Linguistic Patterns and Linguistic Styles for Requirements Specification (I): An Application Case with the Rigorous RSL/Business-Level Language

Alberto Rodrigues da Silva, INESC-ID, Instituto Superior Técnico, Universidade de Lisboa.

System requirements specification describes technical concerns of a system and is used throughout the project life-cycle. Requirements specification helps sharing the system vision among its stakeholders, as well facilitating the communication, project management and system development processes. For an effective communication, everyone communicates by means of a common language, and natural language provides the foundations for such language. Although natural language is the most common and preferred form of requirements representation, it also exhibits intrinsic characteristics that often present themselves as the root cause of many requirements quality problems, such as incorrectness, inconsistency, incompleteness and ambiguousness.

This paper presents the RSL (short name for "Requirements Specification Language") which is a language to improve the production of requirements specifications in a more systematic, rigorous and consistent way. RSL includes *constructs* logically arranged into *views* according to the specific requirement engineering concerns they address. These constructs are defined as *linguistic patterns* and are represented textually by multiple *linguistic styles*. Due to space constraints, this paper focuses only on its *business level constructs and views*, namely on glossary terms, stakeholders, business goals, processes, events and flows. RSL can be used and applied by different types of users such as requirement engineers, business analysts, or domain experts. They can produce system requirements specifications with RSL at different level of detail, considering different writing styles and different types of requirements (e.g., business goals, system goals, functional requirements, quality requirements, constraints, user stories, and use cases). In addition, they can use other types of constructs (e.g., terms, stakeholders, actors, data entities) that, in spite of not being requirements, are important to complement and enrich the specification of such requirements. Based on a simple running example, we also show how RSL users (i.e., requirements engineers and business analysts) can produce requirements specifications in a more systematic and rigorous way.

Key Words: RSL, Requirements Specification, Requirements Patterns, Linguistic Patterns

1. INTRODUCTION

Requirements Engineering (RE) consists of several processes and practices to provide a shared vision and understanding of the system to be developed between business and technical stakeholders. The adverse consequences of disregarding the importance of the early activities covered by RE are well-known (Standish Group, 2009; Eveleens and Verhoef, 2010). ***System requirements specification, software requirements specification or just requirements specification (SRS)*** *is a document that describes multiple technical concerns of a software system*. An SRS is used throughout different stages of the project life-cycle to help sharing the system vision among the main stakeholders, as well as to facilitate the communication and the overall project management and system development processes (Sommerville and Sawyer, 1997; Pohl, 2010).

EuroPLoP '17, July 12–16, 2017, Irsee, Germany

ACM ISBN 978-1-4503-4848-5/17/07…$15.00
https://doi.org/10.1145/3147704.3147728

A good SRS provides several benefits, namely (Cockburn, 2001; IEEE, 1998; Kovitz, 1998; Robertson and Robertson, 2006; Withall, 2007): establish the basis for agreement between customer and supplier; provide a basis for estimating budget and schedule; provide a baseline for validation and verification; and serve as a basis for future maintenance activities. An SRS is sometimes also integrated in legal documents surrounding a project's Request for Proposals or Contracts. An SRS tends to follow a previously defined template that prescribes a given document structure (with chapters and sections) with supplementary practical guidelines. By definition, such SRS templates should be adapted and customized to the needs of the organization involved. In any case, these templates prescribe the use of multiple constructs and models that can be considered "modular artifacts", in the sense of their definition and reuse. However, there are several dependencies among and even within these modular artifacts, which are important to minimize or prevent (Verelst et al., 2013).

On the other hand, to achieve an effective communication, everyone should be able to communicate by means of a common language, and ***natural language (NL)*** provides the foundations for such language: NL is flexible, universal, and humans are proficient at using it to communicate with each other. NL has minimal adoption resistance as a requirements documentation technique. However, although NL is the most common and preferred form of requirements representation, it also exhibits some intrinsic characteristics that often present themselves as the root cause of many requirements quality problems, such as incorrectness, inconsistency, incompleteness and ambiguousness (EEE, 1998; Kovitz, 1998; Robertson and Robertson, 2006; Pohl, 2010).

To avoid or mitigate such problems, we find in the literature practical recommendations for writing requirements better, including ***guidelines*** such as (Videira and Silva, 2005; Videira et al, 2006; Pohl, 2010; Fernandes and Machado, 2016): the language should be simple, clear and precise; should follow a standardised format to give coherence and uniformity to all sentences; the sentences should be short, simple, and written in an affirmative and active voice style; the vocabulary (used terms) should be limited. These recommendations are better supported by the guidance of controlled languages. A ***controlled natural language (CNL)*** defines *a restricted use of a NL grammar (syntax) and a set of terms (including the semantics of the terms) to be used in the restricted grammar* (Robertson and Robertson, 2006; Pohl, 2010). The adoption of CNLs has the following advantages: (i) the CNL sentences are easy to understand, since they are similar to sentences in NL; (ii) are less ambiguous than expressions in NL, since they have a simplified grammar and a predefined vocabulary with a precise semantics; and (iii) are semantically verifiable and can be computational manipulated, since they have a formal grammar and the predefined terms.

Linguistic patterns are *grammatical rules that allow their users to speak properly in a common language*. From the linguist's point of view, a grammar is not only a collection of rules, but rather a set of blueprints that guide speakers in producing comprehensible and predictable sentences. Languages and their variations are rule-governed structures, and are therefore grammatical. In other words, all languages consist of patterns that make sense of the features that include the arbitrary symbols, sounds, and words that make up that language (DeCapua, 2008).

Despite the diversity of names found in the literature – e.g., "requirements template" (Durán et al, 1999), "syntactic requirements pattern" (Pohl, 2010), or "standardized format" (Fernandes and Machado, 2016) –, we adopt in this paper the terms "linguistic pattern" and "linguistic style" to mean the definition and representation of such grammatical rules. Examples of RE's linguistic patterns found in the literature are discussed for example in (Ferreira and Silva, 2013):

Patterns for **Individual Requirements**:

<when> the system shall <process> <object>.
When the glass break detector detects the damaging of a window, the system shall inform the head office of the security service.

The system shall <action> <to_whom> <what>.
The system shall ask the operator for the passenger name.

<actor> can <action> <what> <how>.
A user can view landscape files in AutoCAD format.

Pattern for **Use Case steps**:

<subject> <verb> <direct_object> <prepositional_phrase>.
The system deducts the amount from the account balance.

Pattern for **User Stories**:

As a <type of user>, I want <some goal>, so that <some reason>.
As a user, I want to backup my selected documents, so that I can save and protect my information.

RSLingo is a long-term research initiative in the RE area that recognizes that natural language, although being the most common and preferred form of representation used within requirements documents, is prone to produce such ambiguous and inconsistent documents that are hard to automatically validate or transform. Originally RSLingo proposed an approach to use simplified Natural Language Processing (NLP) techniques as well as human-driven techniques for capturing relevant information from ad-hoc natural language requirements specifications and then applying lightweight parsing techniques to extract domain knowledge encoded within them (Ferreira and Silva, 2012). This was achieved through the use of two original languages: the RSL-PL (Pattern Language) (Ferreira and Silva, 2013), designed for encoding RE-specific linguistic patterns, and RSL-IL (Intermediate Language), a domain specific language designed to address RE concerns (Ferreira and Silva, 2013a). Through the use of these two languages and the mapping between them, the initial knowledge written in natural language can be extracted, parsed and converted to a more structured format, reducing its original ambiguity and creating a more rigorous SRS document (Silva, 2015a). This approach foresees the whole process of knowledge extraction and conversion to a more rigorous representation as a way to help business stakeholders gain a better understanding of the set of stated natural language statements that represent their real requirements. However, to a technical user already familiar with the concepts of RE, the NLP phase can be omitted, as there is already a much better understanding of the RE concerns and of its documentation process.

In a previous work we discussed the result of our experiences in looking for combinatorial effects at different levels in the RE-process (Verelst et al, 2013). Then, we compared the modular structures of three SRS templates in terms of the extent to which they prevent combinatorial effects from happening; and we proposed a set of practical recommendations to define SRS templates that would mitigate the referred problem (Silva et al, 2014).

In 2015 we also proposed a pattern language to improve the quality of use cases specifications (Silva et al, 2015). These patterns were concrete guidelines that have emerged from our research and industrial experience, particularly in the designing of languages and MDD approaches (Stahl & Volter, 2005; Silva, 2015) and in applying them in concrete applications. However, this "Pattern Language for Use Cases Specification" was proposed for requirement architects interested in designing such types of specifications and not so much for people interested in authoring/writing requirements specifications directly, such as ***business analysts*** *or* ***requirements engineers****, as it is the* ***target audience*** *of this paper.*

Recently, we designed a broader and more consistent language, called "RSLingo's RSL" (or just "RSL" for brevity), based on the design of the former languages, i.e., ProjectIT-RSL (Videira et al, 2006), RSL-IL (Ferreira and Silva, 2013a), RSL-PL (Ferreira and Silva, 2013), but also others such as Pohl (Pohl, 2010), XIS* (Silva et al, 2007; Ribeiro and Silva, 2014/a), and SilabREQ (Savic, D. et al, 2015).

RSL is a control natural language to help the production of SRSs in a more systematic, rigorous and consistent way. RSL is a language that includes a rich set of *constructs* logically arranged into *views* according to multiple RE-specific concerns that exist at business and system abstraction levels. These constructs are defined as linguistic patterns and represented textually by mandatory and optional fragments (text snippets). For example, the people and organizations that can influence or can be affected by the system are represented by the construct Stakeholder. Likewise, the objectives of business stakeholders, regarding the value that the system will bring, are represented by the construct BusinessGoal, etc.

Conceptually RSL is a process- and tool-independent language meaning it can be used and be adapted by multiple users and organizations with different processes/methodologies and supported by multiple types of software tools. However, in practice RSL has been implemented with the Xtext framework (https://eclipse.org/Xtext/) and so, its specifications are rigorous and can be validated and transformed automatically (into other representations and formats). A lightweight tool support is also provided based on an RSL's Excel template publicly available at github (https://github.com/RSLingo).

Due to space constraints ***this paper introduces RSL language and its general architecture but focuses the discussion only on its business level constructs and views***. The explanation of the language is supported by a fictitious business information system, called "BillingSystem" (described in Section 4), which description is deliberately incomplete, vague and inconsistent.

2. DEFINITION OF LINGUISTIC PATTERNS AND LINGUISTIC STYLES

In the context of this paper a ***linguistic pattern*** *is a set of rules that defines both the element(s) and the vocabulary that shall be used in SRS's sentences.* An element rule defines a set of element attributes (e.g., <id>, <name> or <type>) which are defined by the following properties: name, type and multiplicity. On the other hand, a vocabulary rule defines a set of literal terms (e.g., 'Person', 'Business', 'Business_Customer') used to categorize some element attributes and to restricted the use of a limited number of terms.

Example: linguistic pattern Stakeholder

For example the linguistic pattern Stakeholder (see more details in Section 5.2) is defined by the following set of rules (the Stakeholder element rule and the StakeholderType vocabulary rule):

```
Stakeholder::
      <id:ID> <name:String> <type:StakeholderType> <isA:Stakeholder>? <description:String>?

enum StakeholderType::
      Organization | OrganizationalUnit | Team | Person | System | Other;
```

The Stakeholder element rule defines its element attributes (e.g., <id>, <name>, <type>, <isA>, <description>) and for each attribute the respective name (e.g., name, type, isA), type (e.g., ID, String, StakeholderType, Stakeholder) and multiplicity (e.g., '?') properties. The multiplicity of an attribute is not represented by default (means "1", a mandatory attribute), or can be represented by the following characters '?', '+', and '*' meaning, respectively, 0..1 (optional), 1..* (one or many), and 0..* (zero or many). The type of an attribute can be: a predefined type (e.g., ID, String, Boolean); an element type (e.g., the Stakeholder of the isA attribute); or a vocabulary type (e.g., the StakeholderType of the type attribute).

StakeholderType vocabulary rule is prefixed with the "enum" tag and defines the values of its parts, namely the literals 'Organization', 'OrganizationalUnit', 'Team', 'Person', 'System' and 'Other'.

As shown with this simple example a linguistic pattern defines two major aspects regarding the use of a controlled language: (i) a vocabulary, with a limited number of terms; and (ii) and a controlled set of element's attributes with respective name, type and multiplicity (e.g., mandatory or optional).

However, a linguistic pattern can be represented by multiple manners depending on the interests and background of its users. In the context of this paper a ***linguistic style*** *is a concrete representation of a linguistic pattern*. That means that a linguistic pattern can be represented by multiple linguistic styles, either defined by some controlled natural language (CNL) or more rigorously by RSL.

A linguistic style is an ordered set of two types of text fragments: literal text fragments (e.g., 'stakeholder', '[', ']', ', is described by '); and other template text fragments that are represented by the pattern "<element_name.attribute_name>", i.e. the element name followed by its attribute name, delimited between '<' and '>' (e.g., <stakeholder.type>). For brevity reasons it is possible to omit the reference to the "element_name.". The multiplicity constraints are represented by the same characters referred above (i.e., '?', '+' and '*').

On the other hand, the rules that define RSL constructs are compliant with the following general linguistic style, as a set of formal rules defined by a BNF-like format supported by the Xtext framework (Bettini, 2016):

```
'element' name=ID ':' type=ElementType '{'
      ('name' nameAlias=STRING)
      ('isA' super=[Element])?
      ('description' description=STRING)?
      [etc.]
'}'
```

Example: linguistic styles for the Stakeholder pattern

Considering the example above, it is possible to define multiple linguistic styles for the Stakeholder pattern (see more details in Section 5.2), such as:

LS1 (verbose version):

```
Stakeholder <stakeholder.name> is a <stakeholder.type> [, is a <stakeholder.isA>]? [, described
as <stakeholder.description>]?.
```

Or the equivalent compact format, LS1 (compact version):

```
Stakeholder <name> is a <type> [, is a <isA>]? [, described as <description>]?.
```

LS2:

```
(<id>) <name> is a <type> stakeholder [, is a <isA>]? [, described by <description>]?.
```

LS3 defined more rigorously with a BNF rule as defined by the RSL language:

```
'stakeholder' name=ID ':' type=StakeholderType '['
      ('name' nameAlias=STRING)
      ('category' category= StakeholderCategory)
      ('isA' super=[Stakeholder])? &
      ('partOf' partOf=[Stakeholder])? &
      ('description' description=STRING)?
']'
```

3. RSL OVERVIEW

RSL includes *constructs* logically arranged into *views* according to the specific Requirement Engineering (RE) concerns they address. These constructs are defined by *linguistic patterns* and represented textually according concrete *linguistic styles*. RSL is a process- and tool-independent language, i.e. it can be used and adapted by different users and organizations with different processes/methodologies and supported by multiple types of software tools.

This section overviews RSL by introducing its bi-dimensional multi-view architecture, based on abstraction levels and concerns. It also explains how to define SRS templates and how to structure a SRS in RSL as a set of inter-related RSL package files.

3.1 RSL Viewpoints: Abstraction Levels versus RE Specific Concerns

RSL provides several constructs that are logically arranged into views according to two viewpoints: the abstraction level (Levels) and the specific RE concerns (Concerns) they address. As summarized in Table 1, these views are organized according to two abstraction levels: business and system levels; and to five concerns: context, active structure, behavior, passive structure and requirements.

Table 1. Classification of RSL views: abstraction levels versus RE specific concerns.

Concerns / **Levels**	**Package**	**Context**	**Active Structure** (Subjects)	**Behavior** (Verbs, Actions)	**Passive Structure** (Objects)	**Requirements**
Business	package-business	Business System(s) relations	Stakeholder	BusinessProcess (BusinessEvent, BusinessFlows)	GlossaryTerm	BusinessGoal
System	package-system	System	Actor	StateMachine (State, Transition, Action)	DataEntity DataEntityView	SystemGoal QR Constraint FR UseCase UserStory

At the business level, RSL supports the specification of the following business-related concerns: (1) the people and organizations that can influence or will be affected by the system; (2) business processes, events, and flows that might help to describe the business behavior; (3) the common terms used in that business domain; and (4) the general business goals of stakeholders regarding the value that the business as well the system will bring. Considering these concerns, RSL business level comprise respectively the following views: Stakeholders (active structure concern), BusinessProcesses (behavior concern), Glossary (passive structure concern), and BusinessGoals (requirements concern). In addition, the references to the systems used by the business, as well as their relationships can also be defined at this level (context concern).

At the system level, RSL supports the specification of multiple RE specific concerns, namely by the adoption of the following: (1) constructs that allow to describe the actors that interact with the system; (2) constructs that allow to describe the behavior of some system's data entities, namely based on statemachines; (3) constructs that allow to describe the structure of the system, namely based on data entities and data entity views; and (4) constructs that allow to specify the requirements of the system according different styles. Considering these concerns, the system level respectively comprises the following views: Actors (active structure concern); StateMachines (behavior concern); DataEntities and DataEntityViews (passive structure concern); and multiple types of Requirements such as SystemGoals, QualityRequirements (QRs), Constraints, FunctionalRequirements (FRs), UseCases, and UserStories (requirements concern). In addition, all these elements and views should be defined in the context of a defined System (context concern).

3.2 RSL and SRS Templates

RSL also supports our previous discussion on SRS templates, in which we proposed a template that helps to minimize and prevent combinatorial effects at some extent, based on the following practical recommendations (Silva et al, 2014): (R1) Separate business level from system level chapters; (R2) split the system into subsystems, specify each subsystem into a self-contained chapter; (R3) NFRs (non-functional requirements) shall be specified in an independent chapter (but can also be specified at a subsystem chapter, if they are detailed enough) and can be derived from goals; (R4) decouple actors from stakeholders; (R5) decouple functional requirements from use cases; (R6) define a glossary of terms, splitting application domain terms from project or system specific terms; (R7) specify the integration of concerns; (R8) define domain-specific anticipated changes; and (R9) document the appropriate level of abstraction.

In that discussion we identified multiple dependencies among the constructs and models commonly used in SRSs. Figure 1 suggests some of the most popular perceived dependencies using the RSL constructs and views. Additionally, there are other dependencies that are implicitly illustrated in the figure and other dependencies that are not illustrated at all for the sake of simplicity. For example, there are dependencies from several constructs (e.g., stakeholders, actors, data entities) to the terms defined in the glossary, meaning that the name of these constructs can or should be properly defined in the SRS glossary. Another example of implicit relationships are the intra-dependencies that still might exist between FRs, QRs, Constraints, and UseCases.

Several adaptations can occur on the top of this proposed template without a significant impact, for example: move the glossary of terms and the references sections from the Chapter 1 to the Appendixes in the end of the document; or for simple software systems or for SRS defined at an early stage of a project, typically the requirements engineer can merge the chapters focused on the functional concerns of the different subsystems into just one (i.e., can merge Chapters 3, 4, … N-1 into only one chapter); or still in this situation and for the sake of simplicity, the requirements engineer can avoid describing detailed information of the subsystem's functional description (e.g., avoiding to describe use cases and usage scenarios) because it can just be enough to enumerate the main Goals or FRs for each subsystem.

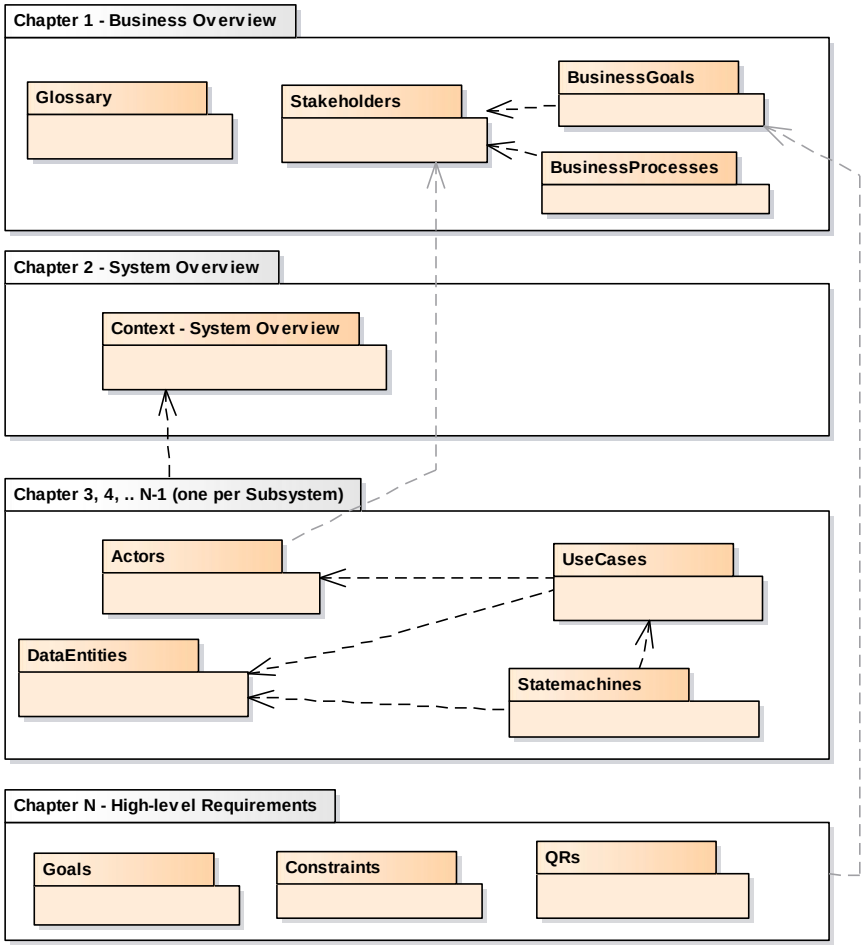

Figure 1: Example of a SRS-template based on the RSL constructs and views.

3.3 SRS Defined as a Set of RSL packages

As illustrated in Figure 2, a SRS specified in RSL (i.e., a SRS Model) is defined as a set of RSL packages. There are two types of packages, defined at business or system levels, respectively: PackageBusiness and PackageSystem. As suggested in Figure 2, any package can use/access constructs defined in other packages by the "import" package relation. In general a concrete SRS specified in RSL can contain just one PackageBusiness file and multiple PackageSystem files, but other combinations are possible depending on the level of detail that is required or the modularity and manageability of such package files.

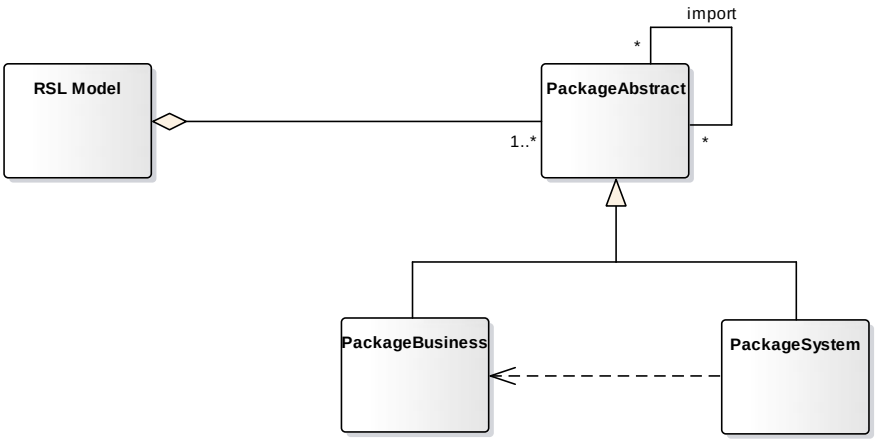

Figure 2: RSL model as a set of business and/or system level packages.

4. RUNNING EXAMPLE: THE BILLINGSYSTEM

To support the explanation and discussion of the RSL language we introduce a fictitious business information system, called *"BillingSystem",* partially described as a variety of informal requirements such as the following text.

Example: Informal requirements of the BillingSystem

BillingSystem is a system that allows users to manage customers, products and invoices. A user of the system is someone that has a user account and is assigned to one or more user roles (or user profiles), such as user, user-operator, user-manager and user-administrator:

User shall be able to login, logout, password recover, and update his own information. A user shall login in the system by specifying his respective username and password. If the user forgets his user name and/or password the system shall provide a feature to recover and change the password by sending via email a link to reset the old password. Each user shall be allowed to change his own information.

User-administrator shall be responsible for managing users, configuring technical features (e.g., user roles, export configuration parameters, general enterprise information). To access the BillingSystem users must be registered in the system. System shall allow user-administrator to register users if they are not registered, or to update particular user information if they are registered. During the user registration process user-administrator needs to specify first and last name of the user, email address, and username. User password shall be automatically generated by the system and sent to his user email before system saves user in the system.

User-operator is responsible for managing customers and invoices. System shall allow user-operator to create/update information related to customers and invoices. For each customer the system shall maintain the following information: name, fiscal id, logo image, address(es), bank information and additional information such as basic person contact information. System shall allow user-operator to define some customers as VIP. If a customer is defined as VIP, for each of her invoice, the system shall allow user-operator to set a predefined discount tax. That amount of discount can change throughout the time and depends on the current policy of the company. For each product the system shall

maintain the following information: name, description, price, VAT category, and VAT value. Product must have only one VAT category and maintain the respective current VAT value.

The creation of invoices is a shared task performed by the user-operator and the user-manager. System shall allow user-operator to create new invoices (with respective invoice details). Before sending an invoice to a customer, the invoice shall be formally approved by the user-manager. Only after such approval, the user-operator shall issue and send that invoice electronically by e-mail and by regular post. In addition, for each invoice, the user-operator needs to keep track if it is paid or not.

The System shall automatically trigger an alert for all the invoices that were sent to customers but not yet paid, after 30 days of their respective issue date. The System shall archive all paid or rejected invoices whose creation date is older than 1 year.

User-manager shall be responsible for monitoring the events that happen in the BillingSystem and for approving invoices before they are issued and sent to their customers. User-manager shall allow monitoring the process of creating, approving and payments invoices. User-manager shall approve or reject invoices. User-manager shall consult several types of reports with aggregated invoice-based information, such as a report of all invoices by customer and year, or a report of all invoices by product and year.

The BillingSystem shall send periodically (e.g., monthly) all issued and paid invoices to the ERP-Accounting external system. This means that the BillingSystem shall send that information based on a web service provided by the ERP-Accounting system.

The BillingSystem shall be developed in a project involving the following companies: MySoftwareHouse as the performing organization, and TheCustomer as the pilot source organization (meaning the organization that shall use this system during the project period). The owner of this system is the MySoftwareHouse organization that intends to market and distribute it in different countries with different languages. Also due to that reason, the system shall be flexible, easy to configure and to use by different types of end-users.

This project shall adopt the Scrum as the management and development agile process; and the system version 1.0 shall be released before the end of the current year.

The system shall be developed on the top of the following technologies: PostgreSQL Server and the ASP.NET Core framework.

After 6 months of the official version 1.0 released the MySoftwareHouse organization intends to sell and implement the BillingSystem for more than 500 customers in its own country and 1000 customers in EU region.

This description is to some extent deliberately incomplete, vague and inconsistent, as it is common in real-world situations. However, this example raises some questions that allow discussing the motivation and use of a language such as RSL.

5. RSL/BUSINESS-LEVEL

As illustrated in Figure 3, a SRS specified in RSL at business-level involves the definition of the following constructs and respective views.

The names of the relations illustrated in Figure 3 suggest their respective semantics. For example, the "isA" reflexive relation between GlossaryTerm means that such terms have hypernym or hyponym (generalization/specialization) relations, or the "partOf" reflexive relation between BusinessGoals means that a business goal can be decomposed or can aggregate other business goals.

To guide the explanation and discussion of the RSL business-level constructs and views we first introduce the Glossary and Stakeholders views, then the BusinessProcesses view, and finally the BusinessGoals view.

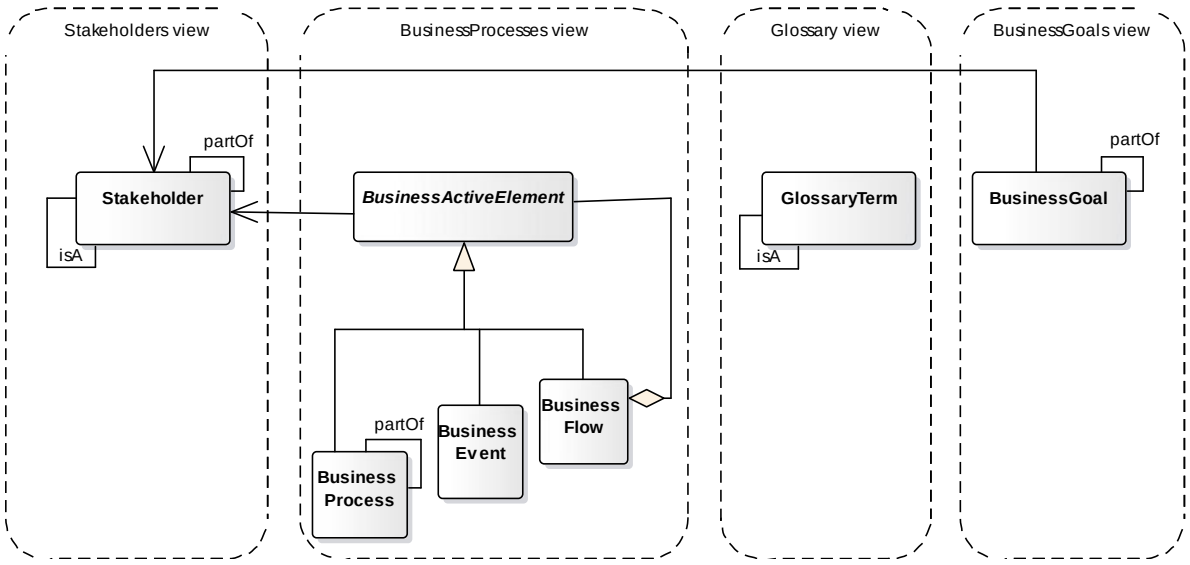

Figure 3: RSL business-level constructs and views.

5.1 Glossary View

Glossary view includes both the business domain and system domain terms used throughout the SRS. The main purpose of this view is to reduce the negative effects of natural language's imprecision through ambiguity resolution techniques. The effort required to identify and properly define the most relevant terms used by business stakeholders brings significant advantages, because these terms refer to the most important real world notions to be considered and they help everyone to use consistently the same terms with the same meaning while defining and building the system.This view is not dependent of any other view; however, other views should use the terms defined in it, such as Stakeholders, Actors, or DataEntities.

A GlossaryTerm shall be identified and specified consistently according its grammatical function, that we designate as TermType, namely as a noun (e.g., user, user-operator, customer, product), verb (e.g., login, logout, password recover, manage), adjective (e.g., adult, children) or adverb (e.g., monthly, automatically). In addition, it shall be possible to specify to which elements the term is applicable to (e.g., stakeholders, actors, data entities), and also specify the respective acronym and synonyms, if relevant. Table 2 summarizes the fragments included in a GlossaryTerm specification. The column

"Multiplicity" defines the multiplicity constraint for each fragment: "1", "0..1" or "0..*" meaning, respectively, that the fragment is mandatory, optional or can have zero or multiple values.

Table 2. Summary of GlossaryTerm fragments.

GlossaryTerm	Multi-plicity	Description	Type/Values
id	1	unique identifier of the element	ID
name	1	name of the element	String
type	1	type of the element (part-of-speech)	TermType: Adjective, Adverb, Noun, Verb
applicableTo	0..*	to which type of element this term can be applied	TermApplication: Stakeholder, System, Actor, DataEntity, Other
acronym	0..1	acronym of the term	String
synonym	0..*	synonym of the term	String
isA	0..1	generalization relation between terms	[GlossaryTerm]
description	0..1	general description of the element	String

Linguistic pattern:

```
GlossaryTerm::
        <id:ID> <name:String> <type:GlossaryTermType>
        <applicableTo:TermApplication>* <acronym:String>? <synonym:String>*
        <isA:GlossaryTerm>? <description:String>?

enum TermType :: Adjective | Adverb | Noun | Verb
enum TermApplication :: Stakeholder | System | Actor | DataEntity | Other
```

(lp1)

Linguistic style in some informal CNL:

```
Term <name> (<id>) is a <type> [, applicable to <applicableTo>]?
[, acronym <acronym>]?     [, synomym <synomym>]? [, is a <isA>]? [, described as
<description>]?.
```

(ls1-cnl)

Linguistic style in RSL:

```
'term' name=ID ':' type=TermType  '['
        ('name' nameAlias=STRING)
        ('applicableTo' ref= RefTermApplication)?
        ('acronym' acronym=STRING)?
        ('synonym' synonym=STRING)?
        ('isA' super=[GlossaryTerm])?
        ('description' description=STRING)?
']';
```

(ls1-rsl)

Example: Glossary for the BillingSystem SRS

Glossary of terms represented according the linguistic style ls1-cnl:

```
Term User (t_user) is a Noun, applicable to Actor/DataEntity/Stakeholder.

Term SysAdmin (t_SysAdmin) is a Noun, applicable to Actor/Stakeholder, synonym
SysAdministrator/SystemAdmin/ user-administrator, is a User (t_ user), described as "SysAdmin is
the user role responsible for the major admin tasks...".

Term Enterprise (t_Enterprise) is a Noun, applicable to Stakeholder, synonym
CustomerOrganization/Organization, described as "an entity that licenses and uses the
BillingSystem".

Term Customer (t_Customer) is a Noun, applicable to Stakeholder/DataEntity, synonym Client,
described as "an entity (organization or person) to whom the invoices are issued to".
```

Glossary of terms represented according the linguistic style ls1-rsl:

```
term t_user : Noun [name "User" applicableTo Actor/DataEntity/Stakeholder
      description "user of the system is someone that..."]

term t_sysAdmin : Noun [name "SysAdmin" applicableTo Actor/Stakeholder
      synonym "SysAdministrator/SystemAdmin" isA t_user
      description "SysAdmin is the user role responsible for the major admin tasks..."]

term t_enterprise : Noun [name "Enterprise"
      applicableTo Stakeholder synonym "CustomerOrganization/Organization"
      description "an entity that license and use the BillingSystem"]

term t_customer : Noun [name "Customer"
      applicableTo Stakeholder/DataEntity  synonym "Client"
      description "an entity (organization or person) to whom the invoices are issued to"]
```

5.2 Stakeholders View

Stakeholders view defines *who is the source* of BusinessGoals and SystemGoals and also *who are the participants* of BusinessProcesses. Without a stakeholders identification you cannot properly understand the business or ensure that the delivered software is a suitable solution. This can happen because either: vital information might be missing, since an important Stakeholder (or a group of them) was not identified early in the project; or one is unable to determine the reason why a requirement was originally stated, or even to clear out its true meaning in subsequent phases.

This view has only one dependency: the most relevant stakeholder‘s names should be present in the Glossary view. On the other hand, the views BusinessGoals, BusinessProcess (at business level), Actors and Goals (at system level) can also depend of this view.

A stakeholder should be classified by its own type (StakeholderType) namely as an Organization, OrganizationalUnit, Team, Person, System, or Other type. In addition and optionally, a stakeholder can be categorized by the role performed in the scope of the business (StakeholderCategory) for instance as a Business_Customer, Business_Sponsor, Business_User, Business_User_Direct, etc. Table 3 summarizes the fragments included in a Stakeholder specification.

Table 3. Summary of Stakeholder fragments.

Stakeholder	Multi-plicity	Description	Type/Values
id	1	unique identifier of the element	ID
name	1	name of the element	String
type	1	type of the element	StakeholderType: Organization, OrganizationalUnit, Team, Person, System, or Other
category	1	the business category performed by the stakeholder	StakeholderCategory: User, User_Direct, Business, Business_Customer, etc.
isA	0..1	generalization/specialization relation between stakeholders	[Stakeholder]
partOf	0..1	aggregation relation between stakeholders	[Stakeholder]
descripti	0..1	general description of the element	String

Linguistic pattern:

```
Stakeholder::
       <id:ID> <name:String> <type:StakeholderType> <category:StakeholderCategory>
       <isA:Stakeholder>? <partOf:Stakeholder>? <description:String>?

enum StakeholderType:: Organization|OrganizationalUnit|Team|Person|System|Other

enum StakeholderCategory:: Business | Business_Customer | Business_Sponsor | [etc.]
```

(lp2)

Linguistic style in some informal CNL:

```
Stakeholder <name> (<id>) is a <type>, category <category> [, is a <isA>]? [, is
part of <partOf>]? [, described as <description>]?.
```

(ls2-cnl-a)

Linguistic style in another informal CNL:

```
<id>: <name> is a <type> involved as a <category> [, is a <isA>]? [, is part of
<partOf>]? [, and is <description>]?.
```

(ls2-cnl-b)

Linguistic style in RSL:

```
'stakeholder' name=ID ':' type=StakeholderType '['
       ('name' nameAlias=STRING)
       ('category' category= StakeholderCategory)
       ('isA' super=[Stakeholder])? &
       ('partOf' partOf=[Stakeholder])? &
       ('description' description=STRING)?
']'
```

(ls2-rsl)

Example: Stakeholders for the BillingSystem SRS

Stakeholders represented according the linguistic style ls2-cnl-a:

```
// Users
Stakeholder User (stk_1) is a Person, category User_Direct.
Stakeholder Operator (stk_2) is a Person, category User_Direct, is a User (stk_1).
Stakeholder Manager (stk_3) is a Person, category User_Direct, is a User (stk_1).
Stakeholder SysAdmin (stk_4) is a Person, category User_Direct, is a User (stk_1).

// Systems
Stakeholder TheBillingSystem (stk_5) is a System, category System.
Stakeholder ERP-Accounting (stk_6) is a System, category System_External.

// Others
Stakeholder Customer (stk_7) is a Organization, category Business, described as "the company
registered in each BillingSystem instance".
Stakeholder TheCustomer (stk_8) is a Organization, category Business_Customer, described as "the
company that shall use and test the BilingSystem".
Stakeholder MySoftwareHouse (stk_9) is a Organization, category Business_Owner...
Stakeholder FinanceInstitute (stk_10) is a Organization, category Business_Government...
Stakeholder Enterprise (stk_11) is a Organization, category Business, described as "the company
that license and use the BillingSystem".
```

Stakeholders represented according the linguistic style ls2-cnl-b:

```
// Users
stk_1: User is a Person involved as a User_Direct.
stk_2: Operator is a Person involved as a User_Direct, is a User (stk_1).
stk_3: Manager is a Person involved as a User_Direct, is a User (stk_1).
stk_4: SysAdmin is a Person involved as a User_Direct, is a User (stk_1).

// Systems
stk_5: TheBillingSystem is a System involved as a System_OfInterest.
stk_6: ERP-Accounting is a System involved as a System_External.

// Others
stk_7: Customer is an Organization involved as a Business, and is the company registered in each
BillingSystem instance.
...
```

Stakeholders represented according the linguistic style ls2-rsl:

```
// Users
stakeholder stk_user: Person [name "User" category User_Direct]
stakeholder stk_operator: Person [name "Operator"  category User_Direct isA stk_user]
stakeholder stk_manager: Person [name "Manager" category User_Direct isA stk_user]
stakeholder stk_sys_admin: Person [name "SysAdmin" category User_Direct isA stk_user]

// Systems
stakeholder stk_system: System [name "TheBillingSystem" category System]
stakeholder stk_erpAcc: System [name "ERP-Accounting" category System_External]

// Others
stakeholder stk_customer: Organization [name "Customer" category Business
      description "Customer registered in each BillingSystem instance"]
stakeholder stk_theCustomer: Organization [name "TheCustomer" category Business_Customer
      description "The company that shall use and test the BilingSystem"]
stakeholder stk_mySoftwareHouse: Organization [name "MySoftwareHouse" category Business_Owner]
stakeholder stk_fi: Organization [name "FinanceInstitute"  category Business_Government]
stakeholder stk_enterprise: Organization [name "Enterprise (MySoftwareHouse's Customer)"
      category Business description "the company that license and use the BillingSystem"]
```

5.3 BusinessProcesses View

BusinessProcesses view shall define in a simple way business processes/tasks, events, and the control flows among them. This view and respective constructs are influenced by the equivalent design of the BPMN (www.bpmn.org) and Archimate (http://www.opengroup.org/) languages, but still in a simpler way. This view can contribute to the general understanding of the business context. The constructs provided by RSL allow identifying and specifying the most relevant information from a business behavior perspective. However, for a detailed specification of such processes a richer notation should be adopted such as BMPN.

As suggested in Figure 3, this view has only one dependency: the participant of/for each business process should be a stakeholder previously defined in the Stakeholders view. On the other hand, some views defined at the system-level (such as UseCases or FRs views) may have dependencies to this view, if relevant.

5.3.1 Business Events

A BusinessEvent is defined as something that happens (internally or externally) and influences the business behavior. BusinessEvents can be classified according its type (BusinessEventType) as: Send, Receive, Timer, Terminate, Cancel, Error, Signal, Conditional or Undefined. Table 4 summarizes the fragments included in a BusinessEvent specification.

Table 4. Summary of BusinessEvent fragments.

Business Event	Multi-plicity	Description	Type/Values
id	1	unique identifier of the element	ID
name	1	name of the element	String
type	1	type of the element	BusinessEventType: Send, Receive, Timer, Terminate, Cancel, Error, Signal, Conditional or Undefined
isInitial	0..1	check if this is a start event	Boolean
isFinal	0..1	check if this is a final event	Boolean
participant	0..1	reference for the participant (stakeholder)	[Stakeholder]
description	0..1	general description of the element	String

Linguistic pattern:

```
BusinessEvent::
        <id:ID> <name:String> <type:BusinessEventType>
        <isInitial:Boolean>? <isFinal:Boolean>?
        <participant:Stakeholder>? <description:String>?

enum BusinessEventType:
        Send | Receive | Timer | Terminate | Cancel | Error | Signal |
        Conditional | Undefined
```

(lp3)

Linguistic style in some informal CNL:

```
<name> is a <type> [if <isInitial> start]? [if <isFinal> end]? event [, occurs in
the scope of <participant>]? [, described as <description>]?.
```

(ls3-cnl)

Linguistic style in RSL:

```
'businessEvent' name=ID ':' type=BusinessEventType '['
        ('name' nameAlias=STRING)
        (isInitial ?= 'isInitial')?
        (isFinal ?= 'isFinal')?
        ('participant' stakeholder=[Stakeholder])?
        ('description' description=STRING)?
']'
```

(ls3-rsl)

5.3.2 Business Processes

A BusinessProcess is defined as *a behavior element that groups behavior based on an ordering of processes or tasks* (simple or elementary processes). A BusinessProcess can be classified according to the BusinessProcessType as Manual, User, Service, Send, Receive, Script, BusinessRule, or Undefined, with the same meaning of the equivalent types of BPMN. A BusinessProcess can be part of a more large process and vice-versa, based on the "partOf" relations. For simplicity reason both business processes and business tasks (usually recognized as elementary processes) are represented by the same construct: BusinessProcess. Table 5 summarizes the fragments included in a BusinessProcess specification.

Table 5. Summary of BusinessProcess fragments.

Business Process	Multi-plicity	Description	Type/Values
id	1	unique identifier of the element	ID
name	1	name of the element	String
type	1	type of the element	BusinessProcessType: Manual, User, Service, Send, Receive, Script, BusinessRule,
participant	0..1	reference for the participant (stakeholder)	[Stakeholder]
participantExternal	0..1	reference for an external participant (stakeholder); only relevant for processes of type Send or Receive	[Stakeholder]
partOf	0..1	aggregation relation between processes	[BusinessProcess]
description	0..1	general description of the element	String

Linguistic pattern:

```
BusinessProcess::
        <id:ID> <name:String> <type:BusinessProcessType>
        <participant:Stakeholder>? <participantExternal:Stakeholder>?
        <partOf:BusinessProcess>? <description:String>?

enum BusinessProcessType:
        Manual | User | Service | Send | Receive | Script | BusinessRule | Undefined
```

(lp4)

Linguistic style in some informal CNL:

<name> is a <type> process [,(sent by | receive from | performed by) <participant>]? [(to | from) <participantExternal>]? [, is part of <partOf>]?	(ls4-cnl)

Linguistic style in RSL:

`'businessProcess' name=ID ':' type=BusinessProcessType '['`
`  ('name' nameAlias=STRING)`
`  ('participant' participant=[Stakeholder])?`
`  ('participantExternal' participantTarget=[Stakeholder])?`
`  ('partOf' partOf=[BusinessProcess])?`
`  ('description' description=STRING)?`
`']'`
	(ls4-rsl)

5.3.3 Business Flows

In spite of the aim to keep the business process specification simple, however, there are situations where it is needed to show some form of control flow between business processes or between business processes and events. The BusinessFlow construct supports such situations based on three types of flows: sequential, conditional (equivalent to exclusive-gateway in BPMN), and parallel (equivalent to paralell-gateway in BPMN). Table 6 summarizes the fragments included in a BusinessFlow specification.

Table 6. Summary of BusinessFlow fragments.

Business Flow	Multi-plicity	Description	Type/Values
id	1	unique identifier of the element	ID
name	1	name of the element	String
type	1	type of the element	BusinessFlowType: Sequence, SequenceConditional, Parallel
reference For Business ProcessesOr Events	*	a set of references for processes and events	[BusinessProcess] | [BusinessEvent]
description	0..1	general description of the element	String

Linguistic pattern:

`BusinessFlow::`
`  <id:ID> <name:String> <type:BusinessFlowType>`
`  [<referenceEvent:BusinessEvent> | <referenceProcess:BusinessProcess>]*`
`  <description:String>?`
 	
`enum BusinessFlowType::`
`  Sequence | SequenceConditional | Parallel`
	(lp5)

Linguistic style in some informal CNL:

<id> is a <type> flow with([<referenceEvent> | <referenceProcess>]*).	(ls5-cnl)

Linguistic style in RSL:

`'businessFlow' name=ID ':' type=BusinessFlowType '['`
`  ('condition' condition=STRING)?  // only if type SequenceConditional`
`  'activeElements' bAEs=RefBAE`
`  ('description' description=STRING)?`
`']';`
	(ls5-rsl)

5.3.4 Application Example

Example: Business processes, events and flows for the BillingSystem SRS

Considering the following snippet adapted from the original BillingSystem description (Section 4):

> *The **creation of invoices** is a shared task performed by the user-operator and the user-manager. System shall allow user-operator to **create new invoices** (with respective invoice details). Before sending an invoice to a customer, such invoice **must be formally approved** by the user-manager. Only after such approval, the user-operator can select and send that invoice electronically by e-mail.*
>
> *The System shall **automatically alerts** the user-operator for all the invoices that were sent to customers but not yet paid, after 30 days of their respective approval date. The System needs to **archive all paid or rejected invoices** which creation date is older than 2 years.*
>
> *The BillingSystem shall send periodically (e.g., monthly) all approved and paid invoices to the ERP-Accounting external system. […]*

Some textual fragments are particularly identified and annotated, namely:

- The processes or tasks (**bold** text marked with yellow (light gray) background color), e.g., create invoice, approve invoice, issue invoice, send invoice to the customer;
- The participants of such processes (underlined text), that should be previously defined as stakeholders, e.g., user-operator, user-manager, customer, BillingSystem, ERP-Accounting system;
- The events (text marked with green (dark gray) background color), e.g., invoice sent, monthly, every day, invoices not payed after 30 days of their respective issue date;

Text analysis and preliminary specification

From a text analysis like this, someone shall identify (and then specify) the main processes and events in a systematic way.

On one hand, she shall identify the events that may occur in the scope of the business. In addition, she may tag each event with the respective types and participants, for example:

- ev1: 1st day of each month [type/Timer] [participant/BillingSystem]
- ev2: Invoice not paid after 30 days of their respective approval [type/Conditional] [participant/BillingSystem]

- ev3: Invoices' s creation date older than 2 years [type/Conditional] [participant/BillingSystem]

On the other hand, she shall identify the main (top-level) processes, and then decompose them into simpler (low-level) processes. In addition, she shall tag each process with the respective type, and the participant responsible for it, for example:

- p1: Create and send approved invoice [type/undefined]:
    - p1_1: Register invoice [type/user] [participant/user-operator]
    - p1_2: Analyse invoice to approve [type/manual] [participant/user-manager]
    - p1_3: Register invoice approval [type/user] [participant/user-manager]
    - p1_4: Select and print invoice [type/user] [participant/user-operator]
    - p1_5: Send invoice by email [type/send] [participant/user-operator] [participantExternal/customer]
- p2: Notify customer of invoices to be paid [type/send] [participant/BillingSystem] [participantExternal/customer]
- p3: Archive old invoices [type/service] [participant/BillingSystem]
- p4: Export approved and paid invoices [type/send] [participant/BillingSystem] [participantExternal/ERP-Accounting]

CNL-based specification

From the preliminary specification in natural language, as shown above, someone can specify the business behavior of the BillingSystem in a CNL style as aligned with the linguistic patterns above (i.e., lp-cnl-3, lp-cnl-4 and lp-cnl-4) as follows:

```
//events

ev1_1stDayOfMonth (1st Day of the Month) is a Timer event, occurs in the scope of
TheBillingSystem.
ev2_InvoiceNotPaid (Alert Invoice NotPaid) is a Conditional event, occurs in the scope of
TheBillingSystem.
ev3_OldInvoices (Invoices to Archive) is a Conditional start event, occurs in the scope of
TheBillingSystem.


//processes and flows

p1 (Create and Send Invoice) is an Undefined process.
p1_1 (Register invoice) is a User process, performed by Operator, is part of p1.
p1_2 (Analyse invoice to approve) is a Manual process, performed by Manager, is part of p1.
p1_3 (Register invoice approval) is a User process, performed by Manager, is part of p1.
p1_4 (Select and print invoice) is a User process, performed by Operator, is part of p1.
p1_5 (Send invoice by email) is a Send process, sent by Operator to Customer, is part of p1.
bf1 is a Sequence flow with (p1_1, p1_2, p1_3, p1_4, p1_5).

p2 (InvoicesToBePaid) is a Send process, sent by TheBillingSystem to Customer.
bf2 is a Sequence flow with (ev2_InvoiceNotPaid, p2).

p3 (ArchiveOldInvoices) is a Service process, performed by TheBillingSystem.
bf3 is a Sequence flow with (ev3_OldInvoices, p3).

p4 (ExportPaidInvoices) is a Send process, sent by TheBillingSystem to ERP-Accounting.
bf4 is a Sequence flow with (ev1_1stDayOfMonth, p4).
```

RSL-based specification

Also from the specification in both NL and CNL above, someone can specify the business behavior of the BillingSystem in RSL as follows:

```
// events
businessEvent ev1_1stDayOfMonth : Timer [name "1st Day of the Month" participant stk_system]
businessEvent ev2_InvoiceNotPaid : Conditional [name "Alert Invoice NotPaid" participant
stk_system]
businessEvent ev3_OldInvoices : Conditional [name "Old Invoices to Archive" participant
stk_system]

// processes and flows
// p1: Create and send approved invoice with sub-processes
businessProcess p1 : Undefined [ name "Create and Send Invoice"]
businessProcess p1_1 : User [name "Register invoice" participant stk_operator partOf p1]
businessProcess p1_2 : Manual [name "Analyse invoice to approve" participant stk_manager partOf
p1]
businessProcess p1_3 : User [name "Register invoice approval" participant stk_manager partOf p1]
businessProcess p1_4 : User [name "Select and print invoice" participant stk_operator partOf p1]
businessProcess p1_5 : Send [name "Send invoice by email"  participant stk_operator
participantExternal stk_customer partOf p1]
businessFlow bf1_1 : Sequence [activeElements p1_1, p1_2, p1_3, p1_4, p1_5]

// p2: Notify customer of invoices to be paid
businessProcess p2: Send [name "InvoicesToBePaid" participant stk_system
participantExternal stk_customer]
businessFlow bf2 : Sequence [activeElements ev2_InvoiceNotPaid, p2]

// p3: Archive old invoices [type/service] [participant/BillingSystem]
businessProcess p3: Service [name "ArchiveOldInvoices" participant stk_system]
businessFlow bf3 : Sequence [activeElements ev3_OldInvoices, p3]

// p4: Export approved and paid invoices
businessProcess p4: Send [name "ExportPaidInvoices" participant stk_system participantExternal
stk_erpAcc]
businessFlow bf4 : Sequence [activeElements ev1_1stDayOfMonth, p4]
```

5.4 BusinessGoals View

BusinessGoals view purpose is to answer to following key issues: why is this system needed from a business perspective? how can coarse-grained, high-level business goals be decomposed later on into more concrete requirements? what is the priority level of each business goal? Also, it allows establishing a bridge between the system-of-interest's capabilities and its business context.

The BusinessGoals view depends on the Stakeholders view: each business goal shall define who the stakeholder is (that shall be previously defined in the Stakeholders view). On the other hand, system-level constructs, like Goals, FRs, QRs, Constraints, etc., might have relationships with BusinessGoals.

BusinessGoals should be classified by its own type (BusinessGoalType) namely as Abstract or Concrete respectively depending on weather the goals are defined in a generic or in a more concrete and objective way. Table 7 summarizes the fragments included in a BusinessGoal specification.

Table 7. Summary of BusinessGoal fragments.

BusinessGoal	Multi-plicity	Description	Type/Values
id	1	unique identifier of the element	ID
name	1	name of the element	String
type	1	type of the element	BusinessGoalType: Abstract, Concrete
stakeholder	0..1	reference for the stakeholder who stated this business goal	[Stakeholder]
partOf	0..1	aggregation relation between business goals	[BusinessGoal]
priority	0..1	priority level of this requirement	Priority: Must, Should, Could, Won't (other priority levels shall be defined if more appropriated)
description	0..1	general description of the element	String

Linguistic pattern:

```
BusinessGoal::
       <id:ID> <name:String> <type:BusinessGoalType>
       <stakeholder:Stakeholder>? <partOf:BusinessGoal>? <priority:PriorityType>?
       <description:String>?

enum BusinessGoalType::
       Abstract | Concrete
```

(lp6)

Linguistic style in some informal CNL:

<name> (<id>) is a <type> business goal [, defined by <stakeholder>]? [, is part of <partOf>]? [, priority <priority>]? [, described as "<description>"]?.

(ls6-cnl-a)

Linguistic style in another informal CNL:

<name> (<id> is a <type> business goal [, defined by <stakeholder>]? [, is part of <partOf>]? [, priority <priority>]?).

(ls6-cnl-b)

Linguistic style in RSL:

```
'businessGoal' name=ID ':' type=BusinessGoalType '['
       ('name' nameAlias=STRING)
       ('stakeholder' stakeholder=[Stakeholder])?
       ('partOf' partOf=[BusinessGoal])?
       ('priority' priority=PriorityType)?
       ('description' description=STRING)?  ']';
']'
```

(ls6-rsl)

Example: Business goals for the BillingSystem SRS

Consider the following snippet adapted from the original BillingSystem description (Section 4):

The BillingSystem shall be developed in a project involving the following companies: MySoftwareHouse as the performing organization, and TheCustomer as the pilot customer organization meaning the organization that shall use this system. The owner of this system is the MySoftwareHouse organization that intends to distribute it in different countries with different languages. Also due to that reason, the system shall be flexible, easy to configure and to use by different types of end-users.

This project shall adopt the Scrum as the management and development process; and the system v1.0 shall be released before the end of current year.

After 6 months of the official version 1.0 released the MySoftwareHouse organization intends to sell and implement the BillingSystem for more than 500 customers in its own country and 1000 customers in EU region.

From a text analysis like this the analyst can identify (and then specify) the business goals in a systematic way. Some sentences can easily be identified as business goals (e.g., "the product shall be sold to more than 500 customers"; or "the owner organization shall be the MySofwareHouse"), while others can be better categorized as system goals (and not business goals), such as "the system shall be flexible, easy to configure and to use" and so, they shall not be identified as business goals. In addition, some other sentences (e.g., "this project shall adopt the Scrum as the management and development process") can be or not be classifying as business goals depending on the pragmatic rules defined by the specific organization. Furthermore, for each business goal, the business analyst shall define the respective type, stakeholder and priority, if relevant. For example:

- bg1: MySoftwareHouse is the performing organization and the owner of the system [type/Concrete] [stakeholder/ MySoftwareHouse]
- bg2: TheCustomer is the pilot customer organization [type/Concrete] [stakeholder/ MySoftwareHouse]
- bg3: Distribute the BillingSystem in different countries with different languages [type/Abstract] [stakeholder/ MySoftwareHouse]
- bg4: The development project for building this system shall adopt the Scrum process type/Concrete] [stakeholder/project-manager]
- bg5: The product shall be deployed to many customers [type/Abstract] [stakeholder/ MySoftwareHouse]
    - bg5.1: The product shall be deployed to more than 500 national customers [type/Concrete] [stakeholder/ MySoftwareHouse]
    - bg5.2: The product shall be deployed to more than 1000 EU customers [type/Concrete] [stakeholder/ MySoftwareHouse]

CNL-based specification

From the preliminary specification in natural language, as shown above, someone can specify the business goals in some CNL styles as aligned with the linguistic patterns above as follows:

```
// first writing style (ls6-cnl-a)
bg1_Owner is a Concrete business goal, defined by mySoftwareHouse, priority Must, described as
"MySoftwareHouse shall be the performing organization and the owner of this project".

bg2_PilotCustomer is a Concrete business goal, defined by mySoftwareHouse, priority Could,
described as "TheCustomer shall be the pilot customer organization of this project".
```

```
bg3_MultipleCountries is a Concrete business goal, defined by mySoftwareHouse, priority Should,
described as "The BillingSystem shall be distributed in different countries with different
languages".

bg4_ScrumProcess is a Concrete business goal, defined by ProjectManager, described as "The Scrum
process shall be adopted to manage this project".

bg5_NrOfCustomers is a Abstract business goal, described as "After 6 months of the v.1 released,
the product shall be deployed into many customers".

bg5_1_NrNationalCustomers is a Concrete business goal, is part of bg5_NrOfCustomers, described as
"... more than 500 national customers".

bg5_2_NrEUCustomers is a Concrete business goal, is part of bg5_NrOfCustomers, described as "...
more than 1000 EU customers".


// second writing style (ls6-cnl-b)
MySoftwareHouse shall be the performing organization and the owner of the project (bg1_Owner is a
Concrete business goal, defined by mySoftwareHouse, priority Must).

TheCustomer shall be the pilot customer organization of the project (bg2_PilotCustomer is a
Concrete business goal, defined by mySoftwareHouse, priority Could).

The BillingSystem shall be distributed in different countries with different languages
(bg3_MultipleCountries is a Concrete business goal, defined by mySoftwareHouse, priority Should.

The Scrum process shall be adopted to manage the project (bg4_ScrumProcess is a Concrete business
goal, defined by ProjectManager).

After 6 months of the v.1 released, the product shall be deployed into many customers
(bg5_NrOfCustomers is a Abstract business goal).

... more than 500 national customers (bg5_1_NrNationalCustomers is a Concrete business goal, is
part of bg5_NrOfCustomers).

... more than 1000 EU customers (bg5_2_NrEUCustomers is a Concrete business goal, is part of
bg5_NrOfCustomers).
```

RSL-based specification

Also from the specification in both NL and CNL above, someone can specify the business goals of the BillingSystem in RSL as follows:

```
businessGoal bg1_Owner: Concrete [name "MySoftwareHouse shall be the performing organization and
the owner of the system" stakeholder stk_mySoftwareHouse priority Must]

businessGoal bg2_PilotCustomer: Concrete [name "TheCustomer shall be the pilot customer
organization" stakeholder stk_mySoftwareHouse priority Could]

businessGoal bg3_MultipleCountries: Abstract [name "The BillingSystem shall be distributed in
different countries with different languages" stakeholder stk_mySoftwareHouse priority Should]

businessGoal bg4_ScrumProcess: Concrete [name "The Scrum process shall be adopted to manage the
project" stakeholder stk_projectmanager]

businessGoal bg5_NrOfCustomers: Abstract [name "The product shall be deployed to many customers"
stakeholder stk_mySoftwareHouse description "after 6 months of the v.1 released, the product
..."]

businessGoal bg5_1_NrNationalCustomers: Concrete [name "... more than 500 national customers"
stakeholder stk_mySoftwareHouse partOf bg5_NrOfCustomers description "the system shall have more
than 500 national customers (implementations)"]
```

```
businessGoal bg5_2_NrEUCustomers: Concrete [name "... more than 1000 EU customers" stakeholder
stk_mySoftwareHouse partOf bg5_NrOfCustomers description "the system shall have more than 1000
customers (implementations), in EU region"]
```

6. CONCLUSION

SRS describes multiple technical concerns of a system and is used throughout several stages of the project life-cycle. SRS helps sharing the system vision among the main stakeholders, as well to facilitate the communication, project management and system development processes. For an effective communication, everyone should be able to communicate by means of a common language, and natural language provides the foundation for such language. However, although natural language is the most common and preferred form of requirements representation, it also exhibits some intrinsic characteristics that often present themselves as the root cause of many requirements quality problems, such as incorrectness, inconsistency, incompleteness and ambiguousness.

RSL is a requirements specification language that includes several constructs logically arranged into views according to the specific RE concerns they address. Each of these constructs is defined as a linguistic pattern and represented by a corresponding RSL's linguistic style. In addition, as shown throughout the paper, these constructs can be also represented by other linguistic styles according multiple control natural languages that can be considered. In that way, RSL can be also considered an intermediate format from which the other representations can be converted to, and vice-versa.

RSL can be used and applied by different users such as requirement engineers or business analysts. They can produce SRSs at different levels of detail, and also using different types of requirements, specifically: business goals, system goals, functional requirements, quality requirements, constraints, user stories, and use cases. In addition, they also use other types of related constructs (e.g., terms, stakeholders, actors, data entities, state machines) that, in spite of not being classified in RSL as requirements (see Table 1), are still important to complement and enrich the specification of such requirements. Due to space constraints, this paper focused the discussion only on the RSL's business level constructs and views. However, system-level constructs and views are already discussed in other publication (Silva, 2017). In spite of other proposals for RE-specific templates or linguistic patterns, and as far as we know from the literature review, RSL is the first of such proposals that provides such a comprehensive language that integrates a large number of constructs, that can be represented in a consistent and systematic way by multiple linguistic patterns and styles.

RSL is currently supported by several software tools, namely an Excel spreadsheet template (publicly available at github[3]), an Eclipse and XText-based tool (the RSLingo-Studio tool[4]), and a web-based collaborative environment. In addition, RSL's linguistic patterns could be adapted with variations of the presented CNL linguistic styles (i.e., ls-cnl-i), that would be, in many cases, more easily written and read by business stakeholders.

ACKNOWLEDGEMENTS

This work was partially supported by national funds under FCT projects UID/CEC/50021/2013 and CMUP-EPB/TIC/0053/2013. Thanks to Lise Hvatum, our EuroPLoP'2017 shepherd, for her relevant criticism and suggestions that helped to improve the paper.

[3] http://itbox.inesc-id.pt/ITLingo/RSLingo
[4] http://itbox.inesc-id.pt/ITLingo/RSLingo

REFERENCES

Bettini, L., 2016. Implementing Domain-Specific Languages with Xtext and Xtend. Packt Publishing Ltd.

Cockburn, A., 2001. Writing Effective Use Cases. Addison-Wesley.

DeCapua, A., 2008. Grammar for Teachers: A Guide to American English for Native and Non-Native Speakers, Springer.

Durán, A., Bernárdez, B., Toro, M., Corchuelo, R., Ruiz, A., & Pérez, J., 1999. Expressing customer requirements using natural language requirements templates and patterns. In IMACS/IEEE CSCC'99 Proceedings.

Eveleens, L., Verhoef, C., 2010. The Rise and Fall of the Chaos Report Figures, IEEE Software.

Fernandes, J.M., Machado, R. J., 2016. Requirements in engineering projects, Springer.

Ferreira, D., Silva, A. R., 2012. RSLingo: An information extraction approach toward formal requirements specifications, Proceedings of MoDRE'2012, IEEE CS.

Ferreira, D., Silva, A. R., 2013. RSL-PL: A Linguistic Pattern Language for Documenting Software Requirements, in Proceedings of RePa'13, IEEE CS.

Ferreira, D., Silva, A. R., 2013a. RSL-IL: An Interlingua for Formally Documenting Requirements, in Proc. of the of Third IEEE International Workshop on Model Driven Requirements Engineering, IEEE CS.

IEEE 1998. IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications.

Kovitz, B. 1998. Practical Software Requirements: Manual of Content and Style. Manning.

Lamsweerde, A., 2009. Requirements Engineering: From System Goals to UML Models to Software Specifications. Willey.

Pohl, K., 2010. Requirements Engineering: Fundamentals, Principles, and Techniques, Springer.

Ribeiro, A., Silva, A. R., 2014. XIS-Mobile: A DSL for Mobile Applications, Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC).

Ribeiro, A., Silva, A. R., 2014a. Evaluation of XIS-Mobile, a Domain Specific Language for Mobile Application Development, Journal of Software Engineering and Applications, 7(11), pp. 906-919.

Robertson, S. and Robertson, J., 2006. Mastering the Requirements Process, 2nd edition. Addison-Wesley.

Savic, D., et al, 2015. SilabMDD: A Use Case Model Driven Approach, ICIST 2015 5th International Conference on Information Society and Technology.

Silva, A. R., 2015. Model-Driven Engineering: A Survey Supported by a Unified Conceptual Model, Computer Languages, Systems & Structures 43 (C), 139–155.

Silva, A. R., 2015a. SpecQua: Towards a Framework for Requirements Specifications with Increased Quality, in Lecture Notes in Business Information Processing (LNBIP), LNBIP 227, Springer.

Silva, A. R., et al, 2015. A Pattern Language for Use Cases Specification, in Proceedings of EuroPLOP'2015, ACM.

Silva, A. R., Saraiva, J., Ferreira, D., Silva, R., Videira, C. 2007. Integration of RE and MDE Paradigms: The ProjectIT Approach and Tools, IET Software, IET.

Silva, A. R., Saraiva, J., Silva, R., Martins, C., 2007. XIS – UML Profile for eXtreme Modeling Interactive Systems, in Proceedings of MOMPES'2007, IEEE Computer Society.

Silva, A. R., Verelst, J., Mannaert, H., Ferreira, D., Huysmans, P., 2014. Towards a System Requirements Specification Template that Minimizes Combinatorial Effects, Proceedings of QUATIC'2014 Conference, IEEE CS.

Silva, A. R., A Rigorous Requirement Specification Language for Information Systems: Focus on RSL's Use Cases, Data Entities and State Machines, INESC-ID Technical Report, 2017.

Sommerville, I. and Sawyer, P. 1997. Requirements Engineering: A Good Practice Guide. Wiley.

Stahl, T., Volter, M., 2005. Model-Driven Software Development, Wiley.

Standish Group, 2009. Chaos Summary 2009 Report, The 10 Laws of Caos.

Verelst, J., Silva, A.R., Mannaert, H., Ferreira, D., Huysmans, 2013. Identifying Combinatorial Effects in Requirements Engineering. In Proceedings of Third Enterprise Engineering Working Conference (EEWC 2013), Advances in Enterprise Engineering, LNBIP, Springer.

Videira, C., Silva, A. R., 2005. Patterns and metamodel for a natural-language-based requirements specification language. CAiSE Short Paper Proceedings.

Videira, C., Ferreira, D., Silva, A. R., 2006. A linguistic patterns approach for requirements specification. Proceedings 32nd Euromicro Conference on Software Engineering and Advanced Applications (Euromicro'2006), IEEE Computer Society.

Withall, S., 2007. Software Requirements Patterns, Microsoft Press.