RSLingo: An Information Extraction Approach toward Formal Requirements Specifications

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

***Abstract*—Requirements Engineering (RE) is about achieving a shared understanding about the software system to be built. No withstanding the importance of other RE activities, requirements specification deserves special attention due to its documentation purposes: *to communicate requirements, someone has to write them down*. In this paper we present RSLingo, a linguistic approach for improving the quality of requirements specifications, which is based on two languages and the mapping between them: RSL-PL, an extensible language for dealing with information extraction from requirements written in natural language; and RSL-IL, a formal language with a fixed set of constructs for representing and conveying RE-specific concerns. Contrarily to other approaches, this decoupling allows one to deal with requirements as "white-box" items, enabling a deeper understanding at a semantic level. Thus, RSLingo enables the automation of some verification tasks that prevent common requirements quality problems and lays the foundation to better integrate RE with the Model-Driven Engineering paradigm through transformations of requirements representations into design models.**

***Keywords*-Requirements Specification Language; Information Extraction; Requirements Modeling; Transformations.**

I. Introduction

In the field of engineering it is essential to have a thorough understanding of the problem-domain at hand before starting any sort of effort toward its solution [1]. To achieve a cost-effective solution, one must start preventing rework costs from the beginning. The intangible nature of software emphasizes the paramount importance of Requirements Engineering (RE) activities within the field of Software Engineering (SE) [2]. The success of software-intensive system development projects strongly relies on proper documentation, by using textual specifications augmented with models of both the system to be built [1] and the environment in which it will be used (i.e., its application domain) [3].

In short, RE is an important discipline because rework costs and schedule delays due to defects are expensive to fix [1], specially at more abstract levels (such as the requirements-level) where misinterpretations affect fundamental aspects such as *what system is to be built* and its *fitness-for-purpose* according to an end-user perspective.

Despite the RE field's best practices being well documented [4], [5], [6], the *quality* and *proper maintenance* of these artifacts can significantly vary, depending on the expertise of the development team members producing these artifacts, not to mention other influencing factors such as schedule, cost, and availability constraints.

Additionally, in order for one to achieve a common understanding on the system to be built, a great deal of RE effort is about effective communication [1], [3] (i.e., enforcing that everyone shares the same mindset, despite the different individual experiences and backgrounds). To this end, it is important that *everyone communicates by means of a common language* [1]. Considering its purpose, there are some desirable characteristics that such language should exhibit, namely it should be *expressive enough* to describe the problem-domain at hand, yet sufficiently *clear and precise* to avoid ambiguity. While expressiveness is crucial for its practical usage and business stakeholder adoption, the latter are fundamental to enable any sort of computational support upon requirements specifications.

Regarding the choice of a language that is suitable for requirements documentation, we consider (as many other authors [1], [7]) that the best approach for specifying requirements is to write them down as natural language text. This approach provides *enough expressiveness* and simultaneously circumvents *familiarity* and *proficiency* problems with regard to the notation used to specify requirements. These problems are often pointed out as major flaws of approaches that force business stakeholders (e.g., client representatives and end-users) to learn and use new notations, specially those with a higher level of *formality* [1]. Computer-savvy stakeholders, such as developers, might prefer such (semi-) formal notations, yet the majority of business stakeholders, in which the domain experts are included, prefer the *universality* of natural language-based specifications [1].

However, the usage of natural language for documenting requirements usually gives rise to software requirements specifications with well-known problems, such as ambiguity, incompleteness, inconsistency, and incorrectness [2], [4]. Ambiguity is a particularly serious problem [8], because it allows the coexistence of multiple interpretations for the same textual requirement representation [9], which undermines both the goals of achieving a shared vision of the system being built and enabling any sort of further computation upon requirement specifications. In turn, these facts sustain the claim that the quality of RE deliverables

978-1-4673-4389-3/12/$31.00

still strongly depend on human-intensive labour and on the expertise of business analysts and requirements engineers.

In this paper we argue that an *Information Extraction* approach [10], based on linguistic patterns that are frequently used to express RE-specific concerns, can be followed to improve the quality and rigor of requirements specifications written in *ad-hoc* natural language[1]. This approach, called RSLingo, relies on two languages: while one supports the definition of linguistic patterns that commonly appear in requirements specifications, the other enables the formal specification of requirements concerns conveyed through textual requirements representations that follow those linguistic patterns. By establishing a mapping between these two languages, when a match between textual requirements representations and the recognized linguistic patterns occurs, one can automatically produce formal representations of these requirements specifications, by generating structures (with the formal language's constructs) that are semantically equivalent to the matched linguistic patterns. Through this mapping, we can automatically obtain machine-processable representations that, in turn, can be further used as input for other kinds of automated tasks. For instance, to automatically perform requirements verification through the assessment of some requirements quality criteria (e.g., consistency) and to automatically generate alternative representations (e.g., diagrams and tables) that may be used to support requirements validation or even a seamless integration with the Model-Driven Engineering (MDE) paradigm [11].

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 illustrates how the two languages of RSLingo work together and further explains some of the concepts underneath their constructs through a case study. Section IV discusses the strengths and limitations of this approach, based on related work. Finally, Section V concludes this paper and lays down some ideas for future work.

II. The RSLingo Approach

Natural language is the most common and preferred form of requirements representation [1], [7] used within requirements documents, other requirements storage media (e.g., database-supported repositories), and even models[2]. Therefore, we advocate that, to further benefit from the significant effort in developing requirements specifications in natural language [4], [5], suitable languages and related toolsets are mandatory to better and properly support requirements authoring and validation activities [6].

The human effort required to produce requirements specifications is substantial. Also, the manual nature of RE tasks, and the large amount of information to be considered, cause these activities to be time-consuming and error-prone. Thus, it would be helpful if some of these manually performed tasks 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 general 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, such automation would be helpful for generating an initial draft of the domain model and behavior models (e.g., UML class diagrams and UML use case diagrams, respectively) and for providing them as input to software development processes that follow the MDE paradigm.

A. RSLingo in a Nutshell

The full processing of *ad-hoc* natural language text is still too complex to be automatically performed by computers in a general-purpose context [12]. However, in spite of its complexity, there are some simplifications that can be applied to Natural Language Processing (NLP) techniques [12].

Traditionally, the goal of NLP is to produce deep parsing trees annotated with grammatical features [12]. For producing such tree structures, a full parsing approach is required in order to take into consideration all linguistic aspects of the text being parsed. However, a simpler approach can be used when the goal is only to extract relevant information from text. Considering the purpose to which requirements specifications are created, we advocate that simplified NLP techniques can be used to support the requirements specification activity by avoiding hard-to-notice documentation defects, such as inconsistencies that are easily overlooked by humans. Also, the information extracted from textual requirements representations in natural language can be used to improve the overall quality of requirements specifications by (1) enriching requirements representations with implicit information, (2) applying requirements specification best practices, and (3) reducing their ambiguity and inconsistency by crossing the extracted information with general lexical resources, such as WordNet[3] and VerbNet[4].

Despite being less complex than the traditional full parsing approach, these simplified NLP techniques still enable

[1]As opposed to any variation of structured or controlled natural language.

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

[3]http://wordnet.princeton.edu/

[4]http://verbs.colorado.edu/verb-index/

us to obtain the aforementioned domain knowledge required for gaining a deeper insight on the system to be built. Such simplifications mimic the human process of reading and understanding text and strongly rely on the syntactic–semantic alignment imprinted in linguistic patterns [12]. This correlation allows us to employ "lightweight" parsing techniques to extract – in an feasible manner – both explicit and implicit domain knowledge, encoded within textual requirements representations [12].

Therefore, we named this proposal *RSLingo* approach, in which 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*. Although RE is not in itself "a domain" of interest, RSL-IL is designed to address RE-specific concerns to better support RE-related tasks, and so it can be considered as a Domain-Specific Language (DSL) for RE according to the perspective of business analysts and requirements engineers, i.e., the "domain experts" for RE, according to processes and techniques. On the other hand, "Lingo" expresses that its design has roots in natural language, which are encoded in linguistic patterns used during information extraction from the requirements specification [10]. 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 the definition of the mappings 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 concepts, languages, and using RSLingo's toolset[5] during the execution of a specific software project. The process-level also encompasses tasks for properly configuring RSLingo's toolset and reusing assets from previous projects. However, these tasks are out of the scope of this paper (for further detail on this matter, please refer to [13]).

1) RSLingo's Process-Level: The main asset to be produced, within the context of information extraction from textual requirements representations, is the `RSL-PL => RSL-IL Mapping`. As illustrated in Figure 1, the RSLingo approach considers a role, which we named *Requirements Architect*, that is not typically found in RE. The person performing this role must have a thorough knowledge and vast experience in RE, to be able to tailor the RSLingo approach to a family of similar projects. Also, the Requirements Architect must be able to identify common

[5]The current RSLingo's toolset consists in a Wiki-based application that supports both requirements organization and requirements parsing, while providing the social and collaboration-related benefits of Web 2.0 tools.

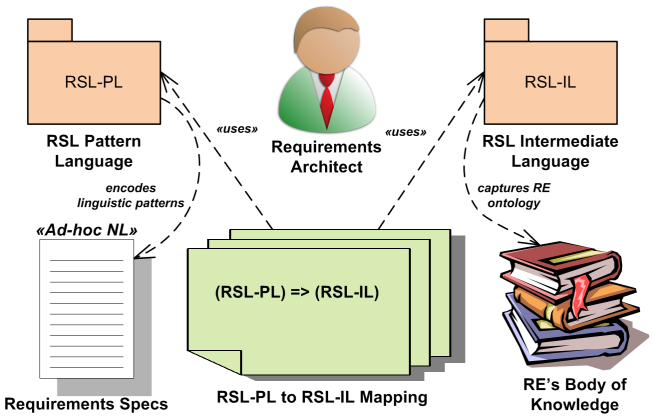

Figure 1. Definition phase of RSLingo at process-level.

linguistic patterns that frequently occur in natural language requirements specifications and know how to translate them to semantically equivalent RSL-IL constructs. However, after this tacit knowledge is encoded into the mapping between RSL-PL linguistic patterns and RSL-IL formal structures, it can be reused multiple times in similar projects.

2) RSLingo's Project-Level: The main concepts of the RSLingo approach at this level are illustrated in Figure 2. As it can be observed, the approach is not disruptive with regard to traditional RE approaches [6]: despite being particularly focused on the writing process of requirements specifications in natural language, it still follows the common RE process [2], relying on other RE activities such as requirements elicitation and negotiation.

During the requirements specification activity, we consider two main roles: the *Requirements Engineer* and the *Business Stakeholder*. RE is social and collaborative by nature [14], thus we value the direct contribution of business stakeholders as well. The Business Stakeholder role can be played by any entity that is (1) external to the development team (e.g., domain expert, client, end-user) and (2) acquainted with the problem-domain. Exhibiting these characteristics, such an entity is considered to be a valuable requirements source. Also, business stakeholders are crucial during the requirements validation activity [3]. Therefore, we consider that the RSLingo approach should encourage 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] in an iterative manner, while using suitable techniques and enforcing RE best practices. However, those playing this role can still directly specify requirements, when necessary.

According to the information flow represented in Figure 2, both the Requirements Engineer and the Business Stakeholder roles contribute with textual inputs to the

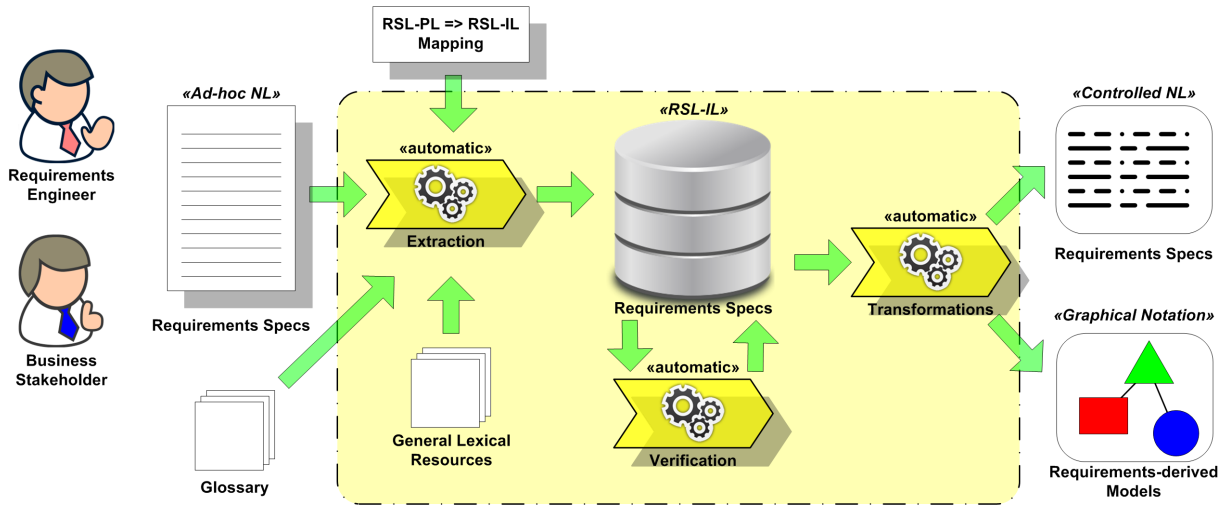

Figure 2. Overview of the RSLingo approach: usage at project-level.

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

In addition to these three manually made artifacts, RSLingo's toolset still requires, during the automatic extraction process, some additional `General Lexical Resources` (e.g., WordNet and VerbNet) for supporting disambiguation tasks and providing additional information on the meaning of terms and their lexical relationships. These `General Lexical Resources` are needed because RSLingo deals with *ad-hoc* natural language. Thus, a lot of world knowledge is required to "fully understand" requirements. Humans instinctively deal with this issue, because they are familiar with natural language and possess this sort of background knowledge, which is acquired through experience. However, computers lack such world knowledge to deal with the meaning of words and derive multiple interpretations from sequences of words that form phrases and sentences. Nevertheless, the extra information provided by these `General Lexical Resources` is overridden by the terms defined in the previously mentioned `Glossary` artifact. The semantics of these project-specific terms defined within the `Glossary` take precedence over those available in the `General Lexical Resources` used to provide a default meaning to terms that are not defined in the `Glossary`. This approach allows some level of customization during ambiguity resolution.

B. How RSLingo's Toolset Works

The first step toward the automation of requirements' text analysis is treating requirements textual representations according to a "white-box" approach to extract and analyze their meaning. For addressing this matter, while not forcing stakeholders to adopt a new notation, we propose the usage of *chunk parsing* [12], which exploits the *syntax–semantic alignment* of natural language text.

Belonging to a broader class of *shallow parsing* techniques [15], chunk parsing contrasts with common NLP techniques (often referred to as *full parsing* techniques), in which a multi-level and completely annotated parse tree is produced, considering most of all linguistic constraints to verify whether it is valid. Contrarily, the aim of chunk parsing is not to enforce grammatical correction, but to exploit the aforementioned alignment between the structure of sentences and their semantics. The structure of each phrase (i.e., linguistic pattern) is based on the *relative position* and *part-of-speech* of each word that it contains. Thus, one can take advantage of this alignment in order to extract the "meaning" of requirements statements or, at least, extract useful information to better support RE tasks. 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-related information, instead of dwelling on the complexity involved in the unification of grammatical features and syntactical correction [12].

In addition to the sentence-level *syntax–semantic alignment*, one must take into consideration the word-level

lexical–semantic alignment. As previously mentioned, we circumvent the lack of world knowledge and common sense, which humans intrinsically possess, by resorting to English linguistic resources such as WordNet and VerbNet lexical databases, and also the Brown and TreeBank annotated corpora of the Natural Language Toolkit (NLTK) [12].

The linguistic patterns defined in RSL-PL refer to individual requirements statements. Thus, it is noteworthy to mention that, before trying to identify a match of these linguistic patterns and textual requirements representations in natural language, RSLingo's toolset must first identify requirements statement boundaries. The identification of these boundaries can be easily achieved by splitting text upon the detection of punctuation marks that often end a sentence in English (common enumeration marks, such as bullets and lists, must also be considered). However, the mechanisms for identifying these well-defined scopes of interpretation that are required for properly analyzing requirements (e.g., indentation, punctuation, or even Wiki markup language) are beyond the scope of this paper.

After the previously explained *syntax–semantic alignment* (at sentence-level) is encoded as linguistic patterns defined in RSL-PL, RSLingo's toolset (based on chunk parsing techniques) identifies the best-fit match for each textual requirements representation of the project at hand. Since the language constructs of RSL-PL also exploit the *lexical–semantic alignment* (at word-level) through parameters of word selectors, the parser also uses the previously mentioned general lexical resources that are publicly available.

When a best-fit match is established between a linguistic pattern and a textual requirements representation, the match's captures provide the relevant information, encoded within the syntactic–semantic alignment of the recognized linguistic patterns. After yielding a best-fit match, a translation from the match's capture to the semantically equivalent RSL-IL formal structures takes place, as described in Information Extraction literature [10]. This translation relies on a previously established mapping between RSL-PL linguistic patterns and RSL-IL formal structures.

The algorithm to identify the best-fit match between the linguistic pattern and the textual requirement representation is based on the Levenshtein Distance algorithm [16]. This algorithm is commonly used for computing the edit distance between two strings in order to determine their similarity. Within the RSLingo approach, we adapted this efficient dynamic programming algorithm to determine the similarity between linguistic patterns and the textual requirement representation. Instead of taking into account characters within the string, we consider words within a phrase. The operations to determine the edit distance between two linguistic patterns were extended to consider additional word attributes related with morphological, lexical, and semantic information.

In short, and as introduced in Figure 2, the automatic domain knowledge extraction process takes three project-specific inputs (`Requirements Specs`, `Glossary`, and `RSL-PL => RSL-IL Mapping`), and identifies the best-fit matches between textual requirements representations and the recognized linguistic patterns encoded in RSL-PL, considering the lexical information provided by the project-level `Glossary` and the `General Lexical Resources` used. The knowledge extracted from linguistic patterns is in turn used to populate a repository of RSL-IL expressions that, all together, represent a formal subset of the original requirements specification written in natural language by those playing the *Business Stakeholder* and *Requirements Engineer* roles.

III. Case Study

To better explain the concepts introduced in the previous section, which form the underpinnings of the RSLingo approach, we provide an illustrative example focused on the requirements specification of a hypothetical call center software system. Through the practical application of RSLingo's concepts, this section overviews the syntax and semantics of both RSL-PL and RSL-IL constructs, and the mapping between them. Also, the explanation provided regarding the information extraction process unveils some of the technical details of RSLingo's toolset and gives a glimpse of how it works in practice[6]. Finally, based on a snippet of the ad-hoc natural language requirements specification of the call center software system, the expected outcome is provided (i.e., the formal RSL-IL specification).

Figure 3 presents the interplay between the three most important artifacts within the RSLingo approach: (1) the `RSL-PL => RSL-IL Mapping` (in the middle of the left column) that is defined by the Requirements Architect at RSLingo's Process-Level; (2) the requirements specification in ad-hoc natural language (in the top right corner), that should be mostly written by *Business Stakeholders* (and also by *Requirements Engineers*, when their supervision and expertise are required) at RSLingo's Project-Level; and (3) the `Glossary`, which represents the role played by both the general-purpose glossary and the custom-tailored, project-specific glossary in RSL-IL. Additionally, Figure 3 illustrates some of the language constructs (in EBNF notation) of both RSL-PL and RSL-IL (in the top and bottom left corners, respectively). As the flows indicated by the larger arrows suggest, both RSL-PL and RSL-IL constructs are used in the `RSL-PL => RSL-IL Mapping`, which simply establishes a translation (through well-defined productions) between linguistic patterns defined in RSL-PL expressions and the equivalent RSL-IL formal structures. As the other larger arrow indicates, this mapping is later used at RSLingo's Project-Level to automatically derive RSL-IL specifications with RSLingo's toolset, as represented in the bottom right corner of the figure.

[6]At the time of the writing of this paper, RSLingo's toolset is still being developed and tested. Thus, it is not yet available for external evaluation.

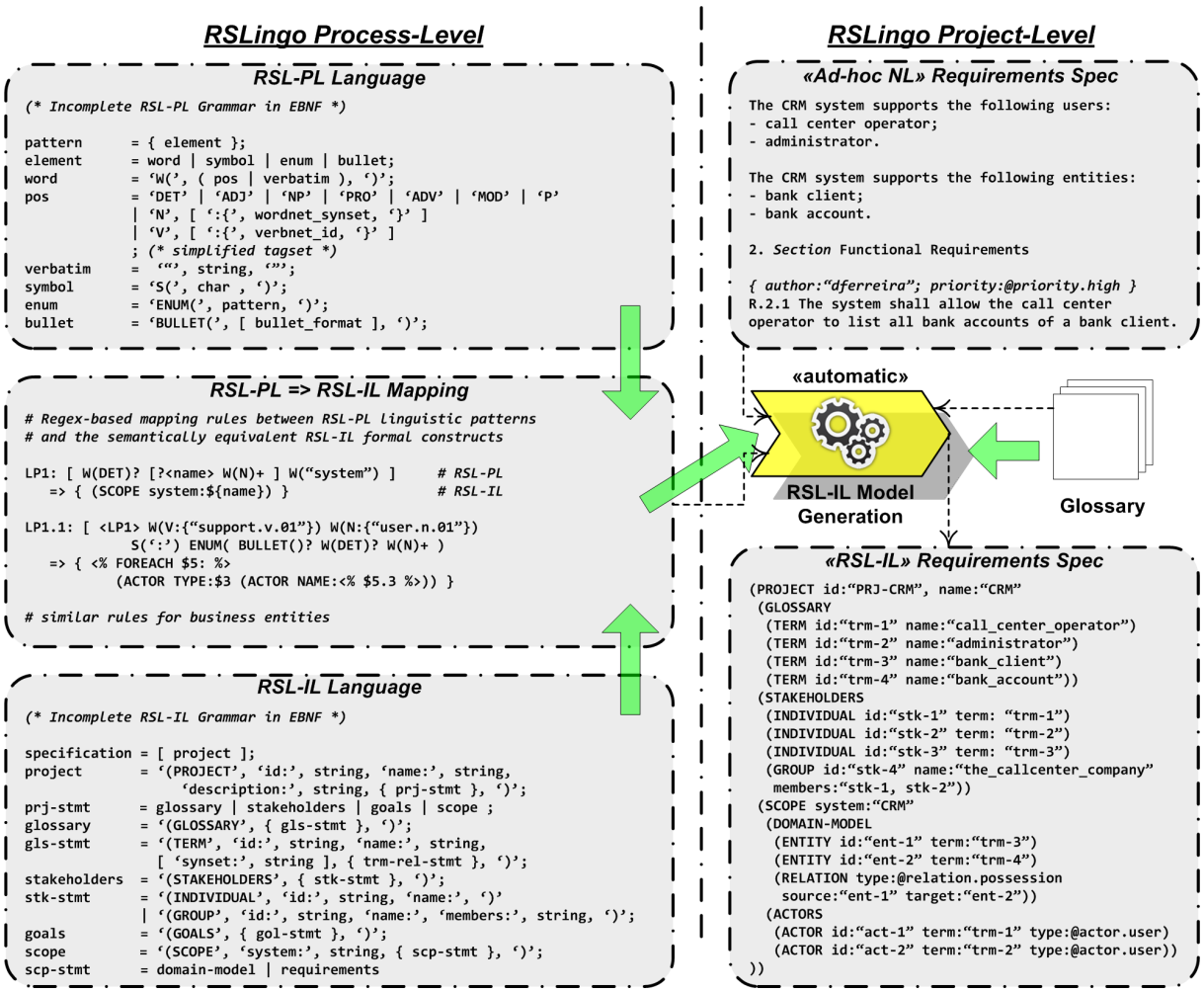

Figure 3. Example of how the RSLingo approach works.

Since the purpose of Figure 3 is to focus on the inter-dependencies between RSLingo's artifacts, it abstracts over concerns that are important according to other viewpoints, such as the document structure of the ad-hoc natural language requirements specification. As was already mentioned, RSLingo's toolset has been developed as a specific Wiki-based tool [13] that supports extensions to the WikiCreole[7] markup language for dealing with *WikiPageItem*s, which provide well-defined regions for stating requirements and their attributes. However, RSLingo's toolset also supports the parsing of plain text requirements specifications.

To cope with plain text specifications, this toolset requires some guidelines to be followed, namely identifying which text must be parsed, and within which context those requirements must be analyzed. To deal with this matter, requirement authors are expected to abide by a few, easy-to-follow conventions, such as those illustrated in Figure 3 in the top-right corner: (1) using hierarchically numbered headings whose text begins with a configurable keyword (e.g., *Section*, *Subsection*); (2) decorating each requirement with a list of key-value pairs within curly brackets, such as UML tagged values; (3) using requirements identifiers, which must be hierarchically numbered; and (4) separating each requirement from the remaining ones through a well-defined paragraph (i.e., by means of an empty line right after and before the requirement text).

To properly address a set of informally stated requirements, such as `R.2.1` introduced in Figure 3, the RE analysis activity would start by interpreting each sentence in order to understand the big picture and derive proper abstractions, namely entities, properties, and the system functionality in terms of actions that act upon the properties of those entities. To illustrate how this can be achieved with the RSLingo approach, we provide in Listing 1 a snippet with a linguistic pattern in RSL-PL and the mapping of its elements into RSL-IL. The RSL-PL pattern (line 1) appears right before the RSL-IL block within curly brackets.

[7]http://wikicreole.org/

Listing 1. Snippet of the mapping from RSL-PL to RSL-IL.

```
1  W(DET) [?<system> W(N)+ ] W("system") W(MOD)
     W(V{related:"allow"}) W(DET) [?<actor> W(N)+ ]
     W("to") W(V) [?<entity>  W(DET) W(N)+ [?<owner>
      W("of") W(DT) W(N)+ ]? ]
2  => {
3    (SCOPE system:${system}
4     (REQUIREMENTS
5      (FUNCTIONAL mandatory:!map($4)
6       text:"The system shall allow the call center
            operator to list all bank accounts of a
           bank client"
7       (SEMANTICS
8        (SUBJECT name:${actor})
9        (ACTIVITY name:$9)
10       (ENTITY name:${entity}.2
11        (RELATION type:@relation.possession
            source:${owner}.3 target:${entity}.2))
12 )))))}
```

Based on this sort of mappings, RSLingo's toolset works as follows. Before starting the natural language chunk parsing process based on the alignment between the linguistic pattern in RSL-PL and the requirements representation in natural language text, it attempts to load any preexistent RSL-IL specification related with the project at hand. This initial step is required for the toolset to be able to load any predefined information (both manually edited and automatically generated RSL-IL statements) related with the natural language requirements to be parsed, such as project-specific glossary entries. Upon each match, following the principles of Information Extraction [10], an equivalent formal structure in RSL-IL is produced, where placeholders are filled with the captures of the match. As modern regular expression flavors [17], RSL-PL supports both numbered capturing groups (line 5) and named capturing groups (line 3). The latter allows one to assign a descriptive name to the capturing group, which makes RSL-PL linguistic patterns more readable and easier to maintain, while also easing the process of mapping the captured values into attributes of RSL-IL constructs. After being populated with the corresponding information from the matched textual requirements representation, these structures are stored in a repository that acts as a "knowledge base" for formal requirements specifications in RSL-IL.

As illustrated, the sort of formal representation achieved with RSL-IL is crucial to enable a seamless integration of high-level design models derived from requirements specifications into the upstream stage of the MDE pipeline. For example, with this information it would be possible to create both structural and behavioral models of the system to be built, namely *class diagrams* (i.e., the domain model) and *use case diagrams* (enriched with an *actor hierarchy diagram*), respectively. Considering this case study, it would be possible to derive such diagrams by applying transformations to the RSL-IL representation of the call center software system's requirements, either through a declarative transformation language or through a algorithmic transversal of the abstract syntax tree of the RSL-IL specification.

Furthermore, the effort required to identify, analyze, and properly document the explicit and implicit information of this simple case study, exemplifies the endeavor of applying these labour-intensive, daunting, and error-prone tasks to a larger scale scenario. Thus, the human effort required to produce the desired formal specification in RSL-IL justifies the need for proper tool support, namely the automatic generation of these specifications through a robust semantic parser, such as the one provided by RSLingo's toolset.

IV. Discussion

Despite being an well-established discipline, whose importance is recognized by both academia and industry, we consider that RE is still not a mature engineering discipline. The body of knowledge provided by literature and textbooks is extensive, providing a wide range of approaches and best practices for eliciting, analyzing, documenting, and managing requirements [6]. However, the RE process still encompasses a plethora of *multidisciplinary techniques* and requires a *significant human effort* to discover what system is to be built. Although industry-strength requirements management tools exist [18], most of the requirements development techniques still consist in producing handmade artifacts whose interpretation strongly relies on natural language text, such as documents, tables, and even diagrams [19]. Therefore, the quality of the major outcome of the RE process (i.e., the reviewed and approved requirements specification) is still strongly influenced by the skills and experience of those performing the RE analysis and documentation activities. A way to deal with this requirements quality problem is to craft a set of templates based on well-formed requirements that abide to the field's best practices and incorporate lessons learned from previous successful projects.

The idea of developing reusable requirement patterns as natural language templates is not new, ranging from abstract patterns (as described in [3]) to more detailed requirement patterns enriched with comprehensive metadata (such as in [20]). Approaches that follow this idea seek to improve the quality of requirements specifications (both in terms of content and writing style) by endowing requirements engineers (or business analysts) with a comprehensive catalog of well-formed and reusable requirement patterns.

One of these approaches is the PABRE [21] method, which focuses on the reuse of non-functional requirements through the creation and evolution of a repository of requirement patterns. Besides the knowledge synthesized within the requirements catalog, the desired practical output of the PABRE method is a "requirements book" to be used in procurement processes. The PABRE method consists in accumulating experience by sampling requirements with similar purpose, and derive from them a refined, well-formed requirement template that has placeholders for variability points. Additionally, each template is enriched with detailed metadata, which is structured according to a frame-based

organization. PABRE also provides tool support, namely: (1) PABRE-Man for assisting the management and evolution of the requirement patterns catalog; and (2) PABRE-Proj for supporting the instantiation of individual requirement patterns at project-level to apply them to a specific context, with the purpose of producing the "requirements book".

Comparing RSLingo with PABRE, it is noteworthy that both distinguish between *process* and *project* stages and both consider an untypical role within the RE process that must be performed by an experienced requirements engineer ("Requirement Patterns Expert" and Requirements Architect, respectively). However, although they share the common purpose of evolving a reusable library of requirement patterns, the way these patterns are used by their respective tools is the opposite: following the PABRE method, the synthesized requirement patterns are used to manually instantiate concrete requirements by using these patterns as form-based templates to be filled at project-time; whereas RSLingo uses similar linguistic patterns (i.e., requirement patterns, but which can be more abstract than those used within PABRE because they abstract over words by considering linguistic concerns) to automatically extract information from requirements written in natural language into RSL-IL, which provides RSLingo's formal underpinnings.

Nevertheless, both [20] and [21] can be used as an additional source of requirement patterns for rapidly evolving the set of recognized linguistic patterns within the RSLingo approach. For instance, these patterns can be adapted by the Requirements Architect at Process-Level for creating additional linguistic patterns in RSL-PL. Also, this sort of template-based approach to improve requirements specification quality can be used to automatically generate well-formed requirements when combined with the RSLingo approach, by selecting the proper requirement pattern based on specific keywords or through a similarity metric, and filling its parameters with requirements in RSL-IL.

Still regarding traditional RE approaches to improve requirements quality, the RSLingo approach clearly contrasts with the common practice of mainly focusing on the organization of requirements (e.g., document structure and requirements relations) and their metadata (e.g., requirements attributes) in databases or diagrams – a sort of "black-box" approach, by analogy with software testing –, around which academia and industry seem to have a greater concern. We regard documentation best practices as being of the utmost importance [5], yet we advocate in addition that 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 sentences in which they are textually represented.

For addressing this problem, several approaches have been proposed, ranging from lightweight heuristic-based solutions to more formal approaches that try to perform complex linguistic analysis on requirements textual representations. A good example of a lightweight extraction technique that is used to populate a semi-formal model is provided by Kamalrudin's work [22]. This approach addresses behavioral requirements, focusing on scenario-based descriptions. The goal is to derive Essential Use Cases, which are shorter, simpler, and technology-free versions of conventional Use Cases. These sequences of abstract steps help requirements engineers to better understand the user's intrinsic needs. Instead of using conventional NLP-based approaches, Kamalrudin et al. follow a domain specific approach. To this end, they developed a library of essential interactions expressed as textual phrases, phrase variants, and limited regular expressions. Through string manipulations and some regular expression matching, they are able to identify textual segments that activate the mapping into more abstract representations. For further validation of the original natural language representation of the behavioral requirement, the sequence of these abstract representations is then compared to another library of well-formed Essential Use Cases.

On the other hand, following an approach that adopts a more complex linguistic analysis, one alternative would be using a Controlled Natural Language (CNL), such as Attempto Controlled English (ACE) [23]. ACE is a mature general-purpose CNL with a solid toolset and a wide literature body of knowledge. Furthermore, ACE possesses some interesting (and useful) characteristics, namely every ACE sentence is unambiguous, even if people may perceive the sentence as ambiguous in full English [24].

Apparently this would partly solve the problem, because 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 [23]. However, since CNLs are typically designed for knowledge engineering, they do not properly address behavioral aspects, such as those required within the scope of RE for prescribing how the system must interact with external entities (i.e., users and other systems).

However, CNLs have a drawback: they are *easy to read*, yet *hard to write*. This happens because writing with CNLs resembles programming with a high-level programming language that, although more familiar, still requires that the statements of the produced specification abide to strict grammar rules. ACE was engineered for resembling *ad-hoc* natural language, hence it is easy for untrained users to become frustrated while using it. They might receive feedback on grammar errors despite what they wrote appearing to be correct English. Therefore, CNLs require significant effort and specialized tools [24], [25] (such as predictive editors) for creating language-compliant specifications (or adapting preexistent ones to become compliant).

While recognizing the importance of controlled natural languages within other fields (e.g., knowledge engineering), we consider that the obstacles it poses to authors' intent to contribute within the context of RE's collaborative work environments are serious impediments to its adoption for requirements documentation by untrained users. Since these limitations prevent domain experts from directly authoring their own requirements statements, we propose a more flexible linguistic approach based on Information Extraction [26].

Not forgetting that the idea underneath the RSLingo approach is to avoid the need to force business stakeholders to learn new formal or semi-formal notations, we followed a different approach. RSLingo's research stream is similar to CIRCE's [27], which provides a "lightweight formal method" supported by model-checking techniques to verify requirements specifications written in natural language. CIRCE's toolset provides user feedback through a multi-perspective approach. To this end, CIRCE uses fuzzy matching parsing – which is based on MAS (Model, Action, Substitution) rules and a scoring mechanism – to extract knowledge from requirements specifications, which are later used to generate different views on the captured information to support requirements analysis.

However, the RSLingo approach goes further in the conceptual distinction between the two different concerns to be considered: (1) the definition of linguistic patterns, and (2) the formal specification of requirements knowledge. This clear separation of concerns is addressed by two 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. Consisting of a well-defined and fixed set of constructs, RSL-IL provides the means to defined a sound and stable "knowledge base" of requirements specifications, supporting reuse of specifications independently of RSL-PL extensions. These two languages (and the mapping between them) provide a requirements specification approach endowed with extensibility and reusability mechanisms.

In addition, RSLingo's toolset not only employs part-of--speech tagging techniques [12] (like CIRCE does), but it also makes use of general lexical resources that are crucial to support disambiguation tasks. Finally, the parsing algorithms used by RSLingo's toolset are based on dynamic programming principles, instead of strongly relying on recursion and being model-parametrized as those used by CIRCE's toolset. These characteristics make CIRCE's toolset computationally heavier and harder to debug (e.g., identifying parsing problems due to inconsistencies in their transformation-oriented rules). Also, tweaking CIRCE's parametrized model, which is used by the scoring mechanism, might generate different outcomes given the same requirements specification.

All these improvements introduced by RSLingo, with regard to other requirements specification approaches, provide a stepping stone toward the ultimate goal of automatically providing alternative requirements representations that support all stakeholders in having a deeper insight on the problem at hand, and thus better understanding the implications of the set of stated natural language statements that represent their *real* requirements [2].

V. Conclusion

In this paper we introduce RSLingo, a linguistic approach that allows one to automatically extract problem-domain knowledge from textual requirements specifications written in natural language and formally represent this captured knowledge. RSLingo is a multi-language approach since its operationalization relies on two RE-specific languages, RSL-PL and RSL-IL, and the mapping between them.

RSL-PL is used to define linguistic patterns for capturing aspects such as specification structure, interpretation context, and sentence constituents and their meaning. On the other hand, RSL-IL is used to formally encode RE-specific concerns such as problem-domain entities, namely their relations, stakeholder goals, features, actors, and functionality scenarios. The decoupling between linguistic aspects and RE-specific concerns provides a flexible approach toward formal requirements specification and the validation of these artifacts by the business stakeholders themselves. This can be achieved through transformations from formally specified requirements in RSL-IL into other (eventually filtered) presentation formats, such as viewpoint-oriented natural language paraphrases, tables, and diagrams.

Unlike other requirements specification approaches, that treat requirements representations as "opaque entities", the RSLingo approach allow us to gain a deeper understanding of the system to build through simple NLP techniques. While recognizing the importance of graphical modeling languages, namely to provide an overview or viewpoints on a specific concern, RSLingo encourages a "back-to--roots" path, encouraging business stakeholders to actively state their requirements in natural language. Thus, with RSLingo, we advocate a greater focus on textual requirements specifications. Nevertheless, despite focusing on a more conventional approach of documenting requirements by writing them down in natural language, RSLingo also lays the foundation for automatically generating artifacts for the subsequent phases of software development.

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 its user's perspective. The next step in the research's roadmap is to exploit the potential of RSLingo approach in producing alternative representations from requirements specifications formally encoded in RSL-IL. For instance, we foresee automatically

generating drafts of design models that, after being enriched with solution-oriented information, can be used as input for the subsequent phases of a software development process following the MDE paradigm. Being a top-down and transformation-based paradigm, MDE considers models as the most important artifacts, instead of source code. Thus, requirements-derived models, which can be generated through the transformation of RSL-IL requirements to UML, can be a very useful input for MDE-based processes.

ACKNOWLEDGMENT

This work was supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under the PEst-OE/EEI/LA0021/2011 project.

REFERENCES

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

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

[3] 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.

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

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

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

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

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

[9] H. Kaindl and D. Svetinovic, "On confusion between requirements and their representations," *Requirements Engineering*, vol. 15, pp. 307–311, 2010, 10.1007/s00766-009-0095-7.

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

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

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

[13] D. Ferreira and A. Silva, "Wiki-Based Tool for Requirements Engineering According to the ProjectIT Approach," in *4th Int. Conf. on Software Engineering Advances 2009 (ICSEA '09)*. IEEE Computer Society, September 2009, pp. 359–364.

[14] B. Decker, E. Ras, J. Rech, P. Jaubert, and M. Rieth, "Wiki-Based Stakeholder Participation in Requirements Engineering," *IEEE Software*, vol. 24, no. 2, pp. 28–35, 2007.

[15] S. Abney and S. P. Abney, "Parsing By Chunks," in *Principle-Based Parsing*. Kluwer Academic Pub., 1991, pp. 257–278.

[16] G. Navarro, "A guided tour to approximate string matching," *ACM Computing Surveys*, vol. 33, no. 1, pp. 31–88, March 2001.

[17] J. Goyvaerts and S. Levithan, *Regular Expressions Cookbook*. O'Reilly Media, May 2009.

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

[19] 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.

[20] S. Withall, *Software Requirement Patterns (Best Practices)*. Microsoft Press, June 2007.

[21] C. Palomares, C. Quer, and X. Franch, "PABRE-Man: Management of a Requirement Patterns Catalogue," *International Conference on Requirements Engineering*, pp. 341–342, 2011.

[22] M. Kamalrudin, J. Hosking, and J. Grundy, "Improving Requirements Quality using Essential Use Case Interaction Patterns," *International Conference on Software Engineering*, pp. 531–540, 2011.

[23] N. Fuchs and R. Schwitter, "Attempto Controlled English (ACE)," in *Logic-Based Program Synthesis and Transformation*. University of Leuven, March 1996.

[24] N. E. Fuchs, K. Kaljurand, and T. Kuhn, "Attempto Controlled English for Knowledge Representation," in *Reasoning Web, Fourth International Summer School 2008*, ser. Lecture Notes in Computer Science, no. 5224. Springer, 2008, pp. 104–124.

[25] T. Kuhn, "Controlled English for Knowledge Representation," Ph.D. dissertation, Faculty of Economics, Business Administration and Information Technology of the University of Zurich, 2010.

[26] R. Gaizauskas and Y. Wilks, "Information Extraction: Beyond Document Retrieval," *Journal of Documentation*, vol. 54, no. 1, pp. 70–105, 1998.

[27] 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.