2012 Eighth International Conference on the Quality of Information and Communications Technology

Formally Specifying Requirements with RSL-IL

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

***Abstract*—To mediate and properly support the interplay between the domains of business stakeholders and the development team, requirements must be documented and maintained in a rigorous manner. Furthermore, to effectively communicate the viewpoints of different stakeholders, it is of the utmost importance to provide complementary views that support a better understanding of the intended software system's requirements. However, the quality of requirements specifications and related artifacts strongly depends on the expertise of whoever performs these human-intensive and error-prone activities.**

This paper introduces RSL-IL, a domain-specific language that can be used to formally specify the requirements of software systems. The formal semantics of RSL-IL constructs enable further computations on its requirements representations, such as the automatic verification and generation of complementary views that support stakeholders during requirements validation.

***Index Terms*—Requirements Engineering, Requirements Specification, Formal Languages.**

I. Introduction

Requirements Engineering (RE) is based on an iterative and incremental process that allows one to strengthen the bridge between the real-world needs of business stakeholders and those providing a solution to them, i.e., the development team. In order to *build the right system*, the development team must learn about the *problem domain*, including concepts and relationships between them, by consulting the most important sources of requirements, the business stakeholders.

Considering the cascading effects (in terms of rework and scheduling) of potential misinterpretations regarding business needs, stakeholders expectations, and operating environment, the importance of proper documentation and communication of system and software requirements becomes clear [1].

Natural language is the most common and preferred form of requirements representation [1], [2]. However, the usage of natural language for documenting requirements usually gives rise to requirements specifications with well-known quality problems [3], such as: ambiguity, incompleteness, inconsistency, and incorrectness. Thus, we advocate that, in order to further benefit from the significant effort in developing requirements specifications [3], [4], suitable languages and supporting tools are mandatory to better and properly support requirements authoring and validation activities [5].

This paper introduces RSL-IL, which can be regarded as a Domain-Specific Language (DSL) considering that it was designed with the sole purpose of addressing RE-related concerns [6], [7]. Although it can be directly used to specify requirements, RSL-IL belongs to a broader requirements specification approach named RSLingo [8], which adopts Information Extraction [9] techniques to populate RSL-IL specifications with relevant information extracted from requirements written in natural language. By supporting the formal specification of requirements and their metadata, the goal is to enable further computations on their representations, such as automatically checking for requirements quality criteria, and generating derived requirements specifications and models.

The structure of this paper is as follows. Section II introduces RSL-IL within the context of RSLingo. Furthermore, this section details the main design concerns of RSL-IL and the respective language constructs that were created to address them. For better explaining these aspects, Section III introduces a simple example. In turn, Section IV compares RSL-IL with related work. Finally, Section V concludes this paper and lays down some ideas for future work.

II. Overview of RSL-IL

Considering the expected usage of a typical requirements specification, it would be helpful if some of the labor-intensive (and error-prone) tasks related with analysis and assessment of quality criteria 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;
- ***Verification:*** for enabling consistency checking on the extracted domain knowledge, via inference and ambiguity resolution based on glossaries and other lexical resources;
- ***Transformations:*** for automatically generating alternative requirements representations such as diagrams, tables, reports, or even viewpoint-based paraphrases of requirements statements in a controlled natural language.

A. The Role of RSL-IL within the RSLingo Approach

RSL-IL belongs to a broader approach called *RSLingo*, whose name stems from the paronomasia on "RSL" and "Lingo". On 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*. This designation was adopted to emphasize that RSL-IL works as a back-end language to convey detailed information about requirements in a formal manner. However, although it aims to the formal specification of requirements models derived from their textual textual representations, RSL-IL is human-readable and its notation can be directly used by requirements engineers to specify requirements. Figure 1 depicts the two requirements specification scenarios supported by the RSLingo approach.

978-0-7695-4777-0/12 $26.00 © 2012 IEEE
DOI 10.1109/QUATIC.2012.30

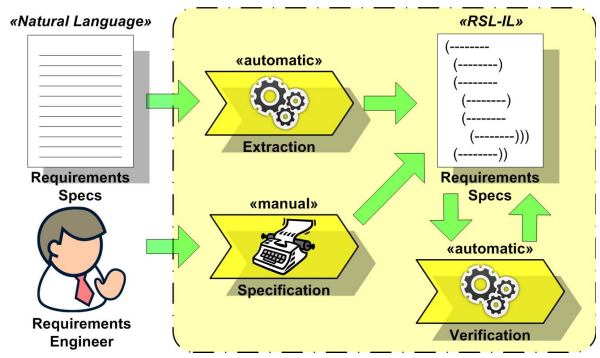

Fig. 1. Authoring requirements specifications in RSL-IL.

On the other hand, "Lingo" expresses that its design is rooted in natural language, which is encoded in linguistic patterns used during the information extraction [9] from requirements specifications. The language designed for encoding these RE-specific linguistic patterns is *RSL-PL*, in which "PL" stands for *Pattern Language*. These linguistic patterns are consumed by shallow parsing techniques that, when combined with lexical resources such as WordNet (http://wordnet.princeton.edu/) and VerbNet (http://verbs.colorado.edu/verb-index/), enable the extraction of relevant information from the textual representation of the requirements that are being parsed. The extracted information is formally specified in RSL-IL notation through predefined transformations from RSL-PL into RSL-IL. Upon a match of a requirement's textual representation with one of the RSL-PL linguistic patterns, a transformation becomes active. This transformation takes into consideration the semantic roles of each word within the linguistic pattern and drives the mapping between RSL-PL and RSL-IL (the details on RSL-PL's notation and how the information extraction approach of RSLingo works are beyond the scope of this paper).

B. High-Level Architecture of RSL-IL

To properly address the majority of RE's concerns, namely "requirements elements" [7] – such as *stakeholders*, *goals*, *scenarios*, *qualities* and *constraints*, *rationale* and *assumptions*, *definitions*, *measurements*, and *priorities* –, we grouped RSL-IL constructs into different packages as illustrated in Figure 2. Each of these packages covers a specific perspective according to which requirements should be specified. Their purpose and the RSL-IL constructs they entail are as follows.

Terminology. To prevent misunderstandings one needs to address the root of the problem: interpreting the meaning of `Terms` employed within requirements representations. Thus, we follow the best practice of providing a project-wide `Glossary`. A `Term` is composed of one or more *words* that represent a concept, and provides a normalized representation of its textual form and also lexical information (e.g., its *part-of-speech*) to allow cross-checking the suitability of its use in the remaining packages of RSL-IL (e.g., verbs should not be used to describe domain entities). Furthermore, `Terms` with similar meaning are grouped into a `GlossaryEntry` to address polysemy. One of these synonyms is marked as the *representative* `Term`, hence it should be systematically used. Also, a `GlossaryEntry` has a *description* of the intended meaning and, optionally, *examples* of sentences employing the representative `Term`. Since constructing a `Glossary` is a daunting task, WordNet can be reused to provide the default meaning of terms. In short, the constructs of the *Terminology* package play a central role regarding the remaining RSL-IL packages. It lays the foundation to attain the goal of enabling semantic-level verifications of requirements specifications because it provides crucial information regarding the smallest units of meaning within requirements statements, i.e., words.

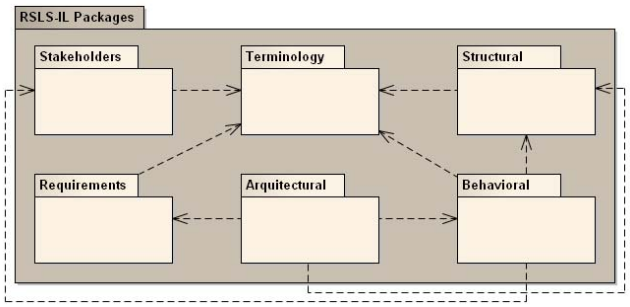

Fig. 2. Overview of the internal organization of RSL-IL's constructs.

Stakeholders. The aspects addressed by this package establish the relation between requirements and real-world goals of business stakeholders via traceability links. Without them, we cannot ensure that we are delivering a suitable solution because we lack the answer to "*why is this system needed?*". The RSL-IL constructs entailed in this package are: `Individual`; `Organization`; and `Goal`. These constructs capture the necessary and sufficient information for generating common stakeholder analysis artifacts [10]. Also, these constructs support conflict resolution by determining who has the authority to decide which option to follow.

Structural. One needs to analyze the set of domain concepts for which the system-of-interest must possess properly abstracted representations. Concepts must be structured and organized according to the principles of conceptual modeling, such as "classes and instances", "parts and wholes", and "specializations and generalizations". Thus, this package lays down the RSL-IL's foundations for domain analysis. The constructs entailed in this package are: `Concept`, which can be *abstract* or *concrete*; `Property`, which are binary relations (with a typified domain and range) that are equivalent to "externalized" attributes in an object-oriented approach; `DataType`, which can be built-in or user-defined; and `Invariant`, which can be globally applied to the whole software system, or applied just within a concept's scope.

Behavioral. To properly describe the system-of-interest, one must answer the question "*what is it supposed to do?*", i.e., describe its behavior that support business processes. To address this matter, RSL-IL entails constructs

based on use case descriptions [10], namely: `Actor`; `UseCase`; `UseCaseRelation`; (triggering) `Event`; (pre- and pos-) `Condition`; `Scenario`; and (signal or data entry) `Interaction`. This package relies on information captured by the constructs of other packages. For instance, an `Actor` (a human or an external system) must be defined within the `Terminology` package and the set of human `Actors` must be a subset of (or derived from) `Individual` stakeholders.

Requirements. All RSL-IL constructs that are related with individual `Requirements` are included in this package, namely their: type (e.g., `FunctionalRequirement` and `QualityRequirement`), `Attributes`, `Relations`, and `Rationale` history. Additionally, it includes constructs for conveying the `Semantics` of `Terms` that are employed within the textual representations of requirements, namely to identify their `T(hematic)-Roles` (based on thematic relations [11]), which describe the role that each argument plays regarding the predicate of the requirement statement.

Architectural. Requirements are assigned to systems, subsystems, and components. This association is often used as a guideline during requirements documentation to properly structure them as a coherent collection of requirements [3]. This package provides a general `Scope` construct, which can be nested according to the Composite design pattern, to address the representation of these architectural relations.

III. Example

This section illustrates how RSL-IL can be applied to specify the requirements of an hypothetical call center software.

The excerpt depicted in Spec. 1 overviews a RSL-IL specification. It entails `Project` as the top-most construct of RSL-IL that, in turn, has several constructs which are aligned with the requirements perspectives addressed by RSL-IL.

Spec. 1. Overview of a requirements specification in RSL-IL.

```
(PROJECT id:"proj-2012-04-ccs" name:"Call Center Software"
  (GLOSSARY (...))
  (STAKEHOLDERS (...))
  (SCOPE id:"scp_ccs" type:SCOPE.SYSTEM
    name:"Call Center Software"
    (PROBLEM-DOMAIN (...))
    (REQUIREMENTS (...))))
```

Regarding the *Terminology* package, Spec. 2 presents an example of a `GlossaryEntry`. Each `Term` that it contains is automatically checked for consistency regarding its *part-of-speech*. Additionally, this assignment explicitly constrains the `T-Roles` that each `Term` can play within the `Semantics` of a given `Requirement` textual representation. Also, the occurrence of a given `Term` that is not marked as *representative* should be systematically replaced to reduce ambiguity.

Spec. 2. RSL-IL constructs within the *Terminology* package.

```
(GLOSSARY
  ((GLOSSARY-ENTRY id:"ge_1" pos:POS.NOUN
    (TERM word:"call center operator" representative:TRUE )
    (TERM word:"operator"))))
```

Spec. 3, presents an excerpt of a stakeholders-related specification. First, all parties involved are defined via the `Organization` construct and its attributes, namely *type* (a built-in enumeration) and *name*. Afterwards, individual stakeholders are defined. Each `Individual` is characterized by its *type* (a built-in enumeration), *name*, and *organization*.

Spec. 3. RSL-IL constructs within the *Stakeholders* package.

```
(STAKEHOLDERS
  (ORGANIZATION id:"org_1" type:ORG_TYPE.ACQUIRER
    name:"TeleContactYou")
  (ORGANIZATION id:"org_2" type:ORG_TYPE.SUPPLIER
    name:"Bytes'R'Us")
  (INDIVIDUAL id:"stk_1" type:STK_TYPE.CUSTOMER
    name:"Bob" organization:"org_1")
  (INDIVIDUAL id:"stk_2" type:STK_TYPE.DEVELOPER
    name:"Mark" organization:"org_2"))
```

Finally, the last two snippets provide a glimpse of the remaining content of the `Scope` construct. Since the *Architectural* perspective is responsible for providing a more concrete and integrated view of requirements (in which they are assigned to subsystems and software components), the respective package relies on RSL-IL constructs defined in other packages. Therefore, a problem domain description must be provided, specifying all `Concepts` involved and their characteristics (`Property`), as depicted in Spec. 4.

Spec. 4. Integrating RSL-IL constructs within the *Architectural* package.

```
(PROBLEM-DOMAIN
  (CONCEPT id:"c_1" name:"contact"
    (PROPERTY id:"p_1" name:"contact name"
      range:DATA_TYPE.STRING)
    (PROPERTY id:"p_2" name:"number"
      range:DATA_TYPE.INTEGER))
  (CONCEPT id:"c_2" name:"call"
    (PROPERTY id:"p_3" name:"sender" range:"contact")
    (PROPERTY id:"p_4" name:"receiver" range:"contact")
    (PROPERTY id:"p_5" name:"duration"
      range:DATA_TYPE.INTEGER)))
```

Spec. 5 shows a `FunctionalRequirement` and how it can be further analyzed in terms of the `Semantics` of its textual representation based on the `T-Roles` of `Terms`.

Spec. 5. Integrating RSL-IL constructs within the *Architectural* package.

```
(REQUIREMENTS
  (FUNCTIONAL-REQUIREMENT id:"FR_1"
    text:"The system shall enable the call center
      supervisor to configure the monthly report."
    (SEMANTICS
      (T-ROLE type:"provider" word:"system")
      (T-ROLE type:"doer" word:"call center suppervisor")
      (T-ROLE type:"action" word:"configure")
      (T-ROLE type:"entity" word:"monthly report"))))
```

IV. Discussion

Contrarily to the current trend of strongly relying on graphical modeling languages (such as UML and SysML) for specifying requirements information, we advocate that more attention should be given to the semantics of conventional textual requirements representations written in natural language.

Considering the *status quo* in terms of requirements specification practices, requirements are treated as "opaque entities" since, regardless of the notation used, a requirement's *true meaning* can only be provided by human interpretation. For example, despite significantly improving human understanding, graphical notations are not self-sufficient: they still require

textual descriptions that paraphrase their meaning in order to be fully understood. In practice, the usefulness of RSLs, whether they are textual or graphical, strongly depends on proper tool support to enforce the language's constraints and report errors or inconsistencies regarding its semantics.

Therefore, we advocate that a RSL must be formal by design, such as the Requirements Modeling Language (RML) [12]. For assigning well-defined semantics (i.e., a unique interpretation) to requirements representations, the RSL itself must be formally described in terms of all its constructs. To address this matter, one can adopt some mathematical or logical formalism as the RSL's underpinnings, or provide a metamodel enriched with natural language descriptions of the semantics and restrictions of the language constructs and their relations. RML adopts the former approach, while the ReDSeeDS RSL (http://www.redseeds.eu) follows the latter. Regarding RSL-IL, it also follows the latter metamodel-based approach, since it has the specific purpose of providing an independent and formal language that only addresses RE-related issues. However, as a formal *intermediate language*, it can be translated into other logical or mathematical formalisms to perform further computations, if required.

The advantages of the RSLingo approach strongly depend on the formal semantics of RSL-IL's constructs. To be flexible and not preclude adoption by business stakeholders, RSLingo must follow a more conventional approach based on textual requirements specifications. Nevertheless, RSL-IL provides the means to build a repository of requirements that act as a knowledge base. This repository receives "projections" of the original natural language representations specified in RSL-IL. Also, RSL-IL lays the foundations for RSLingo to automatically generate semantically equivalent representations in other notations (e.g., diagrams and tables) from these RSL-IL requirements specifications. These automatically generated artifacts support both the validation by business stakeholders and further development activities.

Another unique advantage of using RSL-IL within the RSLingo approach is the stability and reusability it provides, independently of how the linguistic patterns defined in RSL-PL might evolve to cope with additional requirements representations. Since RSL-IL is composed of a small and fixed set of language constructs, it supports the specification and reuse of requirements in a stable manner. Although it was not designed to be an interchange format such as ReqIF [13], RSL-IL serves a similar purpose but at a deeper semantic level, enabling further computations upon the specified requirements.

Finally, a small remark regarding the concrete syntax of RSL-IL. The adopted notation was designed to be easy to use by untrained users. It follows a LISP-like syntax in which the scope of RSL-IL constructs is defined by parenthesis. Each RSL-IL construct has certain attributes that are represented as key-value pairs. An alternative would be to use a standard markup language, such as XML. Although it would bring some parsing benefits, it would certainly be more verbose, thus negatively influencing the notation's usability. Nevertheless, after the language's abstract syntax is well-defined, it is possible for one to convert one concrete syntax (i.e., notation) into another, according to the purpose at hand.

V. Conclusion

This paper introduces RSL-IL, a formal language designed specifically to support the precise specification of requirements as clearly as possible. This language provides the underpinnings for the RSLingo approach, whose goal is to enable business stakeholders and requirements engineers to collaboratively author requirements specifications. RSL-IL should not be regarded as yet another formal RSL whose design is an end in itself, but instead as a cornerstone that provides the means to: (1) support domain analysis, for a deeper insight on the system-of-interest; (2) enable the automatic verification of some requirement quality criteria; and (3) serve as the basis for the automatic transformation of the conveyed RE knowledge into other complementary representations. The outcome of such automatic transformations can support requirements validation activities via well-formed paraphrases of the requirements specifications written by stakeholders, which can be even enriched with complementary information, such as diagrams and tables.

As future work we plan to exploit the potential of RSLingo in producing alternative representations from requirements formally encoded in RSL-IL. For instance, we consider the possibility of automatically generating high-level 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.

References

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

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

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

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

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

[6] 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.

[7] I. Alexander and L. Beus-Dukic, *Discovering Requirements: How to Specify Products and Services*. Wiley, April 2009.

[8] D. Ferreira and A. Silva, "An Overview of the RSLingo Approach," in *The 24th International Conference on Software Engineering and Knowledge Engineering 2012 (SEKE 2012)*, July 2012, To appear.

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

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

[11] A. Carnie, *Syntax: A Generative Introduction (Introducing Linguistics)*, 2nd ed. Wiley-Blackwell, September 2006, ISBN-13: 978-1405133845.

[12] S. Greenspan, J. Mylopoulos, and A. Borgida, "On formal requirements modeling languages: RML revisited," in *Proceedings of the 16th international conference on Software engineering*, ser. ICSE '94. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, pp. 135–147.

[13] Object Management Group, "OMG Requirements Interchange Format (ReqIF), Version 1.0.1," April 2011, Retrieved Monday 16th April, 2012 from http://www.omg.org/spec/ReqIF/1.0.1/.