ProjectIT-Requirements, a Formal and User-oriented Approach to Requirements Specification

Carlos Videira [1], Alberto Rodrigues da Silva[2]

[1] INESC-ID, Universidade Autónoma de Lisboa,
Rua de Santa Marta, nº 56, 1169-023 Lisboa, Portugal
cvideira@acm.org
[2] INESC-ID, Instituto Superior Técnico,
Rua Alves Redol, nº 9 –1000-029 Lisboa, Portugal
alberto.silva@acm.org

Abstract. Software requirements engineering is an essential activity for the successful development of information systems. The outcome of this activity is not always successful, which is visible in the lack of software quality, costs and schedules overruns. Although the efforts made and the initiatives proposed, there is not a widely accepted practice or standard in this area, comparable to what we have achieved, for example, in the modeling activities. This paper briefly describes the state of the art in the area of requirements specification, explains the motivation to develop a new initiative, which we have called "ProjectIT-Requirements" and enumerates the goals we want to achieve with the project, the context in which it integrates and the results obtained from the development of an initial prototype.

1 Introduction

The software development process is normally defined as a structured sequence of activities, executed in a systematic and uniform way, performed by roles with well defined responsibilities, which process a set of inputs to produce some outputs [32]. It also includes the definition of techniques, notations, tools, standards and guidelines that should be used.

It is commonly accepted that the development of information systems is a complex task, involving technical, human and organizational issues. When we started to develop information systems, the limited technology capacities were an important constraint that prevented the implementation of the most adequate solution for the business problems. The first initiatives proposed to solve these problems were concerned with the implementation issues, such as new programming languages or development environments.

However, since the late sixties that the IS/IT community recognized that, to solve the quality, budget and schedule problems, we had to find new and sounder approaches. As a result, the expression "Software Engineering" was first used in 1967 during an OTAN conference [28] to show the importance of applying systematic and rigorous approaches to software development, which had proven effective when ap-

In Actas de las IV Jornadas Iberoamericanas en Ingeniería del Software e Ingenierá del Conocimiento – Volume I - pp 175-190.

plied in other engineering disciplines. Although the many initiatives proposed to solve the software development process problems, the results have not been impressive; for example, The Chaos Report, a study periodically published by the Standish Group [http://www.standishgroup.com], has consistently shown that the number of problematic and unsuccessful projects during the last years is still very high, as Figure 1 shows.

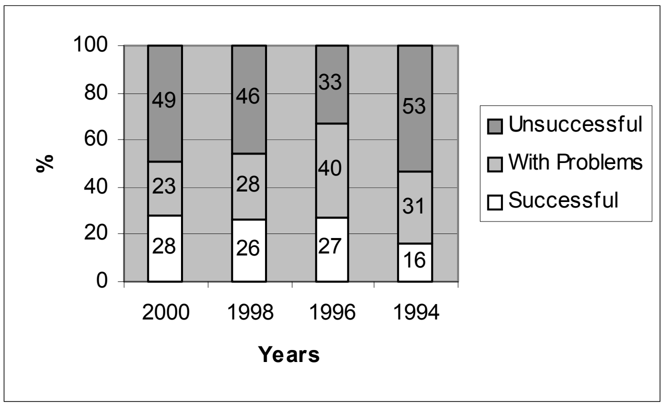


Figure 1 – Level of success of IS/IT projects, according with the Standish Group

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**, which proposes a new approach to successfully achieve some of the goals of this discipline. This paper describes the motivation for this project, the tasks executed and results achieved until now. It is organized in the following sections: section 2 defines the requirements concept and presents an overview of the requirements engineering activities; section 3 describes some requirements engineering practices and related work proposed in the past, which somehow influence our work; section 4 presents an overview of the ProjectIT initiative, whose main goal is to contribute with new ideas to improve the software development process; section 5 describes ProjectIT-Requirements, and presents its goals, architecture and main functionalities; section 6 describes the prototype development process, the results achieved and points some of the issues to be solved in the near future.

2 Requirements and Requirements Engineering

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 different views from the many different people that are somehow interested in the development of information systems. Words such as "needs", "features" or "functionalities" are frequently used as synonyms. The Oxford dictionary says that a "*requirement is a need, a dependency* *for success*". A classical definition from Kotonya says that a "requirement is a statement about a system service or constraint" [21]. A definition proposed by Dorfmann and Thayer suggests that a "*requirement is a software capability needed by the user to solve a problem he has to achieve a goal, or said another way, is a software capability that must be met or possessed by a system or component to satisfy a contract, standard, specification or other formalism*" [9]. In an article published in 1993, Harwell states that a "*requirement is something mandatory to be accomplished, transformed, produced, or provided*" [19].

Recognizing the importance of requirements in the development of information systems, we have grouped all the activities that deal with them in a new discipline called "requirements engineering". The term engineering emphasizes the importance of applying accepted and widely known engineering practices (such as measuring and modeling), in a systematic and repeatable way, to achieve results with quality. In [27] Nuseibeh provides an interesting overview of this discipline, identifying the techniques, activities and roles involved.

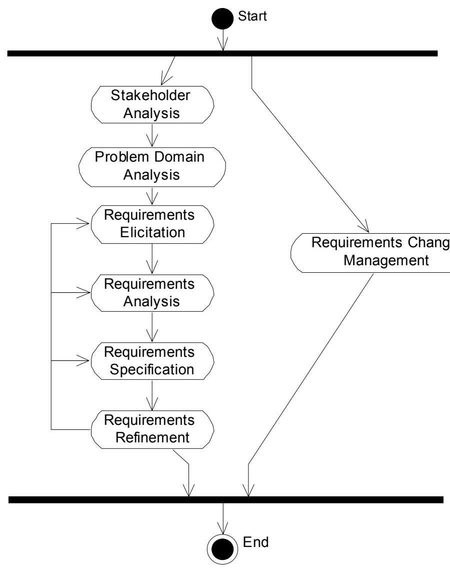


Figure 2 – Common activity diagram of the requirements engineering tasks

A generic workflow of the several activities that compose the requirements engineering process is shown in Figure 2; Kotonya [21] and the Rational Unified Process [22] present others with some variations. The number of different processes that are found in the literature are a consequence of the need to adapt the requirements engineering discipline to the needs of each organization, the internal and external practices and standards followed and each intervenient expectations. A complete description of each of these activities is out of the scope of this paper. In the next section we will present an overview of some existing methods that influence our work.

3 Related work in the area of Requirements Specification

Existing requirements engineering methods can be classified through many perspectives. According to the goals defined for our project, we have identified four areas of prime importance, which will be briefly reviewed in this section: (1) requirements reuse, because currently few tools truly support it, and this is crucial for minimizing the manual work in requirements specification; (2) requirements specification initiatives based on UML, because we want to use UML for defining our model and for integrating with other project initiatives and tools; (3) formal methods, because formalization is important to improve the quality and precision of requirements specifications; and (4) agile approaches, because we want to keep things as simple as possible.

The existing requirements reuse initiatives sometimes follow different strategies. In [38] the authors propose a metamodel for reusing requirements, based upon the integration of information available in a number of different existing functional modelling techniques (like scenarios, use cases, activity diagrams, data flows, task documents and workflows). The information captured is subsequently provided for being reused later. Cybulski proposes a framework which integrates the needs of software reuse in the requirements engineering process, and describes a method (RARE) and a software tool (IDIOM) that support the definition of the domain model, later used in the requirements analysis, reuse and refinement process [6].

UML has become a standard language for modelling software systems and many people have used it for requirements specification. However, some authors have argued that that UML has some deficiencies as a semiformal requirements specification language [16]. Other authors have proposed initiatives to formalize UML, by using formal specification techniques for the semantics of UML notations and diagrams. For example, the Precise UML group developed a precise semantic model for UML class diagrams, to enable formal deductions [11]. In [14] the authors describe a requirements modelling technique, named UMLtranZ, based on UML class and use cases diagrams, expressed in a variant form of the Fusion Model [4]. The models can be transformed to formal Z specifications, which can be further analysed to identify problems and inconsistencies in the requirements specified.

The use of formal requirements specification languages, based upon mathematical notations, such as Z [34], VDM [39] or Larch [18], has also its supporters among the requirements engineering community. They use deductive and inductive reasoning techniques that can be automated and built into specification tools. The question mostly faced by these initiatives is the adoption of those languages by non-technical people. Apparently, some studies demonstrated the capability of the users to learn and apply such methods [3].

The use of controlled natural language for requirements specification, although not a formal approach, facilitates the process of writing, analysing and verifying requirements [20]. Sometimes this specification can even be transformed into executable statements of software specification [31], providing systems simulation capabilities. Others followed a different strategy, by providing tools and techniques that transform formal requirements specifications into natural language, so that the stakeholders can validate them [12], [20].

Other initiatives worth of mentioning include (1) Kaos [7], a formal language for reasoning and modelling the system's goals, and its use in conjunction with formal specification languages, such as Timed Automata, for specifying the system's requirements and its internal behaviour [10]; (2) RML (Requirements Modelling Language), probably the first important attempt to use knowledge representation techniques in requirements engineering [17], or (3) Telos, its successor [26]; (4) RSML (Requirements State Machine Language), a formal system modelling language described in 1994 [24].

More recently, the agile software development approaches have also developed some specific techniques to be applied in the requirements activities. Ambler has discussed the best practices about agile requirements modelling, and how to integrate them with the rest of the agile development process [1]. Extreme Programming [2] proposes the concept of user stories, a set of small sentences that express the user needs, in his own words. The idea is to simplify, whenever possible, the process of information gathering about the system functionalities, doing the effort needed to capture the current user needs and nothing else. Detailed requirements information represents frequently an unnecessary overhead at the beginning of the project, because requirements change.

In the area of Human Computer Interaction, some proposals have been made to improve software usability, requiring the effective user involvement throughout the development process. Some authors, like Dix [8] call it Usability Engineering, emphasizing the usability aspects, while others call it User Centered Design. In terms of requirements, most of these initiatives, like Usage Centered Design [5], or GUIDE [30], propose an iterative process, where different types of requirements are gathered from the user of the system, then the system is designed or prototyped, and the user evaluates the result, which can originate more requirements. In [35] there is an overview of these processes; the important point to retain to our initiative is the close and permanent relation between the user and the IT people, in terms of the requirements activities.

Finally, there are some initiatives in the software development process that have proposed relevant best practices, which we want to take into account and use appropriately. Besides the model driven ideas, we recognize a strong influence from the ideas of product line software development [23] and agile development [2]. Aspect-oriented requirements engineering [29] and viewpoints [13] are also initiatives we will monitor to understand how they can influence our work.

4 The ProjectIT initiative

The Information Systems Group of INESC-ID is a group interested in research topics connected with software engineering and the software development process, and in applying them to the daily projects in which it is involved. In the recent past, the group has been involved in two research projects in the area of software engineering:

1. XIS, whose main contribution was the development of the XIS/UML profile intended to specify, simulate and develop information systems following a MDD approach [33].

2. ProjectPro, a Web based collaborative system to support the definition of basic concepts (projects, releases, sub-systems, requirements) and their relationships [25].

After the initial versions of both these projects, we reached the conclusion that their added value would be greater if we integrate them in a single project, which we have called "ProjectIT". The goal is to provide a complete software development workbench, with support for project management, requirements engineering, analysis, design and code generation activities. The current view of the functional architecture of the project is represented in Figure 3.

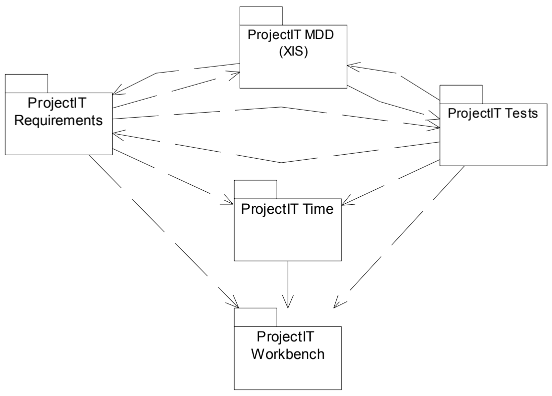


Figure 3 – ProjectIT architecture

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".

5 ProjectIT-Requirements

In the area of requirements engineering, the analysis of the existing initiatives and practices led us to conclude that there are important factors that justify new initiatives:

1. The research initiatives developed in the past, like those identified in sections 2 and 3, have represented an important contribution, but for a number of different reasons they have not been widely adopted.

2. The available tools, such as CaliberRM [http://www.borland.com/caliber/], RequisitePro [http://www-306.ibm.com/software/awdtools/reqpro/], Doors [http://www.telelogic.com/products/doorsers/], RTM [http://www.chipware.com/], although representing an important step forward, are above all requirements management tools; they have many features to input requirements in the system, to classify them, to define relationships, and to analyse them, but most of them still lack the capabilities to automate important tasks, or to guarantee a consistent integration with the rest of the development process (it is our intention to prepare a more detailed paper concerning this aspect).
3. There is an evident gap between the academic and research initiatives developed and their application and implementation in tools adopted by the market.

ProjectIT-Requirements is the component of the ProjectIT architecture that deals with requirements issues. The main goal of the project is to develop a model for the definition and documentation of requirements, which, by raising their specification rigor, facilitates the reuse and integration with development environments driven by models. Taking into account the different types of requirements, we must emphasize that we are currently interested in software requirements [21], those that can more easily be "converted" in software design models by MDD approaches. We recognize that other types of requirements, in particular non-functional requirements (such as usability, performance, security, safety, maintainability), are more difficult to formalize, and require different approaches.

5.1 ProjectIT-Requirements Goals

The main idea behind the project is that we must combine the benefits of formalizing the requirements specification, with the need to use a format and notation that is understandable by every involved project stakeholder. That is why we propose a "controlled natural language", a subset of natural language with specific rules for requirements specification, with a limited vocabulary and a simplified grammar. This type of language will enable every stakeholder in the software development process to easily use the ProjectIT-Requirements tools, and to reduce the time-consuming learning curve to work with the requirements deliverables of the project.

At the same time, by transparently raising the level of the rigor of the requirements specification through formalization (using a limited vocabulary with well defined rules), we will reduce the errors frequently found in the requirements specification process [21]. The generic principles followed by ProjectIT-Requirements are described with more detail in [37]. We have also identified the results we want to achieve, the most significant being:

1. The definition and management of the **project glossary**, which is elaborated by identifying the **business entities** and their **properties**.
2. The definition and formalization of a **requirements specification language**, which can be represented textually and graphically, and provides the basic mechanisms to define rigorously the system requirements. In order to maximize the benefits of close integration with the other components of the ProjectIT architecture,

benefiting from a single and global metamodel, we have decided to define a new language and not reusing an existing one.

3. The definition and management of **requirements architectures**, as a way to promote the reuse and increase the productivity in requirements engineering.
4. The definition of mechanisms to provide the **requirements management** in the context of specific projects based in requirements architectures.
5. The capability of **establishing relationships** between requirements and other project-based concepts, for example, projects, people, modules, etc.
6. The extension of the current XIS/UML profile in order to provide the **seamless integration with modelling activities** and the other ProjectIT sub-systems, resulting in the future ProjectIT/UML profile.

5.2 ProjectIT-Requirements in the context of ProjectIT

The main components of the ProjectIT-Requirements architecture in the context of ProjectIT are shown in Figure 4.

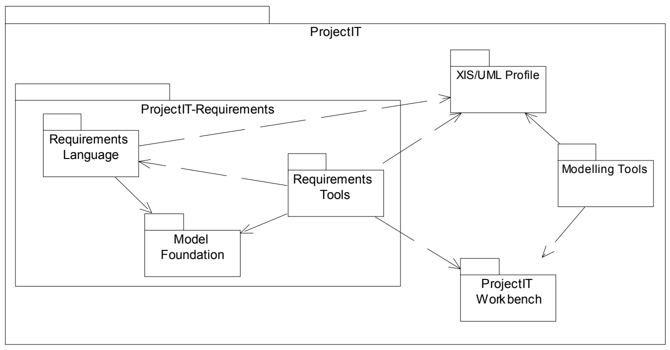


Figure 4 – Relationship between ProjectIT and ProjectIT-Requirements

The **Model Foundation** defines the project vision, its strategy, how it interacts with other ProjectIT components and the vocabulary used throughout the other parts of the project. The **Requirements Language** is defined by a model, which shows the concepts to be used for requirements description, and by a grammar for defining the rules to map these concepts into sentences. The first defines the language's syntax and the second its semantics. The **Requirements Tools** supports the definition of requirements and their exploration in real projects, in conformance with the language defined. The XIS/UML Profile acts as the bridge between ProjectIT-Requirements and ProjectIT-MDD, enabling the mapping between the requirements and the modelling and design artefacts produced (in fact, the requirements language will be represented by a model common to the XIS/UML Profile model).

5.3 ProjectIT-Requirements architecture

The ProjectIT-Requirements architecture, shown in Figure 5, is based upon an integrated set of tools that together will enable the definition of requirements and their relationships and provide the support for reusing previously defined requirements and the integration with modelling tools, developed in the context of ProjectIT-MDD.

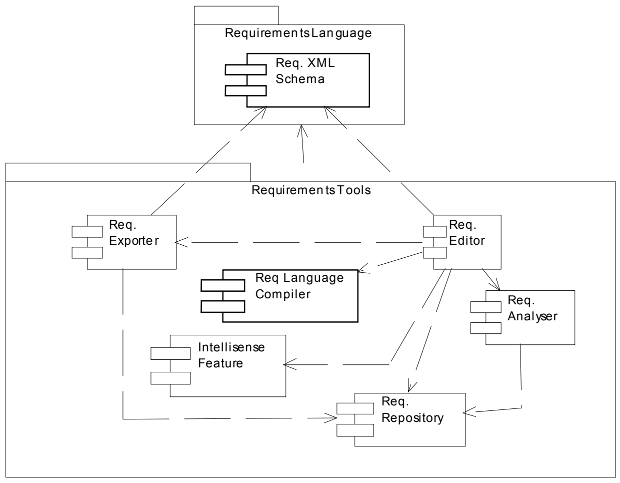


Figure 5 – ProjectIT-Requirements tools architecture

The information is stored in a **Repository** (either a DBMS or a XML file), whose information is managed by the **Requirements Editor**. It acts like a traditional editor for introducing controlled requirements text, but has additional options, like the possibility of classifying requirements (for later requirements reuse) and viewing traceability information. New requirements can be introduced in the system by reusing existing requirements. The editor will also provide other features that will be used if appropriate. For example, requirements can be associated with other concepts (such as sub-systems, systems, projects) managed by ProjectIT-Workbench; a requirement can be associated with different roles (e.g., the stakeholder who asked for it) and with different project deliverables (e.g., requirements specification, test cases).

The **Requirements Analyser** component, closely integrated with the editor, is responsible for viewing requirements and traceability information in different formats (e.g., documents, reports, matrixes, graphs); it works like a management information system for obtaining statistical information about requirements.

The **Exporter** component provides bidirectional integration with external tools. This component enables functionalities such as importing requirements written in a word processor, performing some validations and suggesting the changes needed to conform to our requirements language rules, besides importing and exporting requirements using standard formats. The integration with the modelling components of ProjectIT will be performed according the rules and semantics described in the XIS/UML profile, and its XML schemas.

Finally, the **Requirements Compiler** and **Intellisense**, are responsible for checking the requirements definitions, introduced in the editor, against the rules defined by the **Requirements XML Schema**, which is generated from the model, and represents our requirements specification language. Our vision is to build a tool for writing "requirements documents", like in a word processor, and as we write, it will warn us of errors violating the requirements language and grammar rules we have defined (very similar to what happens nowadays in the Word application, underlining in red some syntactical errors).

6 ProjectIT-Requirements prototype development

The architecture described in the above section represents our vision of the set of tools needed to implement the complete functionality of ProjectIT-Requirements. The need to establish a compromise between the functionality expected and the delivery of results, which clearly eases the validation of the concepts proposed, led us to initiate an iterative project, delivering small results in short periods of time. As such, we decided to develop an initial prototype of the most important deliverables of the project: (1) the requirements specification language, which we have called ProjectIT-RSL; (2) the main requirements tools, the editor and the compiler.

6.1 ProjectIT-RSL

The ProjectIT-RSL is currently a simple language that can be represented by a metamodel (a brief overview is shown in Figure 6), and by a grammar for defining the rules to map these concepts into sentences [36]. This metamodel identifies the basic concepts that the language will describe; the grammar will define the syntactic and semantic rules between these concepts, which will be validated by the compiler and the intellisense features.

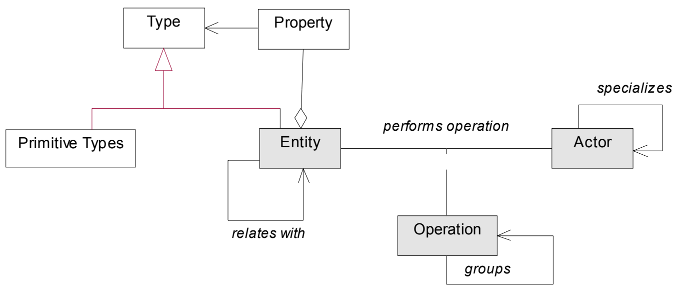


Figure 6 – ProjectIT-RSL metamodel simplified representation

Following our goal to keep requirements definition as simple and close to reality as possible, we analysed how requirements are most often specified. In most situations, requirements are expressed by normal language sentences that have a subject, a verb and other words that complement their meaning. We "transformed" this natural language into requirements language saying that *actors* carry out *operations*, which can access one or more *entities*. These are precisely the main concepts of our language:

- **Actors** are active resources (e.g., an external system or an end-user) that perform operations involving typically one or more entities.
- **Entities** are the static resources affected by the operations (e.g., a document, the data about a client or an invoice stored in a database). Entities have **Properties** that represent and describe their state.
- **Operations** are described by their respective workflows, which consist of a sequence of simpler Operations that affect Entities. This recursive definition will end in atomic and primitive Operations (e.g., create, update or delete operations) provided by default by our framework.

Another important concept not represented in the metamodel (for simplicity reasons) is the **System,** which represents a software component (either a complete application or a reusable component) with which an actor interacts, by executing the operations of the system to access or manage the entities available. Each of the three basic concepts can be reused in different systems; the idea is to provide requirements reuse mechanisms supported by the reuse of the requirements basic elements.

With this type of language we can express some of the most frequent software requirements for the kind of systems we are working on, which are above all interactive software applications. This model supports a simple description of a system (as shown below), which can later be used for generating part of it.

```
WinInvoice is a system
WinInvoice has three types of users:
 - the operator;
 - the manager;
 - the administrator.
The Client entity is described by:
 - a name;
 - a unique Social Security Number;
 - zero to three phone numbers;
 - a credit limit;
 - a sequential client number automatically assigned.
The operator can create bills and view clients.
The manager has the same permissions as the operator, but he cannot
create bills.
```

Considering the text above, we can see that the sentence specifying the permissions of the operator conforms to our model: the actor is the "operator", the operation is "create" and the entity is "bill". This description also points to some of the issues in ProjectIT-RSL that we are aware of, but have not implemented yet; some are minor issues, like the need to standardize on the terms (for example, always use the singular, such as bill instead of bills) and the style (the sentences should be kept as simple as possible and more elaborated sentences should be divided, when possible, into simpler ones).

Probably the most important question still under discussion is the level of resemblance to natural language that we will adopt; here the option is between a description like the one above, very close to natural language, or a "programming language" style, which is the approach followed in the current implementation, mainly due to practical reasons as we will see. Besides this issue, the next important step is to provide in ProjectIT-RSL the capability to specify a complex operation using simpler operations and workflows, thus gathering enough information for being later used by code generation techniques.

With the evolution, the metamodel of the ProjectIT-RSL language will integrate and extend the current XIS/UML profile. Their merging will result in ProjectIT/UML profile, a common metamodel for all initiatives developed under the ProjectIT programme.

6.2 Prototype Development

To test our initial model of ProjectIT-RSL in an iterative way, we built a prototype of the editor and compiler of the ProjectIT-RSL. Although we intend to develop an application integrated in a new development environment, to shorten the initial development time and to quicker validate the model of ProjectIT-RSL and see how it supported the requirements specification task and what type of additional features were still needed, we decided to take the benefits of the features provided by Visual Studio .NET and the .NET Framework [http://msdn.microsoft.com/] and we built a prototype in this environment. We chose this development environment because it provides some of the features we consider important, such as intellisense and syntax validation when writing code. Microsoft also provides the Visual Studio Industry Partner Program [http://msdn.microsoft.com/vstudio/extend/], abbreviated VSIP, which are a set of COM APIs that enable the integration of new features in the Visual Studio.NET development environment, such as the possibility of adding new languages, or creating new types of projects. With these features, we can directly use the Visual Studio development environment, to edit new types of projects in new languages, without having to develop a new editor from scratch.

The VSIP APIs are powerful, but difficult to understand and very low-level from the developer's point of view (they are written in C++ with the code difficult to understand). That is why we decided to use the Babel library that comes together with VSIP, but provides a higher abstraction layer above it. For example, the intellisense feature is handled directly by Babel and can be accessed from it. We also used implementations of Lex and Yacc (Flex [http://www.gnu.org/software/flex/] and Bison [http://www.gnu.org/software/bison/bison.html]) to implement the features associated with language checking in the editor, and also for the compiler of the ProjectIT-RSL. They are classical tools used to check syntactic and semantic rules from programming languages, based upon a grammar description and analysis of regular expressions. Figure 7 shows how the different components of the architecture integrate to provide the global functionality.

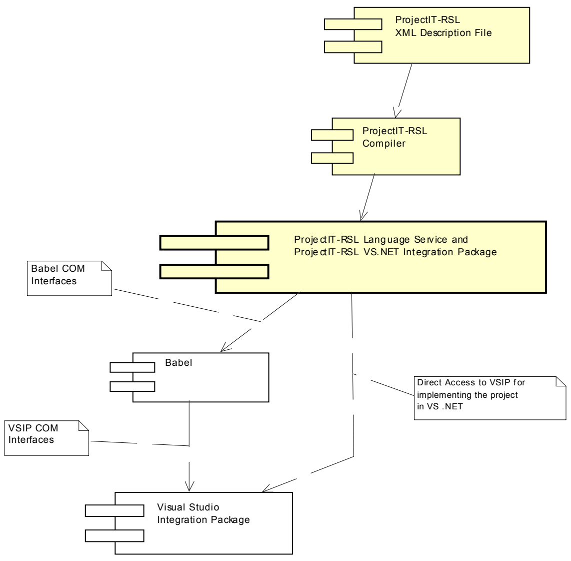
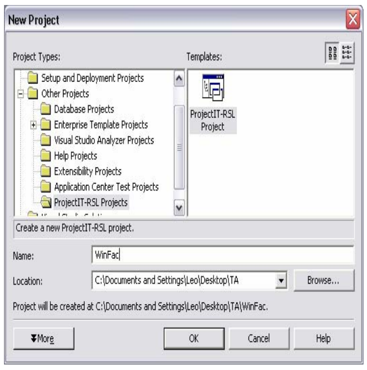

ProjectIT-RSL XML Description File
ProjectIT-RSL Compiler
ProjectIT-RSL Language Service and ProjectIT-RSL VS.NET Integration Package
Babel COM Interfaces
Babel
Direct Access to VSIP for implementing the project in VS .NET
VSIP COM Interfaces
Visual Studio Integration Package

Figure 7 – ProjectIT-Requirements tools architecture

Using these tools we have implemented the possibility of creating new projects in Visual Studio .NET, the ProjectIT-RSL projects (Figure 8).



Create a new ProjectIT-RSL project.

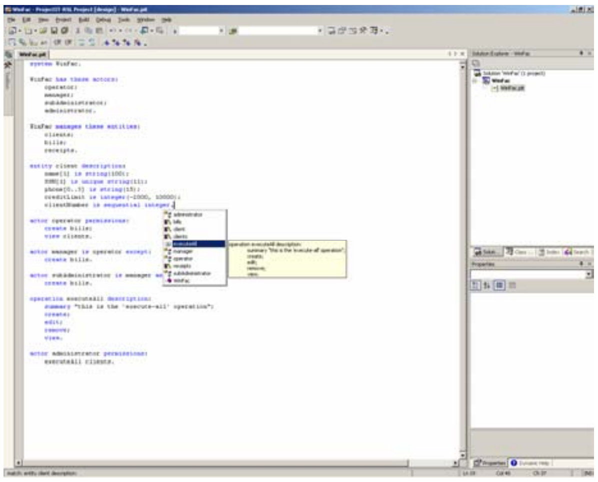
Name: WinFac

Location: C:\Documents and Settings\Leo\Desktop\TA

Project will be created at C:\Documents and Settings\Leo\Desktop\TA\WinFac.

Figure 8 – Creating a ProjectIT-RSL project in Visual Studio .NET

This project uses the standard Visual Studio .NET editor, extended with the capability of writing the ProjectIT–RSL sentences, according with the model presented in Figure 6. The editor, shown in Figure 9, 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 by Bison for language checking, and according with the rules previously defined) 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 correction or of the next allowed and available term, according with the language.



```
system WinFac.

WinFac has these actors:
    operator;
    manager;
    subAdministrator;
    administrator.

WinFac manages these entities:
    clients;
    bills;
    receipts.

entity client description:
    name[1] is string(100);
    NIF[1] is unique string(11);
    phone[0..3] is string(15);
    creditLimit is integer(-1000, 10000);
    clientNumber is sequential integer;

actor operator permissions:
    create bills;
    view clients.

actor manager is operator except:
    create bills.

actor subAdministrator is manager ex
    create bills.

operation executeAll description:
    summary "this is the 'execute-all' operation";
    create;
    edit;
    remove;
    view.

actor administrator permissions:
    executeAll clients.
```

```
operation executeAll description:
        summary "this is the 'execute-all' operation";
        create;
        edit;
        remove;
        view.
```

Figure 9 – The ProjectIT-RSL editor

Upon successful compilation, the compiler produces an XML file, which is the input to ProjectIT-MDD. This component processes this file, and will be able to generate different project artifacts, including a UML model of the information specified and code, following with an MDD approach.

7 Conclusions

In this paper we tried to justify the need for new initiatives in the area of requirements engineering, which is recognized to be a crucial part of the software development process. We strongly believe that the integration with other project tasks, in a model driven development approach, reusing previously developed requirements, expressed in a more formal (yet universally understandable) way, and which are used by automated tasks, are essential characteristics that must be taken into consideration by any modern initiative in the area of software engineering. These are precisely the main drivers of ProjectIT-Requirements.

We have already established ProjectIT-Requirements architecture, defined an initial version of ProjectIT-RSL, a basic component of our project. To test it, we have built a prototype of the editor and compiler reusing many features provided by the Visual Studio .NET development environment. We are now initiating the next iteration of the project's implementation, which includes adding more features to ProjectIT-RSL (like the support for the specification of more complex and versatile operations and workflows, and the checking of additional rules), and the implementation of requirements reuse based on requirements architectures. In this process, we will integrate the developed features into our development environment, and enable the integration with ProjectIT-MDD. When we accomplish these goals, it is our intention to use the developed tools in real projects, to test and proof the ideas we are proposing.

We are aware that this is an ambitious project, and that there are still some open issues for which we will actively research in the near future. We pretend to make some innovative proposals, but we also intend to integrate the ideas of previously developed initiatives. We strongly believe that the apparent contradiction between formalization and user involvement, which we consider both fundamental for the success of software development, can be overcome if we follow a pragmatic approach like the one proposed in this paper.

References

1. Ambler, Scott, Agile Requirements Modeling, Essay, available at http://www.agilemodeling.com/essays/agileRequirements.htm, 2004
2. Beck, K. Extreme Programming Explained: Embracing Change, Addison Wesley, October 1999
3. Bowen, J., Hinchey, M., Seven More Myths of Formal Methods, PRG-TR-7-94,:Oxford University Computing Laboratory, 1994
4. Coleman, D., et al., Object-Oriented Development: The Fusion Method. Prentice Hall, Englewood Cliffs, NJ, Object-Oriented Series edition, 1994
5. Constantine, L., *Essentially Speaking*. In Software Development, Vol. 2, No. 11, pp. 95-96, 1994
6. Cybulski, J., Application of Software Reuse Methods to Requirements Elicitation from Informal Requirements Texts, PhD Thesis, La Trobe University, Australia, March 2001
7. Dardenne, A., A. van Lamsweerde, A., Fichas, S., Goal directed requirements acquisition, Science of Computer Programming, 20:3–50, 1993
8. Dix, A. et al., *Human Computer Interaction*. Prentice Hall, Boston, 2003
9. Dorfmann, M., Thayer, R., Standards, Guidelines, and Examples of System and Software Requirements Engineering, Los Alamitos, California, IEEE Computer Society Press, 1990
10. Dubois, E., Yu, E., Petit, M., From Early to Late Formal Requirements: a Process-Control Case Study, Proc. 9th Int. Workshop on Software Specification and Design, Isobe, IEEE CS Press, 34-42, April 1998
11. Evans, A., France, R., Lano, K., Rumpe, B., Developing the UML as a Formal Modelling Notation, UML'98, Beyond the notation International Workshop, Mulhouse France, 1998
12. Feather, M., Reuse in the context of a transformation-based methodology, in Software Reusability: Concepts and Models, Biggerstaff, T.J. and Perlis, A.J. (Editors). New York, New York: ACM Addison Wesley Publishing Company. p. 337-359, 1989
13. Finkelstein, A., Nuseibeh, B., Kramer, J., A framework for expressing the relationships between multiple views in requirements specification. Transactions on Software Engineering. 20(10): p. 760-773, 1994
14. France, R., Grant, E., Bruel, J., UMLtranZ: An UML-based rigorous requirements modeling technique, Technical report, Colorado State University, Ft. Collins, Colorado, January 2000
15. Fuchs, N., Hofmann, H., Schwitter, R., Specifying Logic Programs in Controlled Natural Language, 94.17,: Department of Computer Science, University of Zurich, 1994
16. Glinz, M., Problems and Deficiencies of UML as a Requirements Specification Language, Proc. of the 10th IEEE Int. Workshop on Software Specification and Design, 2000

17. Greenspan, S., Mylopoulos, J. and Borgida, A., Capturing More World Knowledge in the Requirements Specification, Proceedings 6th International. Conference on SE, Tokyo, 1982
18. Guttag, J., Horning, J., Larch: Languages and Tools for Formal Specifications, New York. Springer-Verlag. 1993
19. Harwell, R. et al, What Is A Requirement?, in Proceedings of the Third International Symposium of the NCOSE, 1993
20. Johnson, W., Feather, M., Using evolution transformation to construct specifications, in Automatic Software Design, Lowry, M.R. and McCartney, R.D. (Editors). Menlo Park, California. The MIT Press. p. 65-91, 1991
21. Kotonya, G., Sommerville, I., *Requirements Engineering Processes and Techniques*, New York. Jonh Wiley & Sons, 1998
22. Kruchten, P., *The Rational Unified Process: An Introduction*, Addison Wesley, January 2004
23. Kuloor, C., Eberlein, A., Requirements Engineering for Software Product Lines, Proc. of the 15th Int. Conference on Software & Systems Engineering and their Applications, Paris, 2002
24. Leveson, N., Heimdahl, M., Hildreth, H., Reese, J., Requirements specification for process-control systems. IEEE Transactions on Software Engineering, pp 684-706, September 1994
25. Moreira, V., ProjectPRO, Plataforma de Gestão de Requisitos, Relatório de Trabalho Final de Curso, Lisboa, July 2003
26. Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M., Telos: Representing Knowledge about Information Systems," ACM Transactions on Information Systems, October 1990
27. Nuseibeh, B., Easterbrook, S., Requirements Engineering: a roadmap. In Proc. of the Conference on The Future of Software Engineering, pages 35-46, Limerick, Ireland, 2000
28. Pressman, R., Software Engineering: A Practitioner's Approach: European Adaptation. 5th Edition, McGraw-Hill Education, Europe, 2000
29. Rashid, A., Sawyer, P., Moreira, A., Araújo, J., Early Aspects: a Model for Aspect-Oriented Requirements Engineering, Requirements Engineering 2002 (RE'02), Essen, Germany, 9-13 September 2002
30. Redmond-Pyle, D., Moore, A., *Graphical User Interface Design and Evaluation*. Prentice-Hall, London, 1995.
31. Schwitter, R., Fuchs, N., Attempto - from specifications in controlled natural language towards executable specifications, in GI EMISA Workshop, Natürlichsprachlicher Entwurf von Informationssystemen. Tutzing, Germany, p. 163-177, 1996
32. Silva, A., Videira, C., UML, Metodologias e Ferramentas CASE, Centro Atlantico, Lisboa, 2001
33. Silva, A., Lemos, G., Matias, T., Costa, M., The XIS Generative Programming Techniques, Proc. of the 27th Annual Int. Computer Software & Application Conference, Dallas, 2003
34. Spivey, J., An introduction to Z and formal specifications, Software Engineering Journal. 4(1): p. 40-50, 1989
35. Videira, C., A review of user centered processes, I Jornadas de Postgrado en Ingeniería Informática, Salamanca, Spain, May 2004
36. Videira, C., Carmo, J., Silva, A., *The ProjectIT-RSL Language Overview*, accepted for publication in UML 2004, Lisbon, October 2004
37. Videira, C., Silva, A., *A broad vision of ProjectIT-Requirements, a new approach for Requirements Engineering*, accepted for publication in CAPSI 2004, Lisbon, November 2004
38. Villegas, O., Laguna, M. Garcia, F., Reuse based Analysis and Clustering of Requirements Diagrams, Proceedings of REFSQ'02, 8th Int. Workshop on Requirements Engineering: Foundation for Software Quality, Essen, Germany, September 2002
39. Woodman, M. Heal, B., Introduction to VDM. McGraw-Hill, London, 1993