Obtaining Formal Requirements Representations with the RSLingo Approach

David de Almeida Ferreira, Alberto Rodrigues da Silva
INESC-ID, Instituto Superior Técnico (IST), Lisbon, Portugal
{*david.ferreira, alberto.silva*}*@inesc-id.pt*

***Abstract*—In order to carve out from the open space of possibilities the software system that the business stakeholders need and expect, it is crucial to properly document all observable and desired characteristics of the software system to be built, i.e., its requirements. In this paper we present RSLingo, an information extraction approach based on two domain-specific languages: RSL-PL and RSL-IL. The former allows the definition of linguistic patterns to enable the automatic analysis of requirements specifications written in natural language. While the latter supports the formal specification of requirements--related information that is automatically extracted. Since it enables further processing on the gathered knowledge, the RSLingo approach allows one to automatically verify some quality criteria and generate complementary representations that assist stakeholders during requirements validation.**

***Keywords*-Requirements Specification Language, Information Extraction, Requirements Modeling.**

I. Introduction

Although the body of knowledge provided by literature in Requirements Engineering (RE) is extensive [1], [2], RE activities still require a *significant human effort*. Most RE techniques are human-centric, thus having a multi-disciplinary background. Also, these techniques still lack proper tool support. Several requirements management tools exist [3], yet most RE techniques are manually applied, such as when producing documents, tables, and diagrams [4]. The human-intensive labour that these activities entail makes them time-consuming and error-prone. Thus, despite the RE field's best practices being well documented [1], [5], [6], the quality of these artifacts strongly depends on the experience and skills of the team members developing them.

It would be helpful if some of the manually performed tasks, related with requirements text analysis, could be automated for the following purposes:

- *Domain analysis:* for identifying all the relevant concepts, their relations, and how the software system manipulates them to provide the desired capabilities, while abiding to all stated constraints regarding the functionality that provides such capabilities to its users;
- *Verification:* for performing consistency checking on the extracted domain knowledge, through inference and ambiguity resolution based on glossaries and other lexical resources, to prevent delays and extra costs due to rework arising from belatedly discovered defects [2];
- *Transformations:* for automatically generating alternative requirements representations such as diagrams, tables, reports, or even template-based paraphrases of requirements statements in a controlled natural language, according to specific viewpoints. Also, an initial draft of the domain model and behavior models (e.g., UML class diagrams and UML use case diagrams, respectively) can be produced and provided as an input to software development processes that follow the Model-Driven Engineering paradigm [7].

Recognizing that natural language text is the preferred and recommended medium for documenting requirements [8], and that *ambiguity* is an intrinsic characteristic of natural language [9], we consider that there are two main approaches to deal with this necessary nuisance within the field of RE: (1) following a *controlled natural language approach* [10], in which language ambiguity is removed by forcing users to write according to an unambiguous predefined syntactical structure and a limited vocabulary; or (2) adopting a more flexible and extensible *information extraction approach* [11] based on linguistic patterns where, despite ambiguity not being completely eradicated, only the most plausible interpretation is considered for further processing, according to a best-fit match with the linguistic patterns recognized.

In this paper we overview RSLingo, an Information Extraction [11] approach for automatically analyze and formally specify requirements written in natural language. The novelty of RSLingo arises from its strategy of using two distinct domain-specific languages: RSL-PL and RSL-IL. Each of these languages addresses a specific aspect regarding the concerns that must be taken into consideration while formally documenting requirements. The former addresses the definition of common linguistic patterns of textual requirements representations. The latter was designed to formally convey RE-specific information, thus enabling further processing of this information. Through the mapping process between RSL-PL and RSL-IL, RSLingo aims to improve the quality and rigor of requirements specifications written in *ad hoc* natural language through complementary representations that can be used to validate the specified requirements.

The structure of this paper is as follows. Section II introduces the RSLingo approach, namely the main processes, roles, and the advantages of using a multi-language strategy. Section III discusses the strengths and limitations of the RSLingo approach, by comparing it with related work. Finally, Section IV concludes this paper and lays down some ideas for future work.

II. The RSLingo Approach

Natural language is the most common form of requirements representation [12], [8] used within requirements documents, requirements repositories (e.g., databases), and even diagrams[1]. Thus, we advocate that, in order to further benefit from the effort in developing requirements specifications in natural language [5], [6], suitable languages and supporting toolsets are mandatory to better and properly support requirements authoring and validation activities [1].

The first step towards the automation of requirements analysis is treating requirements' textual representations according to a "white-box" approach in order to extract and analyze their meaning. To this end, while not forcing stakeholders to adopt a new notation, we propose an information extraction approach based on simplified Natural Language Processing (NLP) techniques, such as *chunk parsing* [13]. The aim of chunk parsing is not to enforce grammatical correction, but to exploit the alignment between the structure of sentences and their semantics (i.e., the meaning of requirements). Besides requiring less computational power, chunk parsing is more flexible and robust while allowing one to focus on pieces of information (i.e., the chunks) that carry relevant requirements information. Therefore, despite being less complex than the traditional full parsing approach, these techniques still enable us to obtain the required domain knowledge for a deeper insight on the system to be built. These simplifications exploit the syntactic–semantic alignment imprinted in linguistic patterns [13].

Considering this premise, we named our approach *RSLingo*. The name stems from the paronomasia on "RSL" and "Lingo". On the one hand, "RSL" (Requirements Specification Language) emphasizes the purpose of formally specifying requirements. The language that serves this purpose is *RSL-IL*, in which "IL" stands for *Intermediate Language*. On the other hand, "Lingo" emphasizes that its design has roots in natural language, which are encoded in linguistic patterns used during information extraction from the textual requirements specification [11]. The language designed for encoding these RE-specific linguistic patterns is *RSL-PL*, in which "PL" stands for *Pattern Language*.

The RSLingo approach considers two distinct stages: definition at *process-level* and usage at *project-level*. Process-level, depicted in Figure 1, comprises the definition (or adaptation) of the linguistic patterns encoded in RSL-PL, and also establishing the mapping between these linguistic patterns and the semantically equivalent RSL-IL formal structures. On the other hand, as illustrated in Figure 2, project-level consists in applying its languages and using the RSLingo's toolset during a specific software project.

1) ***RSLingo's Process-Level****:* as illustrated in Figure 1, it considers an unusual role, which we named *Requirements Architect*. As the adopted name suggests, the person performing this role must have a thorough knowledge and vast experience in RE, to be able to tailor the RSLingo approach to the project at hand. Also, the Requirements Architect must be able to identify common linguistic patterns that frequently occur in natural language requirements specifications. The expertise of those playing this role is a crucial factor for the success of the RSLingo approach. However, after this tacit knowledge is encoded into the mapping between RSL-PL linguistic patterns and RSL-IL formal structures (the main asset to be produced), this knowledge can be reused in similar projects multiple times.

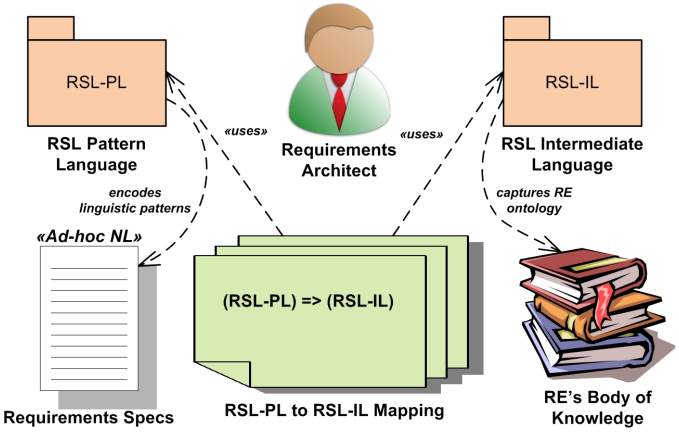

Figure 1. Overview of the RSLingo approach at process-level.

2) ***RSLingo's Project-Level****:* as illustrated in Figure 2, the approach is not disruptive with regard to traditional RE approaches [1]. During the requirements specification activity, we consider two main roles: the *Requirements Engineer* and the *Business Stakeholder*. RE is social and collaborative by nature, thus we value the direct contribution of Business Stakeholders. Those playing the role of Business Stakeholder must be acquainted with the problem-domain, which makes them valuable requirements sources and thus crucial during the requirements validation activity [14]. Therefore, the RSLingo approach encourages Business Stakeholders to directly author requirements themselves. Within this collaborative environment, the purpose of the Requirements Engineer role is to facilitate the requirements specification process and help Business Stakeholders to discover their *real* requirements [2].

According to the information flow represented in Figure 2, both the Requirements Engineer and the Business Stakeholder roles contribute with textual inputs to the requirements specification (written in natural language), which are depicted as `Requirements Specs` with the `ad hoc natural language` stereotype. Additionally, these stakeholders should follow RE's best practices of maintaining a project-specific `Glossary`. This sort of structured dictionary is of paramount importance because it establishes

[1]For a matter of simplicity, we henceforth refer to all of these types of requirements representation media as "*requirements specifications*".

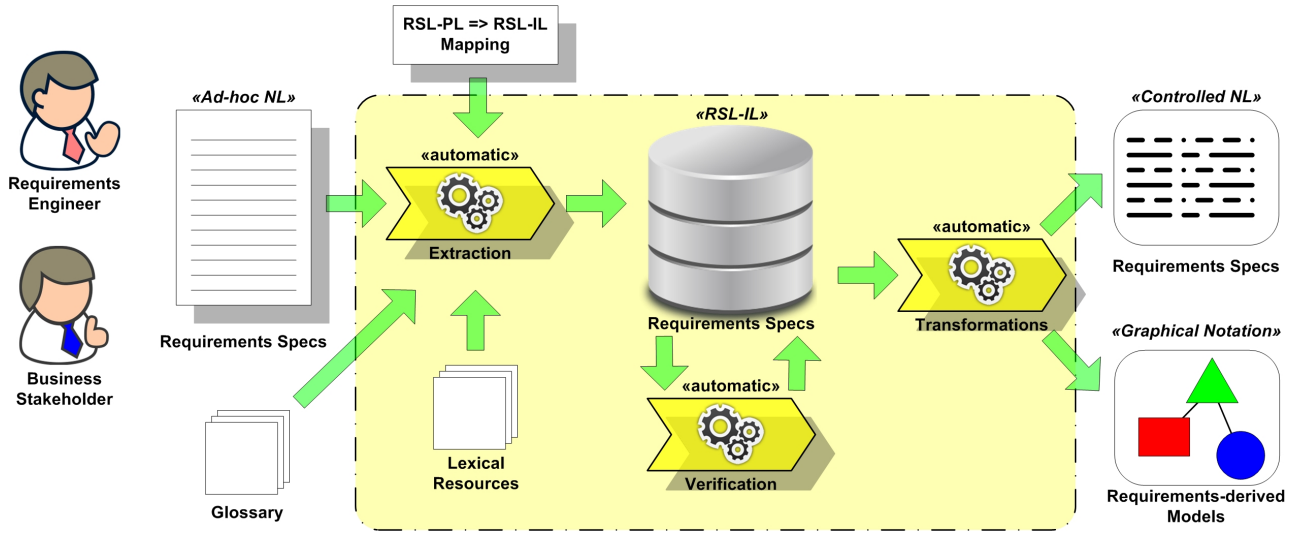

Figure 2. Overview of the RSLingo approach at project-level.

a common vocabulary for key problem-domain terms, which should be consistently used in a crosswise manner throughout the project, since they help all stakeholders to reach a shared understanding of those terms. Finally, the remaining manually created artifact to be considered by the RSLingo approach, required as an input to the RSLingo's toolset, is the `RSL-PL => RSL-IL Mapping` defined by the Requirements Architect at RSLingo's Process-Level.

Besides the sentence-level *syntax–semantic alignment* addressed by chunk parsing techniques, one must also take into consideration the word-level *lexical–semantic alignment*. Thus, to circumvent the lack of world knowledge, and in addition to the three manually made artifacts previously mentioned, the RSLingo toolset requires other `Lexical Resources`, such as WordNet[2] and VerbNet[3] lexical databases. These `Lexical Resources` are needed because RSLingo deals with *ad hoc* natural language. Without these word-level resources it would be hard to "fully understand" requirements, namely to support disambiguation tasks during the automatic extraction process, and provide additional information on the meaning of terms and their lexical relationships. However, in order to enable some flexibility regarding the definition of project-specific terms, the additional information provided by these `Lexical Resources` is overridden by the terms defined in the previously mentioned `Glossary` artifact.

In a nutshell, the RSLingo toolset works as follows: when a best-fit match occurs between a linguistic pattern (defined in RSL-PL) and a textual requirements representation, the captures identified during the match provide relevant information considering the syntactic–semantic alignment of the recognized linguistic patterns. After yielding a best-fit match, a translation from each of the match's captures to the semantically equivalent RSL-IL formal structures takes place, as described in Information Extraction literature [11].

[2]http://wordnet.princeton.edu/
[3]http://verbs.colorado.edu/verb-index/

III. Discussion

RSLingo clearly contrasts with the common practice of only dealing with the organization of requirements (e.g., document structure and requirements relations) and metadata (e.g., requirements attributes) in databases or diagrams – the "black-box" approach –, around which academia and industry seem to have a greater concern. We regard documentation best practices as being of the utmost importance [6], yet computers should be able to automatically extract relevant information pertaining to the applicational domain of the system to be built, which is captured through the semantics of words and phrases of their textual representations.

For addressing this matter, an alternative would be using Controlled Natural Languages (CNL), such as Attempto Controlled English (ACE) [10]. CNLs are subsets of natural languages whose grammars and vocabularies have been engineered in order to reduce or eliminate both ambiguity (to improve computational processing) and complexity (to improve readability). By completely avoiding ambiguity, CNLs can be mapped to structures equivalent to First-Order Logic formulas. CNLs are typically designed for knowledge engineering, thus not addressing behavioral aspects as required in RE. Additionally, they are *easy to read*, yet *hard to write*, because writing with CNLs resemble programming with a high-level programming language with strict grammar rules, hence it is easy for untrained users to become frustrated while using it. Therefore, CNLs require significant effort and specialized tools (such as predictive editors) for creating (or adapting) language-compliant specifications.

While recognizing the importance of controlled natural languages within other fields, the obstacles they pose within RE's collaborative work environments are serious impediments to their adoption by untrained users. These limitations prevent domain experts from directly contributing with their own requirements text, which is why we propose a flexible linguistic approach based on Information Extraction [11].

RSLingo follows an approach similar to the CIRCE project [15]. CIRCE provides a "lightweight formal method", supported by model-checking techniques, to produce a validation of the requirements specifications written in natural language, and its toolset provides feedback to the user with a multi-perspective approach. To this end, CIRCE uses fuzzy-matching parsing to extract knowledge from requirements specifications, which is then used to generate different views to analyze the information captured from requirements text.

However, the RSLingo approach goes further in the conceptual distinction between the two different concerns to be considered: defining linguistic patterns and formally specifying requirements knowledge. This separation of concerns is addressed by two domain-specific languages: RSL-PL and RSL-IL, respectively. The higher abstraction provided by these two languages makes them more user-friendly for those playing the Requirements Architect role. Also, this separation of concerns allows one to evolve the set of recognized linguistic patterns in RSL-PL while maintaining the requirements knowledge already encoded in RSL-IL specifications. Being a fixed language, RSL-IL provides a sound and stable knowledge base of requirements specifications, supporting reuse of specifications independently of RSL-PL extensions. Overall, these two languages (and the mapping between them) provide a requirements specification approach – RSLingo – endowed with extensibility and reusability mechanisms.

IV. Conclusion

In this paper we overview RSLingo, a linguistic approach to extract and formally specify RE-specific information from requirements specifications written in natural language. RSLingo follows a multi-language strategy, since its operationalization relies on two purpose-specific languages, RSL-PL and RSL-IL, and the mapping between them.

Unlike other requirements specification approaches, which treat requirements representations as "opaque entities", the RSLingo approach allow us to gain a deeper understanding of the system to build through NLP techniques.

As future work, we plan to conduct laboratory-controlled case studies to validate the RSLingo approach, because we still need to evaluate it according to the users' perspective. The next step in this research roadmap is to exploit the potential of RSLingo in producing alternative representations from requirements specifications formally encoded in RSL-IL, besides textual requirements paraphrases.

References

[1] K. Pohl, *Requirements Engineering: Fundamentals, Principles, and Techniques*, 1st ed. Springer, July 2010, ISBN-13: 978-3642125775.

[2] R. Young, *The Requirements Engineering Handbook*, 1st ed. Artech Print on Demand, November 2003, ISBN: 978-1580532662.

[3] INCOSE, "Requirements Management Tools Survey," Retrieved Monday 12th March, 2012 from http://www.incose.org/ProductsPubs/products/rmsurvey.aspx.

[4] E. Gottesdiener, *The Software Requirements Memory Jogger: A Desktop Guide to Help Software and Business Teams Develop and Manage Requirements*, 1st ed. Goal / QPC, October 2009, ISBN-13: 978-1576811146.

[5] IEEE Computer Society, "IEEE Recommended Practice for Software Requirements Specifications," *IEEE Std 830-1998*, August 1998.

[6] B. Kovitz, *Practical Software Requirements: Manual of Content and Style*. Manning, July 1998.

[7] D. C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer*, vol. 39, no. 2, pp. 25–31, February 2006.

[8] H. Foster, A. Krolnik, and D. Lacey, *Assertion-based Design*. Springer, 2004, ch. 8 - Specifying Correct Behavior.

[9] D. C. Gause and G. M. Weinberg, *Exploring Requirements: Quality Before Design*. Dorset House Publishing Company, Incorporated, September 1989.

[10] N. Fuchs, U. Schwertel, and R. Schwitter, "Attempto Controlled English – Not Just Another Logic Specification Language," in *Logic-Based Program Synthesis and Transformation*, P. Flener, Ed., Eighth International Workshop LOPSTR'98. Manchester, UK: Springer, June 1999, pp. 1–20.

[11] H. Cunningham, "Information Extraction, Automatic," in *Encyclopedia of Language & Linguistics*, 2nd ed. Elsevier, 2006, vol. 5, pp. 665–677.

[12] A. Davis, *Just Enough Requirements Management: Where Software Development Meets Marketing*, 1st ed. Dorset House Publishing, May 2005.

[13] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 1st ed. O'Reilly Media, June 2009, ISBN-13: 978-0596516499.

[14] K. Pohl and C. Rupp, *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam – Foundation Level – IREB compliant*, 1st ed. Rocky Nook, April 2011, ISBN-13: 978-1933952819.

[15] V. Ambriola and V. Gervasi, "On the Systematic Analysis of Natural Language Requirements with CIRCE," *Automated Software Eng.*, vol. 13, no. 1, pp. 107–167, January 2006.