A linguistic patterns approach for requirements specification [€]

Carlos Videira, David Ferreira, Alberto Rodrigues da Silva
INESC-ID, Universidade Autónoma de Lisboa, Instituto Superior Técnico
cvideira@acm.org, david.ferreira@inesc-id.pt, alberto.silva@acm.org

Abstract

Despite the efforts made to overcome the problems associated with the development of information systems, we must consider that it is still an immature activity, with negative consequences in time, budget and quality. One of the root causes for this situation is the fact that many projects do not follow a structured, standard and systematic approach, like the methodologies and best practices proposed by Software Engineering. In this paper, we present a requirements specification language, called ProjectIT-RSL, based on the identification of the most frequently used linguistic patterns in requirements documents, written in natural language. To guarantee the consistency of the written requirements and the integration with generative programming tools, the requirements are analyzed by parsing tools, and immediately validated according with the syntactic and semantic rules of the language.

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. 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 [3].

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. A classical definition from Kotonya says that a "requirement is a statement about a system service or constraint" [13]. Software requirements are normally described using a requirements specification language.

This paper describes the rules of a new requirements specification language, called ProjectIT-RSL, and in particular how they are applied to analyze the requirements documents. Section 2 presents an overview of the ProjectIT research program and section 3 gives an idea of the components involved in the architecture developed. Section 4 describes the rules we have defined for ProjectIT-RSL. Section 5 presents related work of previous initiatives and section 6 justifies our perception that this proposal has some innovative contributions and presents the future work.

2. The ProjectIT initiative

The Information Systems Group of INESC-ID (http://gsi.inesc-id.pt/) 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 [24], 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, and is supported by a set of tools called ProjectIT-Studio workbench.

[€] 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*).

ProjectIT-Requirements [26] is the component of the ProjectIT architecture that deals with requirements issues. The main goal of this project is to develop a model for requirements specification, which, by raising their specification rigor, facilitates the reuse and integration with development environments driven by models.

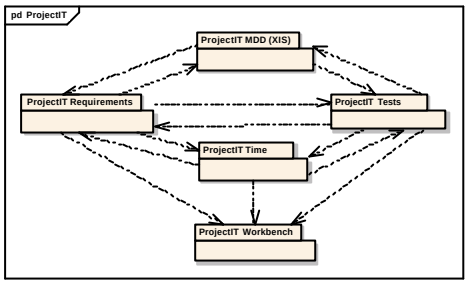

Figure 1 – ProjectIT functional architecture

After having defined the goals of the project [27], we identified and developed its main components, the **Requirements Specification Language**, ProjectIT-RSL, and a set of tools, which include the **Requirements Editor** and a number of components used to validate the requirements. 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 rules.

3. ProjectIT-Requirements architecture

Upon a common base platform for all ProjectIT tools, called **Eclipse.NET** [22] (a migration of the Eclipse platform to the .NET environment), we have developed the base components needed to support the writing and validation of requirements in ProjectIT-RSL. This toolset is called **PIT-Studio/RSL** and its technical architecture is represented in Figure 2.

The **text editor**, which is essentially a plug-in built upon the capabilities of Eclipse.NET, is represented in the diagram by the package **PIT-RSL Plug-in**. This package encompasses all the other inner packages that encapsulate specific text editor's behavior, such as auto-complete, auto-format, warnings and errors annotations (text underline and vertical bars marks), syntax-highlighting, sections folding, content outline view (synchronized with text), templates import view, optimal parsing trees view, and suggestions.

We have developed a **number of parsers**, used for analyzing the requirements text, and represented by the package **PIT-RSL Parser** which contains two specialized parsers: (1) **RSL-Structural Parser (SP)**, a CSTools (available at http://cis.paisley.ac.uk/crow-ci0/) generated parser, is responsible for performing the initial parsing stage analysis that is essentially associated with the requirements document structure, such as the document's sections structure and format/typographical normalization of the free-form requirements text in order to smooth the next parsing process stages. (2) **PIT-RSL Fuzzy Matching Parser (FMP)** is responsible for processing Natural Language (NL) text. Its goal is to find the optimal parsing tree, by successive testing NL patterns contained in a template-substitution set of rules (TS rules). These pattern-based comparisons are made against the NL requirements and every time that a valid match occurs, the FMP performs the respective substitutions in a controlled recursive manner, until no more NL patterns can be applied.

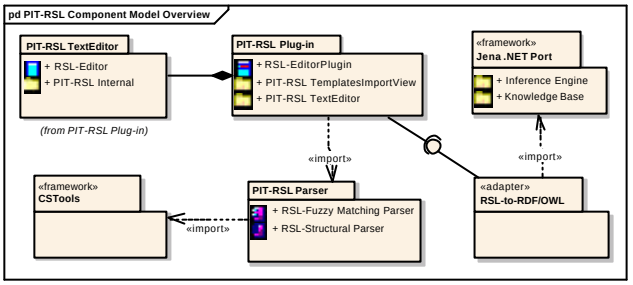

Figure 2 – ProjectIT-Requirements architecture

Finally, and to allow further knowledge inference capabilities, important for requirements validation, **PIT-Studio/RSL** uses two other packages: **RSL-to-RDF/OWL** and **Jena .NET Port**. The former contains the adapter pattern code that provides a clean C# API for using the .NET ported Jena framework without the necessary traces of java syntax code. The Jena .NET Port package represents a .NET port of Jena framework (available at http://jena.sourceforge.net/), which supplies the PIT-RSL plug-in with knowledge-base and inference-engine capabilities, typical of a NL parsing tool.

4. ProjectIT Requirements Specification Language

To define ProjectIT-RSL we followed a simple approach. First, we analysed the format and structure of the requirements documents 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. From these patterns, we determined not only the main concepts used in requirements specification, but also how they are structured and organized, including how they relate with each other and how there are combined into wider scope blocks. We derived a metamodel of the concepts identified, which is also the base of a profile (called XIS [23]), common to all our tools. Based on the patterns identified, we defined the syntax of ProjectIT-RSL, which was tested in a prototype developed using the features provided by Visual Studio .NET and the .NET Framework [6].

4.1. ProjectIT-RSL Concepts Metamodel

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 pattern, where a *subject* performs an *operation* (expressed by a verb), which affects an *object*. Other more elaborated sentences can also include a condition, which describes a constraint about a simpler fact. Another frequently occurring pattern specifies the attributes that define an entity; these are mainly declarative sentences, whereas the previous ones are imperative.

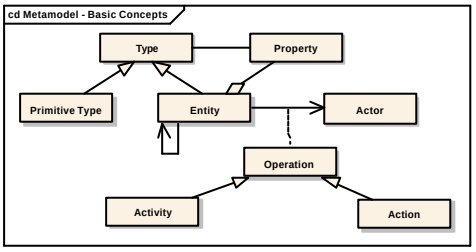

Figure 3. ProjectIT-RSL metamodel– basic concepts

The analysis of patterns led to the identification of the main concepts of our language (the ontology), represented in Figure 3:

- **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.

4.2. ProjectIT-RSL Organization Rules

One of the best practices in requirements specification is the adoption of a standard structure in requirements documents. This not only eases their reading by different people, but also allows a simpler implementation of different validation techniques. That is why our requirements documents are organized in specialized sections, which group related sentences in terms of their goals; for example, we separate the definition of the business domain entities from the specification of functional requirements.

ProjectIT-RSL allows the definition of different "application units": (1) reusable components, which can be specified independently, and integrated in broader systems; (2) complete executable systems, that can "include" some of the previous ones (reusing their functionality); (3) architectural templates and application templates, which allow pattern reuse and instance reuse, respectively. A requirements document can include many application units that can benefit from the features declared in other application units, defined within the first one, in the same document, or even located externally. These visibility rules enable the validation if all needed subsystems are defined.

The idea of the architectural templates is to provide a mechanism to describe abstract information (such as the generic attributes of every document or of an entity), or even the actions involved in the execution of an operation (for example, in managing a concept, such as a product or client, there is always the need to create, delete and update individual records). These templates allow pattern reuse, providing the user with a number of explicit abstract edit points that he must define/override; these edit points are presented to the user by the auto-complete feature of the text editor, when an architectural template is inherited. The application templates provide instance reuse by means of using content copy mechanisms and allowing operation overriding.

The rules expressed below in EBNF notation are the most basic ones, as they abstract the structure of our requirements document.

```
<Requirements Document> : [<Introduction Section>]
    <Application Unit>*
<Application Unit> : <Section>*
<Section> : <Sentence>*
```

4.3. ProjectIT-RSL Sentence Rules

The sentences used in our requirements documents can be divided in two groups, declaration and definition sentences; definition sentences express the information about a concept (such as an entity, Entity Definition), whereas declarations sentences just name a concept, associating it with a specific type (which is what happens in an Operation Declaration).

```
<Sentence> : <Declaration> | <Definition>
<Declaration> : <Application Unit Declaration> |
    <Operation Declaration>
<Definition> : <Operation Definition> | <Actor
    Definition> | <Entity Definition>
```

Figure 4 is a screenshot of our requirements editor showing just the overall organization of a requirements document, in terms of sections.

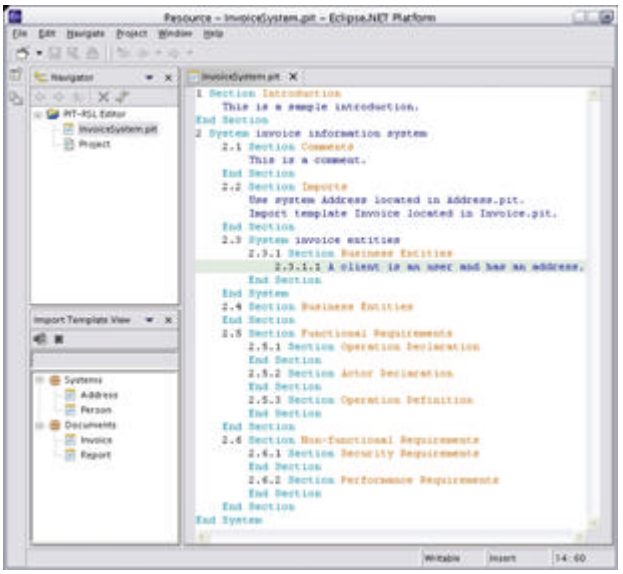

Figure 4 – ProjectIT-RSL Editor

The specification of our base concepts (Entity, Actor and Operation) is defined by the following rules, which basically state that the complete specification of a concept can be done by a number of sentences.

```
<Operation Specification> : <Operation Declaration>
    <Operation Definition>*
<Actor Specification> : <Actor Definition>*
<Entity Specification> : < Entity Definition>*
```

4.4. ProjectIT-RSL Entity Rules

The complete enumeration of all our rules is beyond the scope of this paper, but in the following sections we are going to detail some of them, justifying their existence and presenting some examples.

For example, the number of rules specifically used for validating an entity specification is already very large, as the following EBNF rules, although not complete, show.

```
<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 list allows, for example, the specification of entities using various sentences: (1) simple sentences that enumerate a list of entities' properties (pattern <Entity Property Definition>); (2) sentences that state that an entity is a specialization of another entity (pattern <Entity Inheritance Definition>); (3) sentences that basically state that two entities are the same (pattern <Entity Equivalence Definition>); (4) and sentences that define a possible complex relationship between two entities (pattern <Entity Association Definition>).

The pattern <Entity Inheritance Definition> is another technique, besides the use of templates, which supports the reuse of requirements information, by using generalization/specialization relationships; these relationships are also applied to other concepts, such as Actors or even Operations. The following sentences are examples of the possibilities we now support to specify information about an Entity; although some are written as a single sentence, a number of pre-processing parsing steps is applied to validate the inheritance and aliases relationships, and to break a single line sentence in two separate requirements sentences.

```
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.
A supplier is an entity, and has a list of bank accounts.
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 list of numbers, and by a
location.
```

To avoid the problems related with forward declaration (as we can see from the above relationship between an entity and an address), we initially consider that every property is a basic attribute of an entity, unless it has already been declared as an entity. This information is maintained in a context dependent table; if below the entity definition is defined another entity with the same name as a previous attribute, all occurrences of the previous attribute are substituted, changing their type from FIELD to ENTITY.

It is also interesting to see how we have dealt with the specification of n-to-n association relationships between two entities, using the patterns <Entity Active Association Definition> and <Entity Passive Association Definition>. The solution was the use of two distinct requirements sentences, one in active and the other in passive voice, like in the following example, and from the analysis of such sentences, the parsers detect a n-to-n association and create (if necessary) an association entity from the relationship.

```
Each teacher can teach many courses.
Each course can be taught by many teachers.
```

4.5. ProjectIT-RSL Actor Rules

The definition of an actor is accomplished by using a number of similar patterns, and the available rules are very simple.

```
<Actor Definition> : <Actor Inheritance Definition> |
    <Actor Operation Definition> | <Actor Equivalence
    Definition>
<Actor Inheritance Definition> : < Actor> is an <Actor>
<Actor Equivalence Definition> : <Actor> is the same as
    <Actor>
<Actor Operation Definition> : <Actor> can <Operation
    Declaration>*
```

With these patterns we can write sentences that specify the actor's allowed operations, as the following example shows.

```
The general user is an Actor
The accounting director is a general user, except create
invoice.
The administrator is an Actor
A general user can create invoice, manage product, manage
client, and pay invoice.
```

4.6. ProjectIT-RSL Operation Rules

In terms of the operations, and as we have shown above with the pattern Operation Specification, normally we need to declare a list of all available operations first, which are later defined by a detailed description. The reason for this is twofold: first, it allows us to mention the operations allowed for every actor without the need to deal with forward references, and second it is also an interesting good practice to separate the system's features from their description. Many operations of an interactive system directly affect an Entity, but we must also consider the possibility of Processes that implement more complex workflows (for example, the calculation of interest rates), or of printing reports (these are the current possibilities).

```
<Operation Declaration> : <Entity Manipulation> |
    <Process> | <Report Operation>
<Entity Manipulation> : <Primitive Verb> <Entity> | <
    Primitive Verb> <Entity> <Property>
<Process> : <Verb> <Name>*
<Report Operation> : Print <Report Name>
<Report Name> : <Name>*
```

The number of patterns for the sentences involved in the definition of an operation is naturally greater than those applied for an entity. The following pattern list is thus just representative (for example, the patterns involving conditions are omitted). We must emphasize that we already have some fixed terms, and especially verbs, for which we associate a pre-defined semantics, and consequently are used for generating a pre-defined behavior in the further activity of code generation.

```
<Operation Definition> : <Input Activity> |  <System
    Activity> | <Command Activity> | …
<Input Activity> : <Data Input Activity> | <Choice
    Activity>
<System Activity> : <Validation Activity> | <Presentation
    Activity>
<Data Input Activity> : the <Actor> enters
    [<Entity>]<Property>
<Choice Activity> : the <Actor> chooses
    [<Entity>]<Property>
```

For example, our patterns currently support the specification of an operation such as the following one.

```
Create invoice is:
- The actor chooses the client name
- If the client does not exist the actor can create the
client
- The actor enters the invoice date
```

```
- The actor introduces the list of products, and for each
product
  - The actor chooses a product
  - The actor enters a quantity
  - The actor enters the unit price
  - The system calculates the product total
  - The system updates the total invoice value
- The actor confirms the invoice creation at any time
```

4.7. Template Substitution Rules

Although we want to allow the users of our tools to use natural language, the parsing mechanisms, as well as the integration with code generation tools, imply that we must restrict the terms allowed to a recognizable subset, such as the fixed terms we have seen in the above rules. This set of rules, called the TS rules, can be defined and changed for a specific project, enables the incremental evolution of these terms, just by adding more rules, or by defining synonyms between words. This approach not only supports different writing styles and natural languages, but also is the base for the definition of domain specific languages.

The TS rules set is currently manually written, but in the near future it will be automatically generated by a specialized component. This is particular important as the TS rules are stored in a XML DOM document, which becomes more difficult to handle manually as its size grows. The rules are stored in groups related to the sections they apply, and as such we have specialized business entities, functional requirements and non-functional requirements rules.

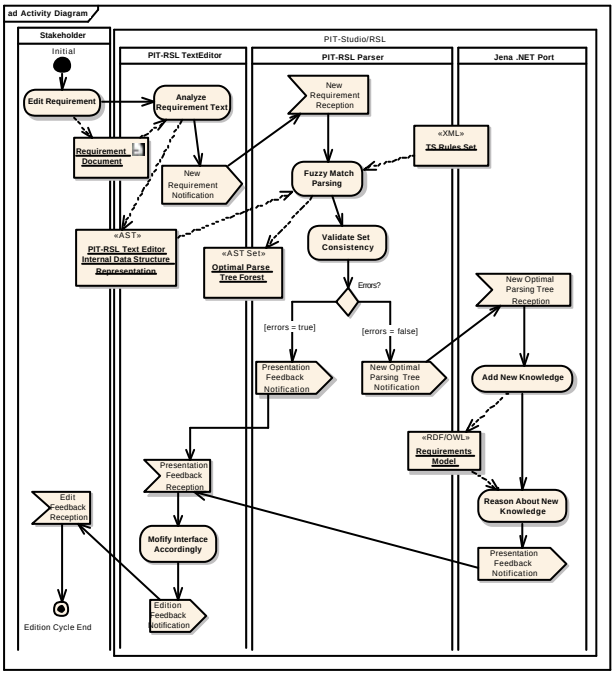

Figure 5 – ProjectIT-RSL main activities

The activity diagram presented in Figure 5 describes the requirements parsing workflow and the intervenient components. According with this diagram, the first requirements textual manipulation produces a parsing abstract syntax tree (AST), which is passed to the PIT-RSL Fuzzy Matching Parser (FMP) as input. Before starting the fuzzy match process, the FMP performs a morpho-syntactic analysis of each requirement, and after this transformation the FMP determines if some of the nouns correspond to previously identified entities.

After gathering the part-of-speech information, the FMP can start processing each textual requirement by successively trying possible matches between textual templates and each requirement. Each requirements document's section has a specific set of TS rules associated. Each time a valid match occurs, the textual fragment matched is replaced by the substitution fragment of the respective TS rule. The FMP recursively tries to find a new valid match with all the TS rules available in the current set against the modified requirement. This process ends when the FMP cannot find more matches with the available TS rules in the current set.

For example, the following sentence represents a requirement and the table below the TS rules required for the fuzzy-matching parsing of the requirement:

```
The client is a user and has a name, an address, and an
account.
```

RULE	TEMPLATE	SUBSTITUTION
TS_1	subclass/NN IS base/ENT	$NODE/ISA/ENT
TS_2	ent/ISA/ENT HAS fields/FIELD	$NODE/DEF/ENT
TS_3	field1/NN AND field2/NN	$NODE/&LIST/FIELD
TS_4	field/NN AND list/&LIST/FIELD	$NODE/&LIST/FIELD

During the search for the optimal parse tree the FMP uses a scoring mechanism for heuristically compare the solutions found. When a new parsing tree is found the FMP compares its scoring with the best parsing tree found at moment. If the new parsing tree presents a best score the FMP stores it, and discards the old one. At the end of the process the best parsing tree corresponds to the optimal parse tree. The sequence of best matches for our example is described below:

RECURSION LEVEL	APPLIED RULE	RESULT
1	TS1	The/DT **@1/ISA/ENT** and/CC has/VBZ a/DT name/NN and/CC an/DT address/NN and/CC an/DT account/NN

2	TS3	The/DT @1/ISA/ENT and/CC has/VBZ a/DT name/NN and/CC an/DT **@2/&LIST/FIELD**
3	TS4	The/DT @1/ISA/ENT and/CC has/VBZ a/DT @3/&LIST/FIELD
4	TS5	The/DT **@4/DEF/ENT**

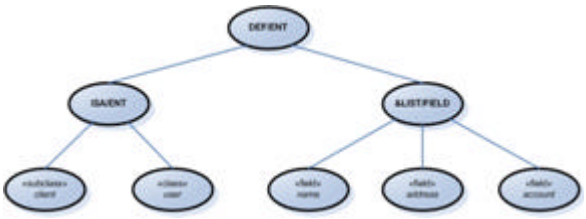

Figure 6 – An example of a generated parse tree

This sequence of matches allows the population of the Optimal Parse Tree Forest (an example of such trees is presented in Figure 6), a set of abstract syntax trees, which is the input for the RSL-to-RDF/OWL that is responsible for transforming these isolated optimal parsing trees into RDF/OWL in order to populate Jena .NET Port models for further semantic reasoning.

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 [1] 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 [21], to help the identification of requirements from natural language text and available domain knowledge. Attempto Controlled English (ACE), first described in [10], is one of those approaches that use a controlled natural language to write precise specifications that, for example, enable their translation into a first-order logic similar representation (called DRS).

The use of parsing techniques to elaborate a conceptual model from natural language requirements is a common approach; in [15] 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 [9], while others are concerned with the elaboration of requirements documents without such problems ([4],[8]). 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 [5]

NL-OOPS [16] and LIDA [19] are systems that process natural language requirements to construct the corresponding object-oriented model. A similar system is described in [18]. RML [11] and Telos [17], 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 [14]. Reubenstein and Waters [20] 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 that 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 [7].

A number of different approaches have researched on the elaboration of requirements specification using patterns of natural language. Approaches such as [4] and [21] 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 [8] 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 tried to formalize the analysis of natural language sentences in order to create a precise conceptual model [12], using linguistic patterns that are mapped in conceptual patterns.

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. The project, called CIRCE, uses NL as the specification language and provides feedback to the user with a multiple-views approach [2]. 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. Although Circe and ProjectIT-RSL have some similarities, there are between them many differences, namely in the architecture, concepts and algorithms used, and above all, in the strategy: the goal of CIRCE has been requirements validation, and only recently moved to research about the integration with model driven approaches, whereas our goal with requirements specification is to obtain a consistent requirements document that is in conformance with a metamodel, which also enables the use of model driven techniques and code generation.

6. Conclusions and Future Work

In this paper we argue that a requirements specification language is essential for the improvement of the results of information systems development, and propose a controlled natural language, supported by tools that validate the written requirements; the integration with Model Driven Development tools adds modelling and code generative features.

Although sharing points with other 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, defined by a set of rules derived from the analysis of linguistic patterns, with (2) support by tools to immediately detect any errors in the written documents (some previous initiatives lacked tool support from the beginning of the specification process), namely (3) the use of a series of parsing and knowledge extraction steps that validate the written requirements, (4) a strong support for requirements reuse, (5) 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 (6) the possibility of using a set of patterns not only for representing system requirements and their detailed description, but also for describing the system goals and user requirements.

In the near future we will concentrate in the development of the requirements reuse mechanisms and in advancing tool support. For example, we will automate the generation of the TS Rules from the ProjectIT-RSL abstract rules, and we will develop plug-ins to show, in different formats, the information stored in the knowledge base. 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

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

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

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

[4] 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

[5] 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

[6] 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

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

[8] 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

[9] 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

[10] Fuchs, N., Schwitter, R., Attempto Controlled English (ACE), CLAW 96, 1st Int. Workshop on Controlled Language Applications, Leuven, Belgium, March 1996

[11] Greenspan, S., Mylopoulos, J. and Borgida, A., Capturing More World Knowledge in the Requirements Specification, Proc. 6th Int. Conf. on SE, Tokyo, 1982

[12] 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

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

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

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

[16] 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

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

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

[19] 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
[20] Reubenstein, H., Waters, R., The requirements apprentice: Automated assistance for requirements acquisition, IEEE Transactions on Software Engineering, 17(3), pp 226–240, 1991
[21] Rolland, C., Proix, C., A Natural Language Approach for Requirements Engineering, Proceedings of the 4th Int. Conf. Advanced Information Systems, CAiSE 1992
[22] Saraiva J., Relatório Final de Curso – Desenvolvimento Automático de Sistemas, IST, July 2005
[23] Silva, A., Lemos, G., Matias, T., The XIS Generative Programming Techniques, Proc. of the 27th Annual Int. Comp. Software & Application Conference, Dallas, 2003
[24] Silva, A., O Programa de Investigação "ProjectIT", Technical report, V 1.0, October 2004, INESC-ID
[25] Videira, C., Silva, A., The ProjectIT-RSL Language Overview, UML Modeling Languages and Applications: UML 2004 Satellite Activities, Lisbon, October 2004
[26] 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
[27] Videira, C., Silva, A., A broad vision of ProjectIT-Requirements, a new approach for Requirements Engineering, in Actas da 5ª Conferência da APSI, Lisbon, November 2004
[28] Videira, C., Silva, A., Patterns and metamodel for a natural-language-based requirements specification language, in Proc. of CaiSE'05 Forum, pp. 189-194, Porto, June 2005