An overview of ProjectIT-RSL metamodel and prototype

Carlos Videira
INESC-ID, UAL, Lisboa, Portugal
cvideira@acm.org

Alberto Rodrigues da Silva
INESC-ID, IST, Lisboa, Portugal
alberto.silva@acm.org

Abstract

Software requirements engineering is an essential activity for the successful development of information systems. The techniques used to specify software requirements range from those that use formal notations to those based upon the use of natural language. In this paper we describe the metamodel of a new controlled natural language for requirements specification, called ProjectIT-RSL, which was defined after the identification of the most common linguistic patterns used in requirements documents written in natural language. The initial version of the language was implemented in a prototype built using the Visual Studio .NET development environment, which enabled the development of a set of tools to validate the syntactic and semantic rules of requirements texts written in the ProjectIT-RSL language.[1]

Keywords: Requirements; Requirements Specification Languages; Controlled Natural Languages; Metamodel; Patterns

1. Introduction

The development of information systems is a complex process usually initiated with the identification and specification of the requirements of the system to be developed, and in particular of its software components. The requirements concept is one of those IS/IT concepts where there is no standard and widely accepted definition. This is a result of many views from different people that are somehow interested in the development of information systems. Words such as “needs”, “features” or “functionalities” are frequently used as synonyms. A classical definition from Kotonya says that a “requirement is a statement about a system service or constraint” [Kotonya 1998]. Several surveys and studies, such as The Chaos Report, published by the Standish Group (http://www.standishgroup.com), have emphasized the problems that can result from mistakes in the early phases of system development, which are a consequence of inadequate, inconsistent, incomplete, or ambiguous requirements.

Software requirements are normally described using a requirements specification language, with different levels of formality. Formal approaches are based upon concepts from the disciplines of logic and mathematics and use a formal notation to specify the requirements; they define a language with a formal syntax and semantics, with a high level of abstraction. Some examples of formal approaches include VDM [Jones 1990] and Z [Spivey 1992]. Informal approaches are normally supported by some form of natural language or modeling techniques

[1] The work presented in this paper is partially funded by FCT (Portuguese Research and Technology Foundation), project POSI/EIA/57642/2004 (*Requirements engineering and model-based approaches in the ProjectIT research program*).

that can be understood by the end-user. The use of modeling techniques already implies some type of semi-formal semantics: the graphical elements of the notation used have a defined meaning, which is normally stated in natural language. The most well-known and successful example of these modeling languages is UML [Rumbaugh 2004].

Requirements must reflect the user's needs and support the communication between customers and software developers. Here is one of the most difficult challenges in the software development process: the communication gap between customers and information systems people. The ideal solution should combine the benefits of simultaneously using natural language and a precise (not necessarily formal) approach for requirements specification, which leads to the idea of using a controlled natural language. A controlled natural language is a subset of the words used in our common natural language, where the selected terms have their usual meaning, but can also be interpreted in specialized contexts (normally using tools).

As a result of the experience gathered from previous research and practical projects, the Information Systems Group of INESC-ID [http://gsi.inesc-id.pt/], in Lisbon, Portugal, started an initiative in the area of requirements engineering, named ProjectIT-Requirements [Videira 2004b], which proposes a new approach to successfully achieve some of the goals of this discipline. One of the results of this project is a new requirements specification language, called ProjectIT-RSL [Videira 2004a]. This paper presents an overview of the metamodel of the language, and describes how we have implemented a small prototype to validate the ideas of the language. It is organized in the following sections: section 2 presents an overview of the ProjectIT research program; section 3 presents an overview of the current ProjectIT-RSL metamodel; section 4 describes the basic features of the prototype we have developed; section 5 presents related work of previous initiatives; finally, section 6 justifies our perception that this proposal has some innovative contributions and presents the work to be performed in the future.

2. The ProjectIT initiative

The Information Systems Group of INESC-ID has been involved for some time in the development of research projects in the area of software engineering and the software development process, applying the results achieved in daily projects. Among others, it is worth mentioning XIS [Silva 2003], whose main contribution was the development of the XIS/UML profile intended to specify, simulate and develop information systems following a MDD approach, and ProjectPro [Moreira 2003], a Web based collaborative system to support the definition of basic concepts (projects, releases, sub-systems, requirements) and their relationships. Recently we decided to integrate both projects in a single project, which we have called "ProjectIT" [Silva 2004].

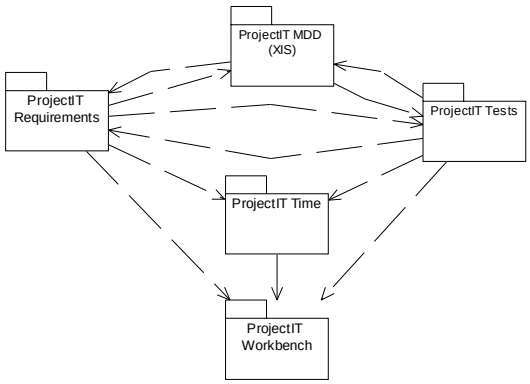

Figure 1 – ProjectIT functional architecture

The goal is to provide a complete software development workbench, with support for project management, requirements engineering, analysis, design and code generation activities. The architecture of this research program is shown in Figure 1. After having defined the goals of the project [Videira 2004c], we identified its main components: the **Requirements Specification Language**, which we have called **ProjectIT-RSL**, and a basic set of tools needed to reach the goals of the project, which are the **Requirements Editor** and the **Requirements Compiler**. Our vision is to build a tool for writing "requirements documents", like a word processor, and as we write, it will warn us of errors violating the requirements language and grammar rules we have defined.

3. ProjectIT-Requirements Specification Language Metamodel

To define ProjectIT-RSL we followed a simple approach. First, we analyzed the format and structure of requirements of the projects we have developed, which are mainly interactive systems. Then, we identified a common set of patterns for the requirements of this type of systems, and finally derived a metamodel of requirements that can adequately represent the patterns identified. ProjectIT-RSL is currently a language represented by a metamodel and a grammar that defines the rules to map the metamodel's concepts into sentences, which are validated by the requirements tools. Requirements are expressed by natural language sentences that have a subject, a verb and other words to complement their meaning. We mapped this natural language into a conceptual language saying that actors carry out operations, which can access one or more entities or entity's properties. For example, we identified the following common patterns [Videira 2005]:

```
<Requirement>: <Simple Requirement> | <Conditional Requirement>
<Simple Requirement>: <Subject> <Operation> <Object>
<Conditional Requirement>: <Condition> <Requirement>
```

These simple patterns enable writing requirements sentences such as:

```
The system manages products.
The user creates orders.
The user enters the invoice date.
The system automatically generates the invoice number.
If the invoice total is greater than 1000€, then the invoice discount equals 10%.
```

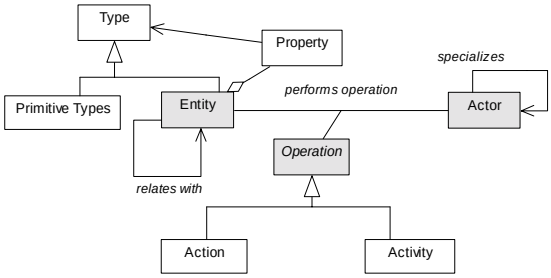

Figure 2 – ProjectIT-RSL metamodel – basic concepts

This analysis led to the identification of the main concepts of our language, shown in Figure 2:

1. **Actors** are active resources (e.g., an external system or an end-user role) that perform operations involving one or more entities.
2. **Entities** are static resources affected by operations (e.g., a client or an invoice). Entities have **Properties** that represent and describe their state.
3. **Operations** are described by their respective workflows, which consist of a sequence of simpler Operations that affect Entities and their Properties. Operations are specialized in **Actions**, which are atomic and primitive (and provided by default by our framework), and **Activities**, which are not atomic.

ProjectIT-RSL Metamodel – Organizational Concepts

The metamodel includes other concepts whose goal is to define the organization of the requirements, and which are represented in Figure 3. All requirements information is expressed in terms of Sentences (which represents every sentence of our text), which can be a Requirement, a Description or a Declaration. These statements are included in requirements documents, which we call Requirements Modules that are specialized in:

1. **Packages** that allow the specification of information about basic concepts (Actors, Entities, Operations, and Requirements) that can be reused in other requirements documents.
2. A **System**, which represents a software component (either a complete application or a reusable software component) with which an actor interacts, by executing operations to access or manage entities and their properties.

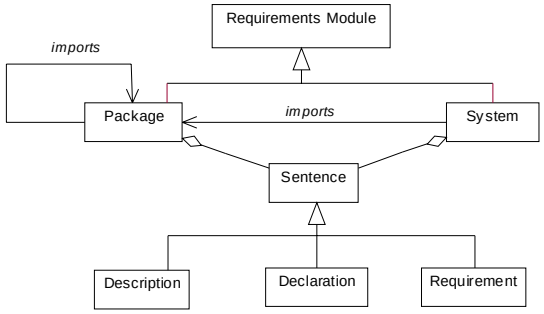

Figure 3 – ProjectIT-RSL metamodel – organizational concepts

Requirements describe what the system is expected to do, and are represented by a number of concepts, as Figure 4 shows. Requirements are currently specialized in:

1. **Functional Requirements** are the current focus of our research and specify the functional features of the system; these requirements are themselves specialized into (1) **Simple Requirements**, which is basically a relationship between an Actor, an Operation and an Entity or an Entity's Property, and (2) **Conditional Requirements**, which add a condition to a Simple Requirement.
2. **Non-Functional Requirements** (for example, for timing restrictions)
3. **Other Requirements** (for example, to include information about the need to use a specific database management system, or other design requirements imposed by the client; although these are generally classified as non-functional requirements, in our approach we prefer to distinguish them, due to the different support for generative programming techniques).

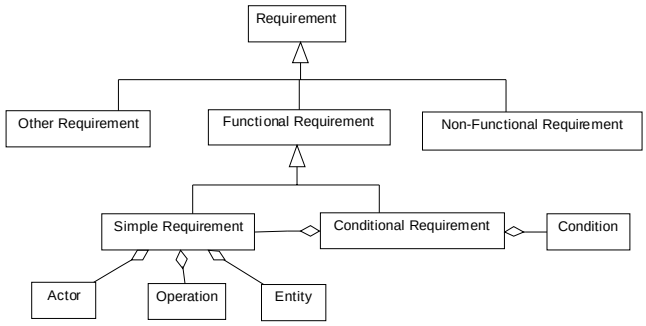

Figure 4 – ProjectIT-RSL metamodel – requirements classes

Besides the requirements sentences, a requirements specification in ProjectIT-RSL can also include two other types of sentences, represented in our metamodel by **Description** and **Declaration** classes (Figure 5); the first represents the sentences that begin the detailed

information about a concept, such as an entity (**Entity Description**), an actor (**Actor Description**) or an operation (**Operation Description**). Declarations represent sentences that enumerate the entities, the operations and the actors of the system (**Entity Declaration**, **Operation Declaration** and **Actor Declaration**, respectively), as well as those name the requirements module being defined, a System or Package (**System Declaration** and **Package Declaration**). For simplicity reasons, there are other classes in our current metamodel that are not shown in the above figures, such as the classes that represent the description of the properties of an Entity.

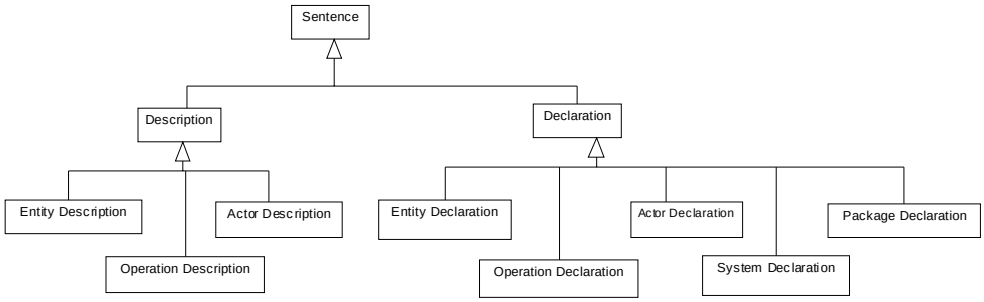

Figure 5 – ProjectIT-RSL metamodel – description and declaration classes

Open Issues

We recognize that there are some open issues in ProjectIT-RSL; some are minor ones, like the need to standardize on the terms used, the choice between singular and plural (for example, invoice or invoices) and the style adopted (the sentences should be kept as simple as possible). One of the most important questions still under discussion is the level of resemblance to natural language that we will adopt; the option is between a description very close to natural language, or one that adopts a style similar to a programming language. Until now we have not dealt with temporal restrictions, mainly because we are focused on interactive software systems, where most of the actions performed are sequential, between the actors and the system, or where the order is irrelevant.

The definition of the complete grammar is a process not yet finished, but will follow some general rules, like limiting the subject of sentences of type Actions and Activities to the word “System” or a name included in the Actor Declaration section, or limiting verbs to a restricted subset, whose syntax and semantics will be understandable by our tools, not only to validate that it is an allowed word (syntax validation), but also because we associate it with a specific pattern behavior, to support code generation.

Other types of requirements will have to be supported by our language. In fact, as Figure 4 has shown, he have already included in the metamodel two classes for Non-Functional Requirements and Other Requirements, for future use. However, as we have not provided any feature in our tools to validate these requirements and to take them into consideration by code generation techniques, they have not been object of current research, which is a situation we will change in the future.

An area considered of prime importance in this project is the support of requirements reuse. Besides the possibility of reusing a previously specified operation in the same requirements module, we have developed other techniques, such as “including” requirements modules in new modules. The information contained in the included module can be reused by the including module. This poses other questions, such as the visibility of the information of the included module, which are currently being object of discussion.

4. Prototype Development

In order to evaluate a preliminary version of ProjectIT-RSL, we decided to use the features provided by Visual Studio .NET (VS .NET) and .NET Framework to build a prototype. The choice was motivated by the built-in features provided by VS .NET, which we consider important, such as intellisense and syntax validation when writing code. Microsoft also provides VSIP, the Visual Studio Industry Partner Program, which are a set of COM APIs that enable the integration of new features in the VS .NET development environment, such as the possibility of adding new languages, or creating new types of projects. Besides VSIP, we used the Babel library, which comes together with VSIP, but provides a higher abstraction layer above it, and implementations of Lex (Flex [http://www.gnu.org/software/flex/]) and Yacc (Bison [http://www.gnu.org/software/bison/bison.html]), to implement the **Language Checker** in the **Editor**, and the **Compiler** of ProjectIT-RSL. The different components of the architecture are described with more detail in [Carmo 2005].

The creation of a new ProjectIT-RSL project follows the normal steps of creating any VS .NET project: (1) project creation; (2) code edition (Figure 6); and (3) code compilation.

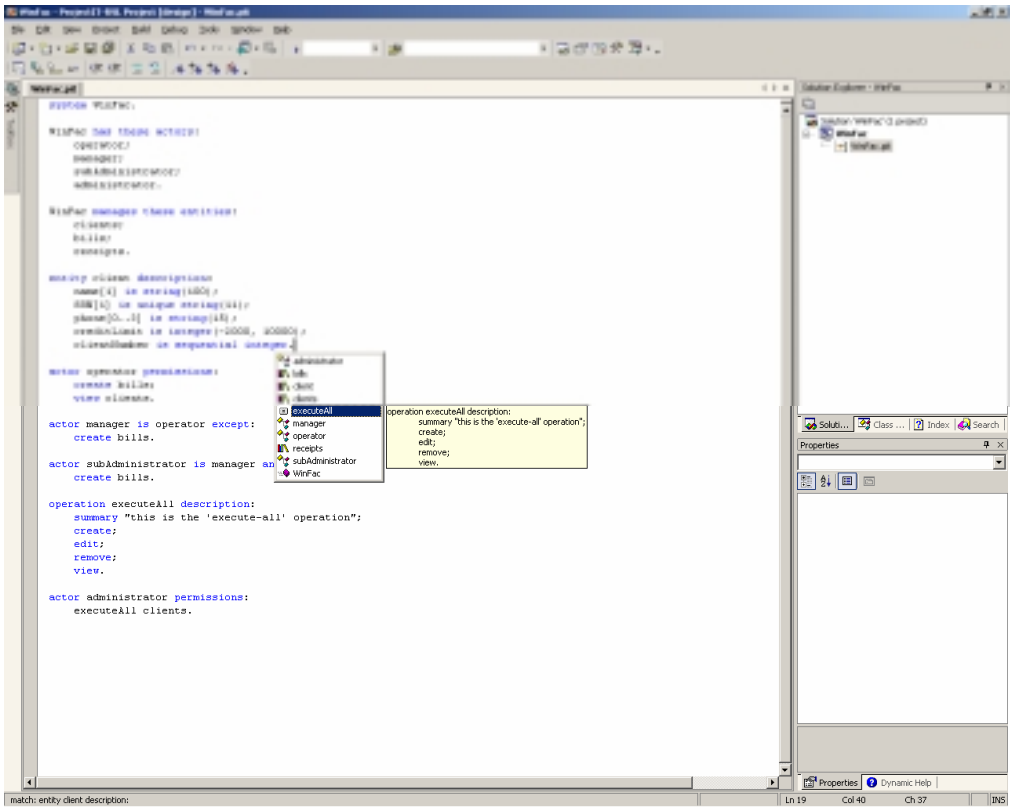

Figure 6 – The ProjectIT-RSL editor, integrated in the Visual Studio .NET IDE

We use the standard VS .NET editor, extended with the capability of writing ProjectIT–RSL sentences. The editor supports on-the-fly syntactic language verification and full syntax highlighting, which means that as we write, all expressions are validated (by the component generated for language checking) and in case there is any error, it is immediately detected and highlighted. Besides, the auto-complete feature is also always available giving the writer suggestions of how to complete the ProjectIT-RSL sentence.

Upon successful compilation, the compiler produces an XML file, which is currently the input to ProjectIT-MDD. The idea is that this XML file will be a common representation of the information used by our requirements specification, model and code generation tools. Because the existing model and code generation tools (currently named XIS-Tools) were based on the XIS-UML profile and developed prior to the existence of ProjectIT, they are not yet able to process part of the information contained in the XML file (for example, the information about the actors that interact with the system, and which defines the permissions of the users). For this reason, the XIS-UML profile is being reviewed, and renamed to PIT-UML profile, to become a unified metamodel, common to all our tools.

5. Related Work

The use of natural language in the initial phases of the software development process has received attention for more than 20 years. Abbot proposed that nouns could be used to identify classes, adjectives to identify attributes, and verbs to identify methods [Abbot 1983]. Tool support for the requirements activities has also been recognized as an important factor. For example, OICSI is a tool developed by Rolland and Proix to help the identification of requirements from natural language text and available domain knowledge [Rolland 1992]. The use of parsing techniques to elaborate a conceptual model from natural language requirements is a common approach; in [Macias 1993] and [Fantechi 2002] we can find descriptions of proposals to use a controlled natural language with a limited syntax in order to specify requirements with more quality.

Some of the previous initiatives were concerned with detecting problems in previously written requirements documents [Fabbrini 2000], while others are concerned with the elaboration of requirements documents without such problems ([Ben Achour 1998], [Denger 2002]). Attempto Controlled English (ACE), first described in [Fuchs 1996], is one of those approaches that use a controlled natural language to write precise specifications that, for example, enable their translation into first-order logic. Although the number of initiatives seems to justify the potential of natural language requirements, there are studies reporting problems in using natural language requirements specifications [Berry 2003].

The introduction of object-oriented concepts led to the development of some initiatives that integrate requirements and software model development. NL-OOPS [Mich 1999] and LIDA [Overmyer 2001] are systems that process natural language requirements to construct the corresponding object-oriented model. A similar system is described in [Nanduri 1996]. RML [Greenspan 1982] and Telos [Mylopoulos 1990], its successor, explored the use of modelling associated with formal techniques for requirements specifications.

The use of formal methods for specification tasks has been carefully reviewed by Axel Lamsweerde in [Lamsweerde 2000]. Reubenstein and Waters [Reubenstein 1991] presented an approach for specifying requirements in LISP using a natural language style. Ambriola and Gervasi proposed a “lightweight formal method” approach, supported by the use of modeling and model checking techniques to produce a formal validation of the requirements written in natural language [Gervasi 2002]. Clark and Moreira proposed the construction of a formal and executable user-centered model, which specifies the behavior of the system in terms of its utilization, as the basis for generating a system-centered model, using the same formal language [Clark 1999].

A number of different approaches have researched on the elaboration of requirements specification using patterns of natural language. Approaches such as [Ben Achour 1998] and [Rolland 1992] reduce the level of imprecision in requirements by using a limited number of

sentence patterns to specify a requirement for a particular domain. Denger and colleagues [Denger 2002] have also identified natural language patterns used to specify functional requirements of embedded systems, from which they developed a requirements statements metamodel. Juristo and Moreno try to formalize the analysis of natural language sentences in order to create precise conceptual model [Juristo 1999]. They based their approach on the identification, with a formal support, of linguistic patterns and how they can be mapped into conceptual patterns and the correspondent model.

Bryant and colleagues proposed an approach to generate models of components, according with the Model-Driven Architecture principles, from natural language requirements [Bryant 2003]. These requirements are then processed and mapped to a PIM and then to PSM, with the support of formal rules expressed in a Two-Level Grammar (TLG). A sequence of transformations enables the generation of VDM++, used for prototyping and specification validation, or for UML model or executable code generation.

Our ProjectIT approach follows the global principles of the Model Driven Architecture (www.omg.org/mda), which is a group of standards and principles that promotes the use of models during the whole development process, defining the system architecture and guiding all the development effort. It also promotes the separation between the specification and implementation of software, defining mechanisms for the transformation between those models, and separating the problem domain from the solution domain. Models play a crucial role in the whole process, leading to the idea that the development process is driven by models, as described in [Frankel 2003].

6. Conclusions and Future Work

In this paper we presented a metamodel of a controlled natural language, supported by tools that immediately validate the requirements written, adding the needed rigour, and which integrates with model driven development tools (MDD) with support for code generative features. Although sharing points with previous initiatives, we think that our approach has a unique combination of ideas that has not been tried in the past, namely (1) a controlled natural language, supported by a metamodel derived from the analysis of linguistic patterns, with (2) support by tools to immediately eliminate any errors in the written documents (many previous initiatives lacked tool support from the beginning of the specification process), (3) a strong support for requirements reuse (curiously, few initiatives have emphasized the importance of reusing requirements information; previous work in this area includes [Cybulski 2001]), and (4) a complete integration with MDD tools for model and code generation, which is fundamental for the automation of tasks and traceability between artefacts of the whole software development process.

We recognize that some of the goals of ProjectIT-RSL have not yet been achieved, so in the near future we will concentrate on the open issues described in section 3, in the development of the requirements reuse mechanisms and in advancing tool support. In a further iteration, it is our intention to include support for the specification of the goals of the system, and from them deriving the requirements specification, following an approach with similarities with the one described in [Letier 2002] and from a method of elaborating prototypes from requirements, proposed in [Martínez 2002]. When our ProjectIT-RSL and its supporting tools reach a sufficient maturity level, it is our intention to use them in real projects, to better test and proof the ideas we are proposing.

7. References

Abbot, R., *Program design by informal english description*, Communications of the ACM, 16(11), pp. 882-894, 1983

Ben Achour, C., *Guiding Scenario Authoring*, Proceedings of the 8th European-Japanese Conference on Information Modeling and Knowledge Bases, pp. 152–171, IOS Press, Vamala, Finland, May 1998

Berry, D., Kamsties, E., *Ambiguity in Requirements Specification*, Perspectives on Software Requirements, eds. J. C. Sampaio do Prado Leite and J. H. Doorn, Kluwer Academic, pp. 191-194, 2003

Bryant, B., Lee, B., Cao, F., Zhao, W., Gray, J., Burt, C., Raje, R., Olson, A., Auguston, M., *From Natural Language Requirements to Executable Models of Software Components*, Monterey Workshop on Software Engineering for Embedded Systems, pp. 51-58, Chicago, Illinois, September 2003

Carmo, J., Videira, C, Silva, A., *Using Visual Studio Extensibility Mechanisms for Requirements Specification*, 1st Conference on Innovative Views on .NET Technologies, Porto, June 2005

Clark, R., Moreira, A., *Formal Specifications of User Requirements*, Automated Software Engineering, Vol. 6, pp. 217–232, 1999

Cybulski, J., Application of Software Reuse Methods to Requirements Elicitation from Informal Requirements Texts, PhD Thesis, La Trobe University, Australia, March 2001

Denger, C., *High Quality Requirements Specifications for Embedded Systems through Authoring Rules and Language Patterns*, M.Sc. Thesis, Fachbereich Informatik, Universität Kaiserslautern, Kaiserslautern, Germany 2002

Fabbrini, F., Fusani, M., Gnesi, G., Lami, G., *Quality Evolution of Software Requirements Specifications*, Proceedings of Software and Internet Quality Week 2000 Conference, San Francisco, 2000

Fantechi, A., Gnesi, G., Lami, G., and Maccari, A., *Application of Linguistic Techniques for Use Case Analysis*, Proceedings of the IEEE Joint International Requirements Engineering Conference (RE'02), IEEE Computer Society Press, Essen, Germany., 2002

Frankel, D. S., Model Driven Architecture: Applying MDA to Enterprise Computing. Wiley, 2003

Fuchs, N., Schwitter, R., Attempto Controlled English (ACE), CLAW 96, First Int. Workshop on Controlled Language Applications, University of Leuven, Belgium, March 1996

Gervasi, V., Nuseibeh, B., *Lightweight Validation of Natural Language Requirements*, Software Practical Experiences, pp. 113-133, 2002

Greenspan, S., Mylopoulos, J. and Borgida, A., Capturing More World Knowledge in the Requirements Specification, Proceedings 6th Int. Conference on SE, Tokyo, 1982

Jones, C., Systematic Software Development Using VDM, Prentice Hall, USA, 1990

Juristo, N., Morant, J., Moreno, A., *A formal approach for generating oo specifications from natural language*, The Journal of Systems and Software, Vol. 48, pp. 139-153, 1999

Kotonya, G., Sommerville, I., *Requirements Engineering Processes and Techniques*, New York. Jonh Wiley & Sons, 1998

Lamsweerde, A., *Formal Specification: a Roadmap*, Proceedings of the Conference on The Future of Software Engineering, pp 147 - 159, Limerick, Ireland, 2000

Letier, E., Lamsweerde, A., Deriving Operational Software Specifications from System Goals, SIGSOFT2OO2/FSE-IO, Charleston, USA, November 2002

Macias B, Pulman S., *Natural language processing for requirements specification*, Safety-critical Systems, pp 57–89, Chapman and Hall: London, 1993

Martínez, A., Estrada, H., Sánchez, J., Pastor, O., From Early Requirements to User Interface Prototyping: A methodological approach, Proceedings of the 17th IEEE International Conference on Automated Software Engineering (ASE'02), Edinburgh, September, 2002

Mich, L., Garigliano, R., *The NL-OOPS Project: OO Modeling using the NLPS LOLITA*, Proc. of the 4th Int. Conf. Applications of Natural Language to Information Systems, pp. 215-218, 1999

Moreira, V., ProjectPRO, Plataforma de Gestão de Requisitos, Relatório de Trabalho Final de Curso, Lisboa, July 2003

Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M., *Telos: Representing Knowledge About Information Systems*, ACM Transactions on Information Systems, October 1990

Nanduri, S., Rugaber, S., *Requirements Validation via Automated Natural Language Parsing*, Journal of Management Information Systems, 12(2), pp 9-19, 1996

Overmyer, S., Lavoie, B., Rambow, O., *Conceptual Modeling through Linguistic Analysis using LIDA*, Proc. of the 23rd Int. Conf. Software Engineering, pp. 401-410, 2001

Reubenstein, H., Waters, R., The requirements apprentice: Automated assistance for requirements acquisition, IEEE Transactions on Software Engineering, 17(3), pp 226–240, 1991

Rolland, C., Proix, C., A Natural Language Approach for Requirements Engineering, Proceedings of the 4th Int. Conf. Advanced Information Systems, CAiSE 1992

Rumbaugh, J., Jacobson, I., Booch, G., *The Unified Modeling Language Reference Manual*, Addison Wesley, August 2004

Silva, A., Lemos, G., Matias, T., Costa, M., The XIS Generative Programming Techniques, Proc. of the 27th Annual Int. Comp. Software & Application Conference, Dallas, 2003

Silva, A., *O Programa de Investigação "ProjectIT"*, Technical report, V 1.0, October 2004, INESC-ID, in http://berlin.inesc.pt/alb/uploads/1/193/pit-white-paper-v1.0.pdf

Spivey, J., The Z Notation - A Reference Manual, Prentice-Hall, New Jersey, 1992

Videira, C., Silva, A., *The ProjectIT-RSL Language Overview*, UML Modeling Languages and Applications: UML Satellite Activities, Lisbon, Portugal, October 2004 [Videira 2004a]

Videira, C., Silva, A., *ProjectIT-Requirements, a Formal and User-oriented Approach to Requirements Specification*, Actas de las IV Jornadas Iberoamericanas en Ingeniería del Software e Ingeniería del Conocimiento - Volumen I - pp 175-190, Madrid, Spain, November 2004 [Videira 2004b]

Videira, C., Silva, A., *A broad vision of ProjectIT-Requirements, a new approach for Requirements Engineering*, in Actas da 5ª Conferência da Associação Portuguesa de Sistemas de Informação, Lisbon, Portugal, November 2004

Videira, C., Silva, A., *Patterns and metamodel for a natural-language-based requirements specification language*, in Proc. of the CaiSE'05 Forum, pp. 189-194, Porto, June 2005