# Patterns and parsing techniques for requirements specification[1]

Carlos Videira [1], David Ferreira [2], Alberto Rodrigues Silva [3]

cvideira@acm.org, dafe@ist.utl.pt, alberto.silva@acm.org

[1] INESC-ID/UAL, Rua Alves Redol, nº 9 –1000-029 Lisboa, Portugal
[2] INESC-ID/IST, Rua Alves Redol, nº 9 –1000-029 Lisboa, Portugal
[3] INESC-ID/IST, Rua Alves Redol, nº 9 –1000-029 Lisboa, Portugal

**Abstract:** Despite the similarity with other engineering disciplines, the development process of information systems still suffers from several critical problems that compromise their success, namely in terms of time, budget and quality. The main cause for this situation is the fact that the majority of IT projects do not follow a structured, standard and systematic approach like the methodologies and best practices proposed by Software Engineering. In this paper, we describe a series of steps involved in the proposal of a new controlled natural language for requirements specification, called ProjectIT-RSL, based on the identification of the linguistic patterns that are most frequently used in requirements documents written in natural language. Requirements are then analyzed by special parsing tools, and immediately validated according with the syntactic and semantic rules of the language.

**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. Requirements describe what the system should do, which is obviously critical for the success of the whole development process. Several surveys and studies (such as The Chaos Report, available at http://www.standishgroup.com) have emphasized the costs and quality problems that can result from mistakes in the early phases of system development, such as inadequate, inconsistent, incomplete, or ambiguous requirements (Bell, Thayer, 1976).

[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*).

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/) started an initiative in the area of requirements engineering, named ProjectIT-Requirements (Videira, Silva, November 2004). One of the results of this project is a new requirements specification language, called ProjectIT-RSL (Videira, Silva, October 2004). To support the definition of this language, we initially developed a prototype (Carmo, Videira, Silva, 2005) using Visual Studio .NET and its extensibility mechanisms. However, because the development of this requirements specification language is integrated in a broader project, called ProjectIT (Silva 2004), and the need to integrate the various components is a critical requirement, we decided to develop an integrated set of tools, called ProjectIT-Studio, which covers all phases of the development of an IT product. To provide a base platform and framework for all the tools, we developed Eclipse.Net, a migration of the Eclipse platform to the Visual Studio .NET environment. Upon this base platform, we developed a text editor plug-in for Eclipse.Net, with features of a specialized text editor adequate for our requirements specification language, ProjectIT-RSL.

This paper describes the main evolution steps of this research project, and in particular of those more closely related with ProjectIT-RSL. It is organized in the following sections: section 2 discusses the importance of requirements specification languages and the use of natural language; section 3 presents a brief overview of the ProjectIT research program; section 4 introduces ProjectIT-RSL, namely its motivation and the main concepts; section 5 briefly describes an initial prototype developed to test ProjectIT-RSL and describes the motivation and issues in developing Eclipse-NET, a new base platform for all ProjectIT subsystems; section 6 presents the main features of the Eclipse-NET plug-in for requirements specification, and the tools used to develop it; section 7 presents related work of previous initiatives; finally, section 8 justifies our perception that this proposal has some innovative contributions and presents the work to be performed in the future.

## 2. The importance of Requirements Specification Languages

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, Sommerville, 1998). 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 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, Jacobson, Booch, 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).

## 3. 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 the daily projects in which it is involved. We started a project, called ProjectIT (Silva, 2004), to provide a complete software development workbench, with support for project management, requirements engineering, and analysis, design and code generation activities. The functional architecture of this research program is represented in Figure 1.

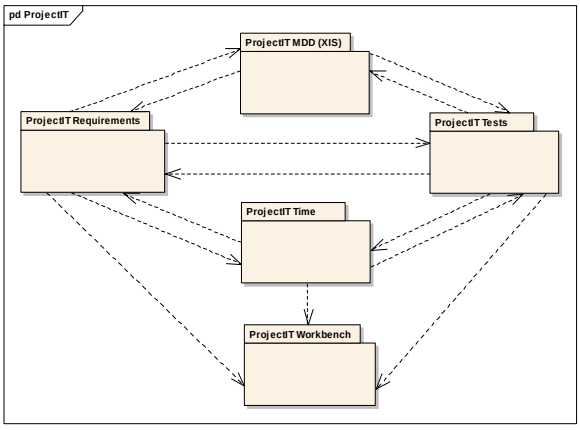


Figure 1. ProjectIT functional architecture

ProjectIT-Requirements (Videira, Silva, November 2004) is the component of the ProjectIT architecture that deals with requirements issues. The main goal of this project is to develop a model for the specification of requirements of interactive systems, which, by raising their specification rigor, facilitates the reuse and

integration with development environments driven by models. Our vision was 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, as described in (Videira, Silva, October 2004).

## 4. ProjectIT-Requirements Specification Language

To define ProjectIT-RSL, we analysed the format and structure of requirements of the projects we have developed, which are mainly interactive systems. Then, we identified a common set of linguistic patterns for the requirements of this type of systems, and finally derived a metamodel of requirements that can adequately represent the patterns identified.

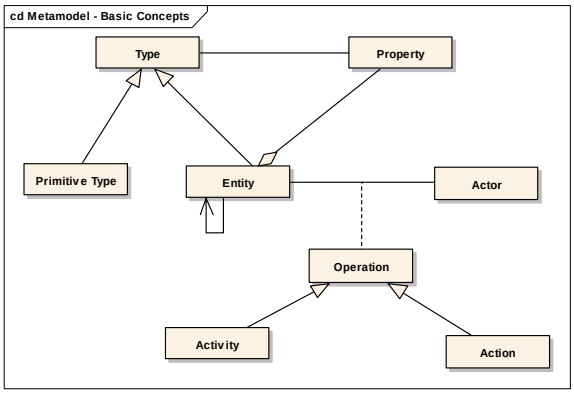


Figure 2. ProjectIT-RSL metamodel – basic concepts

Most of the sentences used in interactive systems requirements specifications specify what the system should do, and present an operational perspective of the system. This is why they follow a simple and common structure, where a *subject* performs an *operation* (expressed by a verb), which affects an *object*. More elaborated sentences can also include a condition, which describes a constraint about a simpler fact.

The analysis of patterns led to the identification of the main concepts of our language (what we can call our ontology), represented in Figure 2:

- **Actors** are active resources (e.g., an external system or an end-user role) that perform operations involving one or more entities.
- **Entities** are static resources affected by operations (e.g., a client or an invoice). 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 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.

Besides these basic concepts, we have also identified more specialized concepts, which were initially described in (Videira, Silva, June 2005) and have recently been revised. All requirements information is written in documents that follow a specific format; the **Sentences** are grouped in specialized sections, according to their goal (**Business Entities**, **Functional Requirements** or **Non-Functional Requirements** specification). These sentences allow the specification of information about:

- **Templates,** which allow the specification of information about basic concepts (**Actors**, **Entities**, **Operations**, **Requirements**) that can be reused in other requirements documents.
- **Systems**, which represent 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.

The following list is just a subset of our current requirements patterns, and illustrates some that are used for business entities information specification.

```
<Entity Definition>: <Entity Inheritance Definition> | <Entity Property
    Definition> | <Entity Equivalence Definition> | <Entity Association
    Definition>
<Entity Inheritance Definition>: <Entity> is a <Entity>
<Entity Equivalence Definition>: <Entity> is the same as <Entity>
<Entity Property Definition>: <Entity> has <Property Definition>*
<Property Definition>: [ a | an | the ] [<Primitive Type>] [<Quantifier>]
    <Property>
<Quantifier> : <number> | at least <number> | at most <number> | a list of |
    each | …
<Property>: Name | <Entity>
<Entity Association Definition>: <Entity Active Association Definition> |
    <Entity Passive Association Definition>
<Entity Active Association Definition>: [<Quantifier>] <Entity> <Active Verb>
    [<Quantifier>] <Entity>
<Entity Passive Association Definition>: [<Quantifier>] <Entity> <Passive
    Verb> [<Quantifier>] <Entity>
```

The above patterns enable writing requirements sentences such as:

```
An entity has a name, an address, a fiscal number and a list of more than one
    telephone.
A client is an entity, and has a date of birth, a number of an identification
    document, the date of the identification document.
An address is composed by the street name, the door number, the floor number,
    the floor location, the name of the village, the name of the country and
    the postal code.
A postal code is composed by a number and by a location.
```

## 5. From a prototype to a new base platform

While defining the Requirements Specification Language, we decided to evaluate, at the same time, its applicability, by developing a prototype. We decided to use the features provided by Visual Studio .NET (VS .NET) and .NET Framework to

build this prototype, because its built-in features, such as intellisense and on-the-fly syntax validation, enabled us to obtain results in a shorter timeframe.

Particularly important in the context in this prototype where the following components:

- 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.
- The Babel library, which comes together with VSIP, but provides a higher abstraction layer above it.
- 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, Videira, Silva, 2005).

A ProjectIT-RSL project follows the normal project creation cycle used in any VS .NET project: (1) project creation; (2) code edition; and (3) code compilation. We used 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. 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.

One of the problems detected with the first version of the ProjectIT project (from which the above prototype was one of the components) was the difficult integration between the various tools. Namely, in the context of the ProjectIT-Studio workbench, the different components were able to communicate (ProjectIT-MDD, ProjectIT-RSL, etc.), but the integration was enabled by the use of a common XML file, which was clearly not the most adequate solution. That was the reason why we decided to build a common base platform for all the tools; the adopted solution was the conversion of the open source Eclipse platform, written in Java, to VS .NET, which we called **Eclipse.NET** (more information in http://sourceforge.net/projects/eclipsedotnet/, and in Saraiva, 2005).

Based on the features of Eclipse, Eclipse.NET offers a powerful extensibility mechanism, implementing a plug-in architecture. It is an open platform that supports the development of integrated tools for the .NET runtime environment. The default configuration of the Eclipse.NET platform includes a set of plug-ins that extends the platform with some of the features typically found in an IDE, but it is more than an IDE, following a similar strategy as adopted by Eclipse.

The Eclipse platform was converted and adapted for being used in ProjectIT-Studio and the ProjectIT-MDD Tool, originally developed in the XIS Project (Silva,

2003), was the first tool integrated in ProjectIT-Studio. Each ProjectIT subsystem (ProjectIT-MDD, ProjectIT-Requirements, for example) can have one or more modules, enabling the reuse of specific features between various tools, which promotes the development, debugging and maintenance of ProjectIT-*Studio*.

## 6. ProjectIT-Requirements architecture

Upon Eclipse.NET we have already developed an initial version of the ProjectIT-RSL plug-in, benefiting from the extension points provided by Eclipse .NET. This plug-in is responsible for performing specific text editing features, not available in Eclipse.NET, namely the auto-complete and syntax highlight according with the rules of ProjectIT-RSL. To extract the most information from the free-form natural language requirements text, so that the code generation tools of ProjectIT can use it to successfully generate code, we implemented a series of transformation and parsing steps, needed for morphological and syntactical analysis.

These components are aggregated in two main packages, one that includes all the parsers needed to perform Natural Language Processing (NLP), using the text of the requirements document (**PIT-RSL Parser**), and the other includes all the functionality associated with the text editor plug-in developed for the Eclipse.NET platform (**PIT-RSL Plug-in**), as shown in Figure 3.

The **PIT-RSL Parser** package contains all PIT-RSL text editor independent parsers, namely a rules based fuzzy matching algorithm for Natural Language Processing (**RSL Fuzzy Matching Parser)**, and a parser generated by CSTools (a compilers' generation tools framework, available at http://cis.paisley.ac.uk/crow-ci0/), which deals with the early typographical and structural analysis stages of the parsing process (**RSL Structural Parser**). CSTools provides a rich object-oriented API and a similar Lex/Yacc scripting language for LARL (1) parsers generation.

The **RSL Structural Parser** is generated by CSTools, upon processing one lexical and one grammar files, which contain the lexical and context-free grammar rules, respectively. This component is required for performing initial parsing stage analysis, such as the document structure and format/typographical normalization.

The **PIT-RSL Parser** also contains the **RSL Fuzzy Matching Parser**, responsible for processing natural language text. The goal is to find the optimal parsing tree, by successive comparisons between natural languages patterns (defined by a set of rules, which we have called **TS rules**) and the written requirements; every time a valid match occurs, the parser performs the respective rules substitutions in a controlled recursive and exhaustive process, until no more patterns can be applied. Each of these rules is composed by a template part that is used to match the requirements sentences and a substitution part, which replaces the original statement when a match is found; the idea is that, with the generated information, more knowledge is obtained and further analysis and validations can be performed.

The **PIT-RSL Plug-in** package represented in Figure 3 is the root package of the Plug-in, and encapsulates specific text editor's (a screen of the text editor is represented in Figure 4) behavior and the base classes responsible for the plug-in

initialization, logging, and termination. It contains **RSL-TextEditor** and **RSL-Templates Import View**. The former encompasses the code responsible for extending the available text editor extension point of the Eclipse.NET platform. The later contains the necessary code for providing a tree viewer GUI component that displays the hierarchical representation tree of templates available for import. It also shows the templates categories and types.

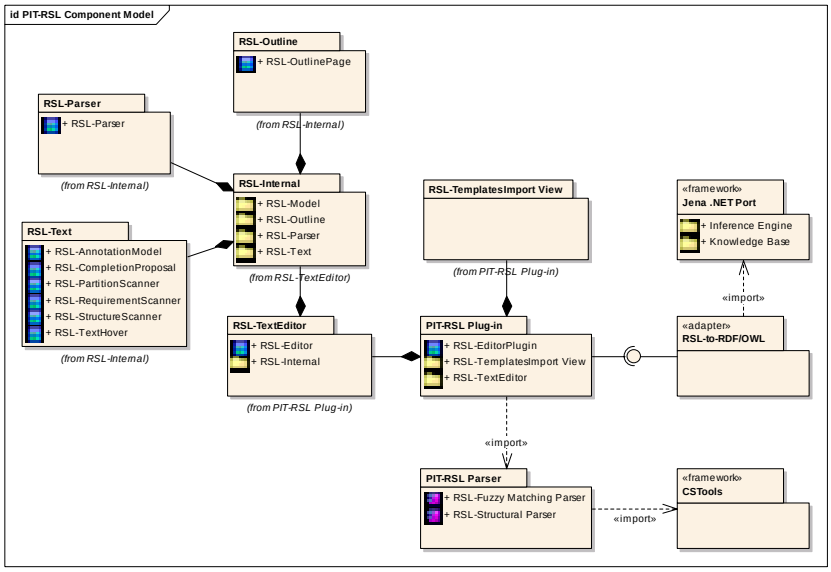


Figure 3 – ProjectIT-RSL Plug-in and Parsers architecture

The **RSL-TextEditor** package includes a package named **RSL-Internal** that is composed by all the necessary code for all internal data representation structures and the operations that allow their creation, manipulation, and disposal. **RSL-Internal** is composed by **RSL-Outline**, **RSL-Parser**, and **RSL-Text**.

**RSL-Outline** covers all the aspects related to the GUI outline component for presenting, synchronizing, and showing the graphical manipulation changes (such as drag-and-drop) in the actual requirements document contents domain model in terms of structure, entities and their inheritance relations. The class that implements this behavior is the emphasized class **RSL-OutlinePage**.

**RSL-Parser** implements the required text editor internal parsing in order to construct the abstract syntax tree of the requirement documents that feeds the input of all the perspective views (e.g. outline view, import view, and optimal parse trees view) and allows requirements semantic analysis for providing feedback to the user, such as syntax-highlighting, warnings and errors annotations, and other visual elements that provide guidance to the user in the requirements specification process.

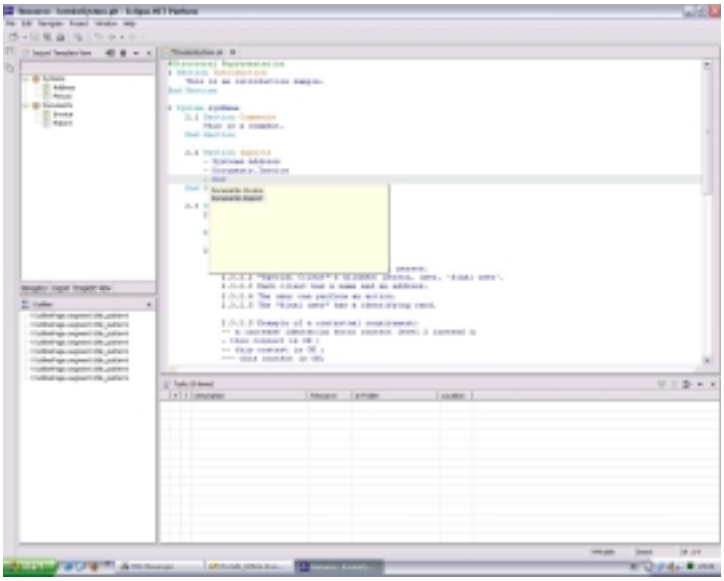

Figure 4 – Example of the editor

**RSL-Text** includes all features associated with visual feedback, previously enumerated in the **RSL-Parser**. The following are the most important classes:

- **RSL-AnnotationModel** allows the definition of annotation rules for text underline and overview ruler side marks for warnings and errors.
- **RSL-Completion Proposal** offers a context dependent content proposal that displays completion tips for the current cursor edit position. The tip is valid in a determined range of characters and reduces its search space while the user is typing the initially wanted keyword or import. At each moment, the user can specify graphically which option is the more adequate.
- **RSL-PartitionScanner** allows the definition of context areas in the requirements document, supporting different behaviors according with the current active area, such as syntax-highlighting, annotations, and auto-completion proposals.
- **RSL-StructureScanner** corresponds to the definition of requirements document's structural keywords (those that define the organization of a requirements document in terms of its sections), to allow syntax-highlighting rules for each of them.
- **RSL-RequirementScanner** offers a RSL-StructureScanner's similar behavior for the requirements free-form text.
- **RSL-TextHover** provides the hover support for the PIT-Studio/RSL text editor.

Finally, there are two other packages of importance in this architecture, the **RSL-to-RDF/OWL** and **Jena .NET Port**. The second package represents a .NET port of Jena framework. This framework provides the PIT-RSL plug-in with the knowledge-

base and inference-engine capabilities, typical of a natural language parsing tool. The first contains the adapter pattern code that provides a clean C# API for using the .NET ported Jena without the necessary traces of java syntax code.

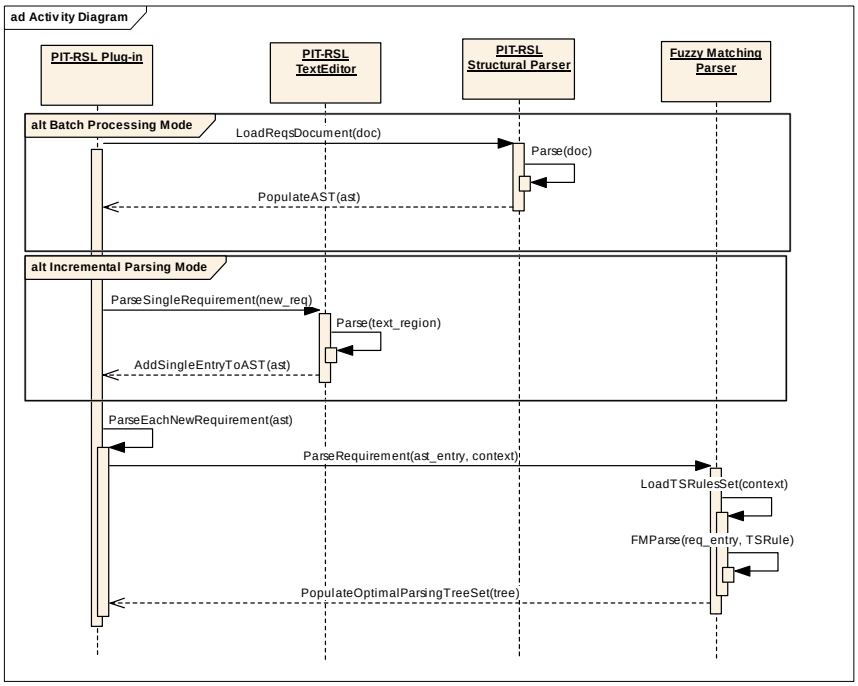


Figure 5 – Sequence of steps involved in the parsing process of a requirements document

As the messages exchange sequence represented in Figure 5 shows, there are several steps involved in the parsing process of a requirements document. In the beginning, the Eclipse .NET environment is launched, together with the ProjectIT-RSL plug-in. The diagram emphasizes that there are two types of parsing modes: the batch mode and the incremental mode. The later provides a rich set of on-the-fly validation features to aid the user during the requirements specification activity because it provides feedback while the user is typing. This parser is deeply interconnected with the **ProjectIT-RSL TextEditor** component, since it uses its mechanisms for graphical presentation of the generated warning and errors during the incremental parsing mode process. The batch mode is useful for importing existing requirements documents or for using the RSL language without the ProjectIT-RSL specialized text editor. It can be used in a stand-alone version of the RSL without any support of the base framework, the ProjectIT-Studio.

The early transformation stages performed by the referred parsing processes manipulate each requirement statement in the following order: (1) typographical and format transformations (including putting all text in one line, normalization of white

spaces, and word contractions expansion); (2) morphological syntactic analysis, which include the text annotation with the relevant part-of-speech tags.

After the conclusion of one of these alternative parsing modes processes, which are essential to analyze the document's structure and normalize the requirements document text, the ProjectIT-RSL plug-in iterates over the abstract syntax tree previously produced and delegates on the fuzzy-matching parser the responsibility of finding the optimal parsing tree for each of the parsed requirements according to the ealier identified requirement's context. The context identification allows the fuzzy-matching parser to load only the relevant template substitution (TS) rules. Then, for each rule in the retrieved set of TS rules, the fuzzy-matching parser recursively calls itself until it finds the optimal parsing tree for that requirement by exaushtion of TS rules set or timeout. The process continues, while there are new requirements to parse. The result is stored as a set of domain-based parsing trees, one for each requirement.

Fuzzy Parsing is a parsing technique that goes further than the traditional compiler-based parsing techniques. It offers several advantages in the context of natural language processing since it provides a more flexible, robust, and configurable approach for matching predefined patterns without the syntactic strictness of usual programming language's compilers. By defining patterns' templates, it is possible to discover those whose match best suits the implicit free-form natural language text semantics, due to the semantic adherence to the syntactic structure of Natural Language sentences.

Although not shown in the above diagram (because they are not yet completely implemented), the information generated by the Fuzzy Matching Parser is stored in a knowledge base. This knowledge base is later analyzed by an inference engine, with the goal of extracting implicit knowledge from the initial requirements. The idea is that this knowledge base, which will be implemented as a semantic network and stored as RDF/OWL (the W3C adopted language for data specification and representation, with more information available in http://www.w3.org/TR/owl-features/), will be shared by ProjectIT-MDD, to extract the information needed to generate code. This knowledge base will be used in the future by specialized components that will display the information according different views, for example, showing diagrams (UML or others), or converting it to XMI to share with external modeling tools. To implement the semantic network used as the knowledge base and inference engine, we have chosen the Jena project (http://jena.sourceforge.net/), a Java framework that provides a solid environment to define and manipulate concepts via RFF/OWL data specification. To convert Jena to Visual Studio .NET, we have used IKVM.NET (http://www.ikvm.net/), a Java Virtual Machine for the .NET runtime.

## 7. 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 (Abbot, 1983) proposed that nouns could be used to identify classes, adjectives to identify attributes, and verbs

to identify methods. OICSI is a tool developed by Rolland and Proix (Rolland, Proix, 1992), to help the identification of requirements from natural language text and available domain knowledge. Attempto Controlled English (ACE), first described in (Fuchs, Schwitter, 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.

The use of parsing techniques to elaborate a conceptual model from natural language requirements is a common approach; in (Macias, Pulman, 1993) 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, et alt. 2000, Fantechi, et alt, 2002), while others are concerned with the elaboration of requirements documents without such problems (Ben Achour, 1998, Denger, 2002). 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, Kamsties, 2003)

NL-OOPS (Mich, Garigliano, 1999) and LIDA (Overmyer, Lavoie, Rambow, 2001) are systems that process natural language requirements to construct the corresponding object-oriented model. A similar system is described in (Nanduri, S., Rugaber,, 1996). RML (Greenspan, Mylopoulos, Borgida, 1982) and Telos (Mylopoulos, Borgida, Jarke, Koubarakis, 1990)

The use of formal methods for specification tasks has been carefully reviewed by Axel Lamsweerde in (Lamsweerde, 2000). Reubenstein and Waters (Reubenstein, Waters, 1991) presented an approach for specifying requirements in LISP using a natural language style. 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, Moreira, 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, Proix, 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, Morant, Moreno, 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.

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, Nuseibeh, 2002). The project, called CIRCE, has some similarities with the one described in the present article, uses natural language as the specification language and provides feedback to the user with a multiple-views approach (Ambriola, Gervasi 2003). They also use fuzzy matching domain-based parsing techniques to extract knowledge from

requirements documents, which is later used to provide different views and models to analyze this knowledge.

## 8. Conclusions and Future Work

In this paper we describe the implementation steps involved in a new controlled natural language for requirements specification, supported by tools that immediately validate the written requirements, and integrated with Model Driven Development approaches, which add modelling and 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 that immediately eliminate any errors in the written documents (some previous initiatives lacked tool support from the beginning of the specification process), (3) a strong support for requirements reuse (following a similar approach as the one described in (Cybulski, 2001)), (4) a complete integration with Model Driven Development tools for model and code generation, which is fundamental for the automation of tasks and traceability between artefacts of the software development process, and (5) the use of a sequence of parsing and knowledge extraction steps that validate the written requirements.

In the near future we will concentrate in the development of the requirements reuse mechanisms and in advancing tool support. For example, we want to automate the generation of the set of validation rules, and we will develop plug-ins to show, in different formats, the information stored in the knowledge base. In a further iteration, it is our intention to include tool 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, Lamsweerde, 2002), and also integrating ideas from i* (Yu, Mylopoulos, 1998) and from a method of elaborating prototypes from requirements, proposed in (Martínez, Estrada, Sánchez, Pastor, 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.

## References


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

Ambriola, V., Gervasi, V., The Circe approach to the systematic analysis of NL requirements, Technical Report TR-03-05, University of Pisa, Informatics' Department, 2003.

Bell, T., Thayer, T., Software requirements: Are they really a problem?, Proceedings of the 2nd International Conference on Software Engineering, pp. 61-68, 1976

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

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

Fuchs, N., Schwitter, R., Attempto Controlled English (ACE), CLAW 96, First International 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 International. Conference on SE, Tokyo, 1982

Jones, C., Systematic Software Development Using VDM, Prentice Hall, New Jersey, 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

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

Saraiva J., Relatório Final de Curso – Desenvolvimento Automático de Sistemas, IST, July 2005

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 2004 Satellite Activities, Lisbon, Portugal, October 2004

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, 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

Yu, E., Mylopoulos, J., Why goal-oriented requirements engineering, Proceedings of the 4th REFSQ, pp. 15–22, Pisa, Italy, 1998