be considered is support for constraint specification (preferably using a standard language such as OCL, and the continuous validation of models based on those constraints. The mechanism for specifying "model-to-model" transformations could also be improved, by providing a set of "model-to-model" transformation primitives (instead of using source-code to define those transformations), or by following an approach more similar to SiTra, which provides some base interfaces to define transformation rules. We also intend to provide a set of tools to facilitate the specification of "model-to-model" transformations by using "low-level" source code, QVT, template-based mechanisms, and/or other transformation languages and approaches that may appear during the course of the ProjectIT research program. Finally, "model patterns" are a typically useful functionality which will be addressed in the future.

## References


alphaWorks MTF (n.d.). Retrieved 04/09/2007 from http://www.alphaworks.ibm.com/tech/mtf

ArgoUML (n.d.). Retrieved 05/06/2006 from http://argouml.tigris.org

ATL (n.d.). Retrieved 04/09/2007 from http://www.eclipse.org/m2m/atl/

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Gentleware: Poseidon for UML (n.d.). Retrieved 17/12/2006 from http://www.gentleware.com/products.html

IBM (n.d.). IBM Software – Rational Rose Data Modeler – Product Overview. Retrieved 05/06/2006 from http://www-306.ibm.com/software/awdtools/developer/datamodeler/

OMG (n.d.). Model Driven Architecture. Retrieved 05/06/2006 from http://www.omg.org/mda

OMG (n.d.). UML. Retrieved 05/06/2006 from http://www.uml.org

OMG (2005). Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Retrieved 22/06/2006 from http://www.omg.org/cgi-bin/apps/ doc?ptc/05-11-01.pdf

Saraiva, J. P. P. M. d. S. (2006). The UML Modeling Tool of ProjectIT-Studio. Master's thesis, Instituto Superior Técnico, Portugal.

Schmidt, D. C. (2006). Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31.

Silva, A. R. da, Saraiva, J., Ferreira, D., Silva, R., and Videira, C. (2007a). Integration of RE and MDE Paradigms: The ProjectIT Approach and Tools. In *IET Software (Special Issue): On the Interplay of .NET and Contemporary Software Engineering Techniques*, 1(6):294-314, Institution of Engineering and Technology.

Silva, A., Saraiva, J., Silva, R., and Martins, C. (2007b). XIS – UML Profile for eXtreme Modeling Interactive Systems. In *Fourth International Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2007)*, pages 55–66, Los Alamitos, CA, USA. IEEE Computer Society.

Silva, A., Videira, C., Saraiva, J., Ferreira, D., and Silva, R. (2006). The ProjectIT-Studio, an integrated environment for the development of information systems. In *Proceedings of the Second International Conference of Innovative Views of .NET Technologies (IVNET'06)*, pages 85–103. Sociedade Brasileira de Computação and Microsoft.

Silva, A. R. d. (2004). The ProjectIT Research Program (whitepaper). Retrieved 05/06/2006 from http://isg.inesc-id.pt/alb/uploads/1/193/pit-white-paper-v1.0.pdf

SiTra (n.d.). Retrieved 25/02/2008 from http://www.cs.bham.ac.uk/~bxb/ SiTra.html

SmartQVT (n.d.). Retrieved 25/02/2008 from http://smartqvt.elibel.tm.fr/

SourceForge.net: Eclipse.NET (n.d.). Retrieved 25/02/2008 from http://sourceforge.net/projects/eclipsedotnet

SparxSystems (n.d.). Enterprise Architect. Retrieved 05/06/2006 from http://www.sparxsystems.com/products/ea.html

VIATRA2 (n.d.). Retrieved 04/09/2007 from http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/subprojects/VIATRA2/