Variability Aspects at a Textual Requirements Specification Level

Alberto Rodrigues da Silva, João Fernandes, Sofia Azevedo
INESC-ID, Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal

***Abstract*—Since the advent of software product lines (SPLs), variability techniques have provided for commonality and variability modeling of functionally similar products, within a given domain. However, variability modeling proposals so far have mostly been targeted at system features, rather than its requirements, stemming from the fact that features are often closer to stakeholders' perception and understanding of variability. Given the importance that a well-defined system requirements specification (SRS) represents to the success of a project, this paper proposes an innovative approach for modeling and managing variability at the SRS level, based on the Common Variability Language (CVL) which is the OMG proposal for a domain-independent variability modeling standard. This approach has been implemented as a core feature of the ITBox system, a Web-based collaborative platform for the management of SRSs.**

Keywords—requirements specification; variability modeling; reusability; product line

I. Introduction

During the development of software projects, it is common that some of the original features are adapted in order to meet specific customers' needs. To manage this variability, companies often begin with developing customer-specific versions of their original products, changing and adding features for each new version. However, as the number of versions increases so does the effort needed to model and maintain all the variation points and respective variants (alternatives for each point). In some cases, the sheer number of these variation points increases the complexity of the project to the point of making it almost impossible to manage.

Since its first introduction, *feature modeling* [11] has been the most popular technique to model commonality and variability of products of a product line. C&V (Commonalities and Variabilities) are modeled from the perspective of product features: "stakeholder visible characteristics of products" in a product line that shall support multiple stakeholders' needs and concerns [4]. With the advent of SPLs (Software Product Lines) for building multiple and related products in a given domain, C&V models have equally increased in popularity. Since its inception, researchers have proposed and implemented numerous extensions and enhancements to the original FODA model [3]. The CVL (Common Variability Language) comes as the OMG's latest proposal as a Variability Modeling standard, defining itself as a domain-independent language for specifying and resolving variability [17]. In practice, CVL allows representing the variability of system features regardless of the DSL (Domain-Specific Model) used to model those features. Furthermore, CVL also introduces a language for expressing constraints over its variability specifications based on the OCL (Object Constraint Language) [13].

The process of variability modeling is still very closely associated with the concept of feature modeling as found in SPLs. However, we and other authors argue that this process should be supported and started early at the RE (Requirements Engineering) level [2,19,20,21]. Requirements related tasks are the first step in any software project, and so variability concerns shall also be clearly expressed, given the extreme importance that a well-defined SRS (System Requirements Specification) document can represent in the success of a project [5]. It is, therefore, important to more formally define a means to express variability concerns at a requirements level. Despite this, little research has investigated software requirements variability so far, particularly at textual level, since many software companies prefer to express their product capabilities in terms of "features which is closer to how designers represent variability [3]. While a feature model tends to break-down a product in just one and-or hierarchy of functional capabilities, a requirements specification tends to define multiple requirements of the product not necessarily by just one hierarchy. This happen because the product requirements are cross-cutting, involving multiple abstraction levels, constructs and perspectives. For example, a feature model for a traditional "billing system" can be break-down in features such as "customers", "suppliers", "invoices", "payments", "products and services"; while a common requirements specification for this same product can involve, in addition, cross-cutting aspects such as non-functional requirements (security, availability, usability, etc.).

This paper proposes an innovative approach for leveraging the concepts of CVL and its domain-independence to model variability aspects in the context of requirements engineering. This proposal presents the approach for applying the language's central variability notions to structured SRS documents, defined in a rigorous RSL (Requirements Specification Language).

This approach has been developed as a core feature of the ITBox system, a web-based collaborative platform for requirements management with a focus on *requirements reusability based on C&V modeling*. ITBox is also introduced in this paper to better support the presentation and discussion of the ideas proposed.

The structure of this paper is as follows. Section II introduces the context and technologies related this paper's background, namely the RSLingo initiative, the RSL language, and the OMG CVL standard. Section III introduces the ITBox collaborative platform and Section IV details its CVL-based variability approach to requirements modeling. Section V further illustrates the concepts introduced though the use of a simple but effective example. Section VI refers some releated work. Finally, Section VI presents the conclusion of the paper and lays down some ideas for future work.

978-1-5386-3488-2/17 $31.00 © 2017 IEEE
DOI 10.1109/REW.2017.59

II. BACKGROUND

This section introduces the context and some technologies underlying this paper, namely the RSLingo initiative, the RSL language, and the OMG CVL standard.

A. RSLingo Initiative

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 requirements documents – is prone to produce ambiguous and inconsistent documents that are hard to automatically validate or transform.

Originally RSLingo proposed an approach to use simplified NLP (Natural Language Processing) 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 [6]. This is achieved through the use of two original languages: the RSL-PL (Pattern Language) [7], designed for encoding RE-specific linguistic patterns, and RSL-IL (Intermediate Language) [8], a domain specific language designed to address RE concerns and better support RE-related tasks. 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 structure format, reducing its original ambiguity and creating a new and more rigorous SRS document [9].

This approach foresees the whole process of knowledge extraction and conversion to a more rigorous representation as a way to help business stakeholders having 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 its documentation process.

B. RSL

Recently we designed a broader and more consistent language called *RSLingo's RSL*[1]" (or just "*RSL*" for brevity) [10]. RSL is based on the design of the former languages, i.e., ProjectIT-RSL [11], RSL-IL [8], RSL-PL [7], but also others such as Pohl [12], XIS* [13, 14, 15] and SilabREQ [16].

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[2] framework (Bettini, 2016) and so, its specifications are rigorous and can be validated and transformed (into other representations and formats) automatically. A lightweight tool support is also provided based on an "RSLingo RSL Excel Template"[3] publicly available at github. This Excel template encodes the different viewpoints within respective sheets, defining a set of properties for each one. As an example, the key properties of the Goals View are shown in Table 1.

TABLE I. PROPERTIES OF THE RSL'S GOALS

Name	Description	Type/Values
Id	Unique Identifier of the goal	string
Name	Descriptive name of the goal	string
Type	Type of the goal	Concrete; Abstract
Source	Reference to the stakeholder who had or defined the goal	<stakeholder id>
PartOf	Reference from the current goal to its parent goal	<parent goal id>
Description	Description of the goal	string
Priority	Level of priority of the goal	Must; Should; Could; Won't
Progress	Current status of the development process	Plan; Design; Develop; Test; Deploy; Concluded

C. CVL Standard

The CVL is the OMG proposal of a domain independent language for specifying and resolving variability [17]. It facilitates the specification and resolution of variability over models of any language defined by the MOF-based meta-model framework.

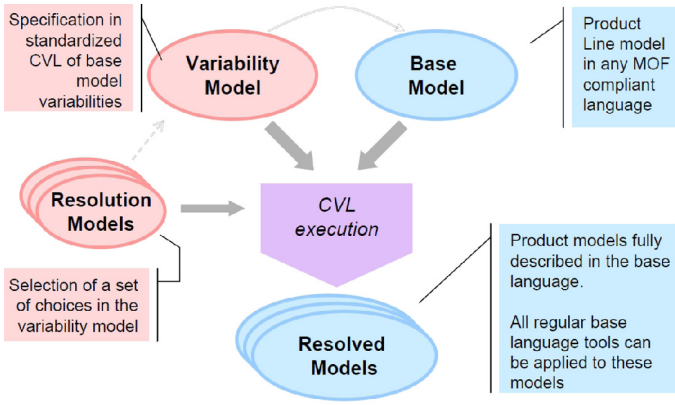

Fig. 1. The CVL models overview (extracted from [17]).

Figure 1 shows the CVL execution (or materialization) process that involves the following models: Base Model, Variability Model, Resolution Models, and Resolved Models.

The *Base Model* is not part of CVL and is the definition of the original product line, defined in any MOF-compliant language. This is what makes CVL a domain-independent language, allowing it to be applied in different contexts.

[1] https://github.com/RSLingo/RSL

[2] http://www.eclipse.org/Xtext/

[3] https://github.com/RSLingo/RSL-Excel-Template

The *Variability Model* is a collection of *variation points (VSpecs)* and *constraints* used to specify variability over the base model. These concepts and the integration between them constitute the core of the CVL language.

The *Resolution Model* consists in a collection of VSpec resolutions, which resolve the VSpecs of a variability model.

Finally, the *Resolved Model* is the new model produced by the materialization process from the base model, variability model and the resolution model.

III. THE ITBOX PLATFORM

ITBox is a web-based collaborative platform for managing requirements specification documents and other technical documentation. Although ITBox can support multiple types of documents, in this paper, it is analyzed according a RE perspective. Fig. 2 shows a screenshot of this platform. ITBox's users can author, review and validate their requirements based on the constructs of the RSL language, but using an *easy to learn and to use interface* such as MS-Excel or Google Sheets. Given that RE processes involved an intense cooperation between many stakeholders (e.g. customers, domain experts, requirements engineers, software developers), the key ITBox 's design goals were the followings:

Provide a collaborative environment for managing requirements documents with an easy to learn and to use interface: The ITBox major purpose is to provide a collaborative editor for software requirements documents. Spreadsheet editors, although lacking many of the fundamental features of the commercial dedicated RE tools, have always been a popular choice to manage requirements specifications due to their widespread availability and simple interface, making them a good baseline tool for nontechnical stakeholders. Until recently, one of the main problems with this desktop software was to maintain synchronized and updated versions of the documents in decentralized projects requiring multiple concurrent changes. However, recent cloud storage services, such as Google Drive or OneDrive, have helped on that regard by offering collaborative mediums to manage these types of documents with automatic synchronization and lightweight versioning tools. As such, the ITBox platform makes use of the Google Drive and Google Sheets APIs to provide both the authoring and a cooperative environment. The first is used for uploading the local SRS documents, sharing them and granting access permission to its users, while the second offers all the data manipulating functions for extracting, and editing the information contained within the platform documents.

Integrate with RSLingo concepts and technologies: As stated before, this includes in particular the adoption of the constructs defined in RSL language and its current Excel representation. Although the platform allows for the usage of any format of SRS document, all the features that involve data manipulation expect the current format of the RSL Excel template.

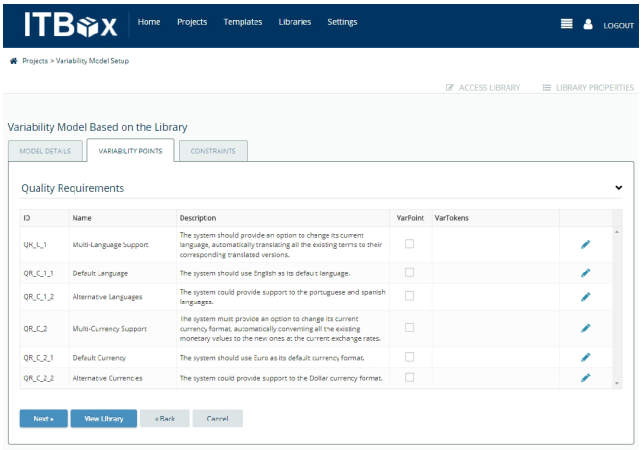

Fig. 2. The ITBox (variability points selection screen).

Include reusability and variability capabilities: To complement the document editor, the system makes use of the data manipulation capabilities provided by the integration with the Google APIs to offer a set of features to increase the efficiently of its processes. The main feature is the variability modeling approach based on CVL that is described in depth in the next section of the paper.

The reusability features provided by ITBox involve the management of *SRS templates* and the management of *SRS libraries.* The former (management of SRS templates) allows any previously developed template (defined as a spreadsheet format) to be uploaded to the platform and later be used to create new SRSs based on the template' structure. The latter (management of SRS libraries) allows the creation of libraries of coarse grained and potentially generic requirements that can afterwards be added and edited if necessary to any new project. The ITBox variability modeling process is particularly applied to this latter situation.

The remaining sections of the paper focus on the Variability Modeling issues as designed and implemented in ITBox.

IV. THE ITBOX VARIABILITY MODELING APPROACH

The ITBox variability modeling process closely follows the proposed CVL approach, as it can be seen in Fig. 3. Complementary, Fig.4 illustrates the main concepts and respective relationships. A detailed description of the process is discussed below.

A. Create Variability Model

At the start of the variability modeling process, the user selects the SRS document (or SRS library) to be used as the Base Model. In ITBox the *base model* is a SRS document created using the RSL language, and in particular based on its RSL Excel Template. This document can either be part of a project already present in the platform, or can be manually uploaded into the system from an outside source, provided it conforms to the RSL language.

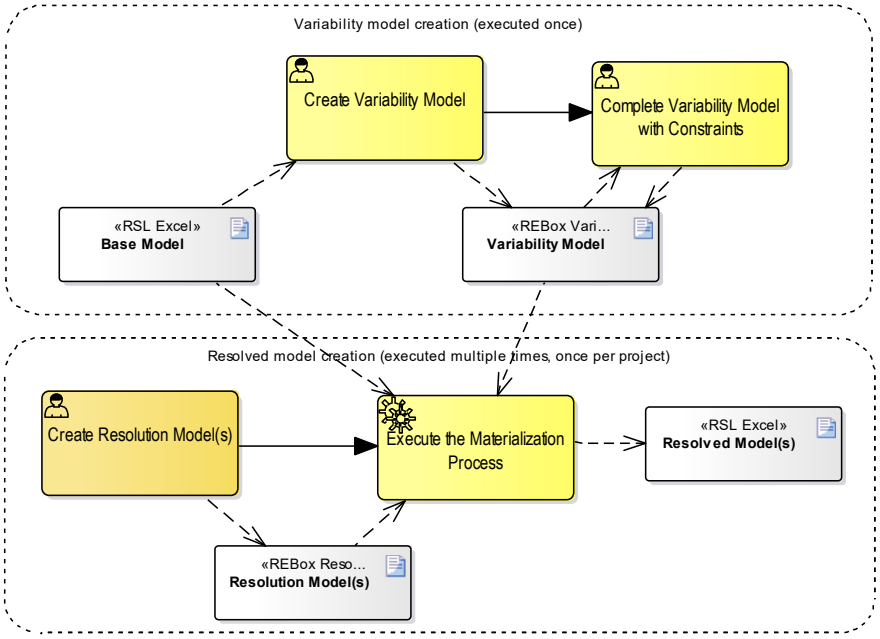

Fig. 3. The ITBox Variability Modeling approach (in BPMN Business Process diagram).

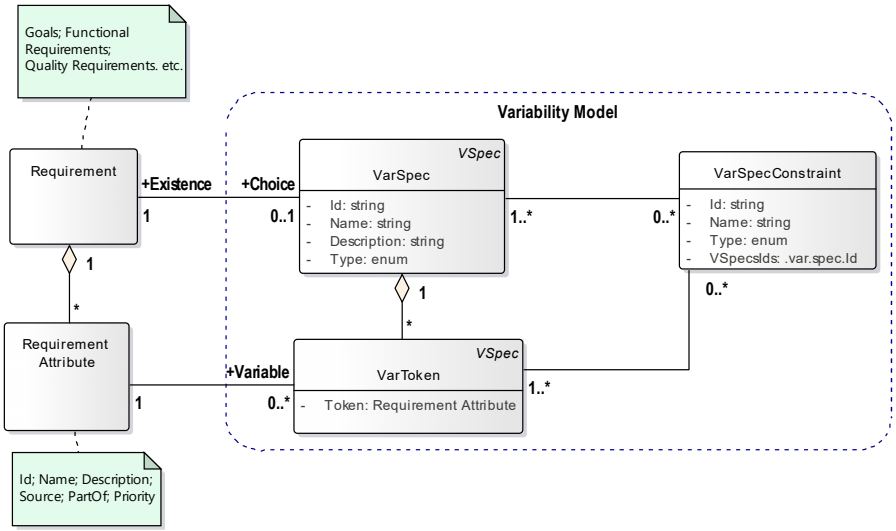

Fig. 4. The ITBox Variability Modeling (domain model in UML).

The system then automatically parses this excel file and extracts the requirements from the RSL supported views (such as Goals, Functional Requirements and Quality Requirements).

Then, regarding these extracted requirements, the user can define the variation points. As defined by CVL, *variation points* are specifications of concrete variability in the base model, defining specific modifications to be applied to it during the materialization process. Variation points can take various types, which define the nature of the dependency between a VSpec and the corresponding Variability Point in the materialization process.

In ITBox, variation points (VSpec) can be defined at two levels: (1) the requirements themselves and (2) their attributes or properties. *Variation points defined over a requirement* are of the Existence type, meaning that in the materialization phase, they will be bound to a Choice VSpec. On the other hand, *variation points defined over a requirement property* are of the Value Assignment type.

After the user defines all the variation points over the base model, the ITBox system automatically generates the first part of the *variability model*: the VarSpecs view. In CVL each variation point references a single VSpec, in a process defined as *binding*, which provides the linkage between variability realization and variability abstraction. Figure 4 details the variability binding model used by the ITBox approach.

The *VarSpecs view* contains all the information about the newly generated VarSpecs and its full list of attributes are presented in Table 2. Note that this view introduces a new concept associated to VarSpecs: the VarToken. *VarTokens* are "children" VarSpecs that are associated with a particular attribute of the requirement that serve as its variation point. A VarSpec can have any number of VarTokens, depending on the number of attributes that were considered as variation points in the last step. Regarding the type, "parent" VSpecs that are associated with requirements are defined as Choices, meaning that in the materialization phase they will require a binary yes/no decision to define if that particular requirement is going to exist in the resolved model. Contrarily, VarTokens are of type Variable, determining that in the materialization process these VSpecs will require providing a resolution value of their specified type.

B. Complete Variability Model with Constraints

Furthermore, the user is then able to define constraints over the VSpecs that were created. In CVL, constraints are used to express intricate relationships among VSpecs that cannot be directly captured by hierarchical relations. To achieve this, it relies on a simplified version of the OCL language. The user can utilize both the parent VarSpecs and their children VarTokens when building the logical expressions that serve to express these constraints. A more indepth example of this process is shown in Section V.

TABLE II. PROPERTIES OF THE VARSPECS VIEW

Name	Description	Type/Values
Id	Unique identifier	String
Name	Descriptive name	String
Type	VarSpec type	VSReqGoal; VSReqFunctional; VSReqQuality
ElementId	The associated Requirement id	< Requirement Id>
ElementName	The associated Requirement name	< Requirement Name>
VarTokens	List of the VarTokens	List of <VarTokens>
Description	Description of the VarSpec	String

TABLE III. PROPERTIES OF THE VARSPECCONSTRAINTS VIEW

Name	Description	Type/Values
Id	Unique Identifier	String
Name	Descriptive name	String
Type	VarSpecConstraint type	Simple; LogicExpression
VarSpecs	List of VarSpecs referred by the constraint	<VarSpecs Ids>
Logical Expression	The propositional logic statement containing the constraint	String
Description	Description of the constraint	String

After all the constraints are created, the system then produces the VarSpecConstraints view, shown in Table 3. With both views created (i.e. VarSpecs and VarSpecConstraints views), the ITBox variability model is concluded.

C. Create Resolution Model and Execute the Materialization Process

The final step in the variability modeling approach is the *materialization process*. As stated in the CVL proposal, this process consists of transforming a base model into a new document by applying variation points. Materialization is driven by a *resolution model* which provides resolutions for the Vspecs defined against the base model. In ITBox , after the Variability Model is created in the previous steps, its information is stored in the system's database. From that moment on, whenever a new SRS document creation process starts, the user is given the option to, instead of creating a new empty document, generate a new one by defining a set of resolution values (i.e. *create the resolution model*) for an existing variability model. When this happens, the platform crosses the information contained in the *variability model* with the resolution model provided by the user and checks every constraint for possible inconsistencies. If nothing is found, a new *resolved model* is produced.

V. SIMPLE EXAMPLE

To better support the explanation of the proposed approach, this section introduces a simple but effective example, depicted in Fig. 5. This example describes a fictional *set of reliability-specific quality requirements*, which defines its expected uptime over the period of 1-year, required maintenance procedures and fault logging details (see Table IV). This example describes the process variability model creation (as depicted in Fig. 3) that has as input the base model and as output the variability model.

TABLE IV. BASE MODEL

A. Create a Variability Model

1) Base Model

The first table of the Fig.5 shows an exact representation of a RSL Excel template Quality Requirements view. This view defines a set of attributes to characterize the requirement, such as unique id, name, type and sub-type, quantitative metric and its value (if applicable), stakeholder who proposed the requirement, its parent requirement (if applicable), description, and finally its priority.

Id (*)	Name (*)	Description
QR_R_1	System Uptime	The system should assure an annual Uptime of 99%.
QR_R_2	System Maintenance	The system should have a weekly maintenance period of 1 hour.
QR_R_2_1	Maintenance Actions	The maintenance should be performed by a technical administrator of the system and should consist of a verification of the system fault log and any other reported problems that occurred since the last maintenance.
QR_R_3	Fault Logging	The system must register all the faults occurred at runtime in a specific log file.
QR_R_3_1	Fault Log Information	Each logged fault must contain the file and line where the fault originated, the date and hour of the occurrence and a stack dump from the moment when it ocurred.

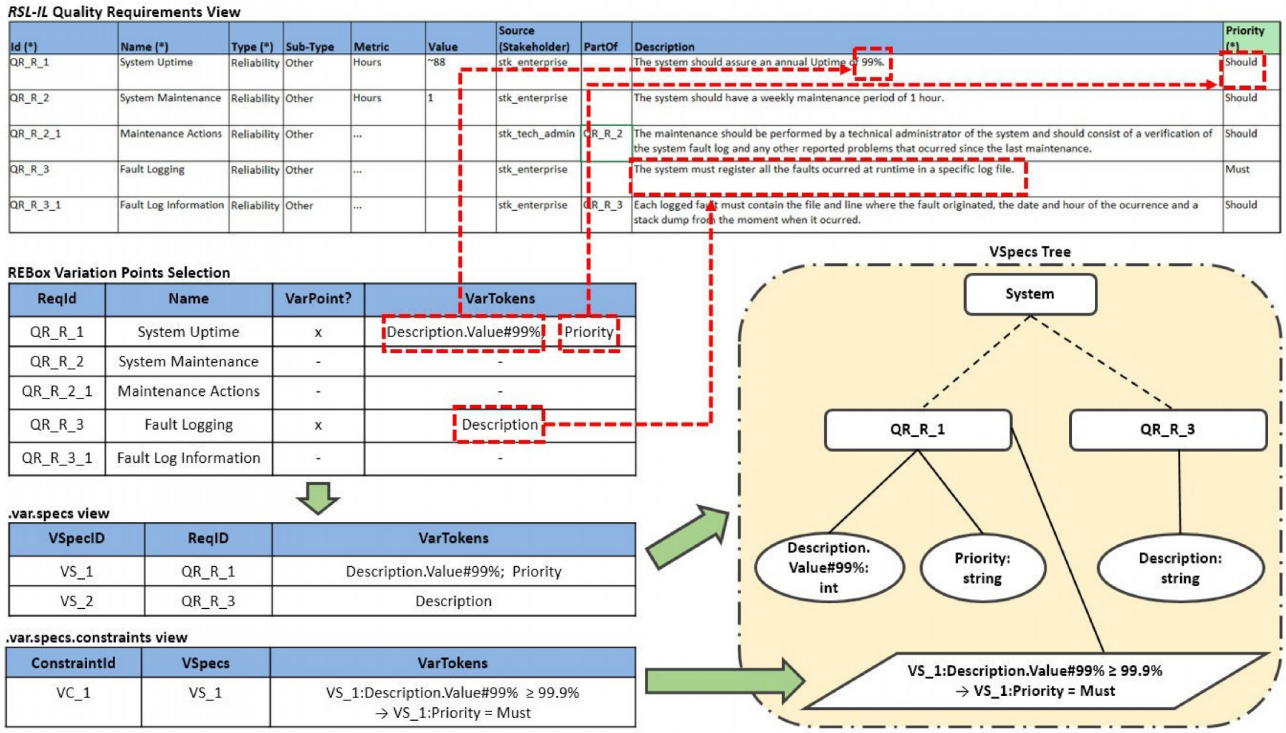

Fig. 5. An example of ITBox 's Variation Points definition, the associated VSpecs and Constraints, and a representation of the corresponding VSpecs Tree.

2) Variability Model

The variability model produced according this process includes tables 3 (VarSpecs) and 4 (VarSpecConstraints) of Fig.5.

The second table of the Fig.5 shows an example of the "Create Variability Model" task (as illustrated in Fig. 3). As described before, this task starts with the automatic extraction of the Goals, Functional Requirements and Quality Requirements from the base model defined in the RSL language and consists in the manual configuration of the model's variation points. To set a requirement as a variation point, the user just needs to check the "`VarPoint?`" field on that requirement's row. This action will mark that requirement as an Existence variation point, meaning that it be linked to to a Choice VSpec in the Binding phase. Furthermore, field "`VarTokens`" lets the user select which of that requirement's attributes will also be modeled as children variation points of the type Value Assignment. In the example, from the five initial requirements, two were selected as variation points: `QR_R_1` and `QR_R_3`. The first one has two attributes modeled as children variation points: "`Description`" and "`Priority`". However, not all of the content of the "`Description`" field is to be considered for variability purposes, but only the substring "`99%`", denoted by the suffix "`.value#99%`".

The third table of Fig.5 shows a simplified version of the VarSpecs view (for the sake of space, some attributes were omitted; the full list is displayed in Table II). This view is automatically generated by the ITBox system after the user selects the variation points in the previous step, and in this cimple example it includes `VS_1` and `VS_2`. This process corresponds to the binding process of CVL, where each of those points is linked to their abstract representation in the variability model. To the right of this third table is an illustrative example of the corresponding VSpecs tree representation, which shows the six VSpecs created based on the input provided in the second table. The parent VSpec corresponds to the system itself. On the second row, the `QR_R_1` and `QR_R_3` requirements are represented as Choice VSpecs, meaning that in the materialization process, both resolutions will require a binary yes/no decision. These elements are linked to their parent by a dashed edge, representing a false "IsImpliedByParent" value, making their materialization value independent of the one their parent takes. Finally, on the third level of this figure are shown three children VSpecs of type Variable, generated by the VarTokens defined in the third table and their corresponding types.

The fourth table of Fig.5 presents a simplified version of the VarSpecConstraints view (again, the full list of fields can be viewed in Table III). Like the name implies, this table defines the Constraints associated with the particular variability model. However, unlike the previous table, the constraints are not generated by the system (nor could they). Based on the VSpecs defined in the VarSpecs table and on the context of the

system, it is up to the user to define the needed logical expressions to capture intricate relationships among VSpecs that cannot be simply expressed by the hierarchical relations in a VSpec tree. This concrete example defines a propositional constraint determining that if (in the materialization process) the annual uptime of the system is set as greater than or equal to 99.9% then, at the same time, the value defined for the Priority VSpec has to be "Must".

Note that in this example both the VSpecs that were involved in the constraint were children of the same VSpec (QR_R_1, with id VS_1). As such, the constraint is defined in the context of the parent node, denoted by the solid edge. It is possible, however, to combine multiple parent VSpecs and their children (e.g. VS_1 → VS_2:Description = "something", meaning that if the QR_R_1 is resolved as true (as defined by VS_1), then the Description of QR_R_3 (as defined by VS_2) must be resolved to the string "something"). In this case, the context of the constraint should be set at the whole model level and therefore it should not be connected to any specific VSpec.

B. Create Resolved Models

As depicted in Fig.3, several concrete resolved models can semi-automatically be produced from the base and variability models defined and on an appropriate tool support.

For instance, in this simple example two resolved models could be produced for two distinct systems (A and B) as suggested in the Tables V and VI, where the yellow (gray) background of some text snippets express such materialization process. For instance, the resolved model for system-A (Table V) included two QRs provided from the base+variability models: QR_R_1 and QR_R_2. In addition, for QR_R_1 it was replaced the varToken "Description.Value#99%" by the string "66%", and for QR_R_2 it was replaced the varToken "Description" by the string "The system shall register all the faults occurred at runtime in the System-A log file".

TABLE V. RESOLVED MODEL FOR SYSTEM-A

Id (*)	Name (*)	Description
QR_R_1	System Uptime	The system should assure an annual Uptime of 66%.
QR_R_3	Fault Logging	The system shall register all the faults occurred at runtime in the System-A log file.
...	...	…

TABLE VI. RESOLVED MODEL FOR SYSTEM-B

Id (*)	Name (*)	Description
QR_R_1	System Uptime	The system should assure an annual Uptime of 90%.
...	...	…

VI. RELATED WORK

In 1998 Coplien et al. argue that the analysis decisions on C&V shall be made during the requirements analysis stage, rather than during the implementation stage by professionals who are not so familiar with the implications and impact of such C&V decisions [21]. They refer that early decisions on C&V contribute to largescale reuse and the automated generation of family members. In 2002 Bosch et al. also mention the need for describing C&V within different modeling levels such as the requirements one [22].

Prior research contributions in what C&V at the analysis stage is concerned, namely at the level of requirements representation (e.g. with use case models) have been reported in various research works e.g. the Variation Point Model in [23] and the Consolidated Variability Metamodel in [24]. The most recent effort to establish a unified variability language is CVL. However, given that CVL is relatively new, not much research has yet been conducted to define clear ways of applying its concepts to the RE domain.

In 2008, Moros et al. contributed with the REMM (Requirements Engineering MetaModel) [18], which allows specifying catalogs of reusable requirements models, as well as defining specific product requirements, namely by reusing previously modeled requirements. Furthermore, REMM allows requirements engineers to define, in the same model, both optional and parameterized requirements, which usually are represented in feature models on the side. REMM, like RSL, covers a comprehensive set of RE concepts and relationships between them, and avoids having to keep the consistency between requirements models and variability models, yet it is not in compliance with CVL. The same goes for the research contribution of Alférez et al., in 2010, VML4RE (Variability Modeling Language for Requirements) [1] in what compliance with the CVL is concerned. VML4RE provides for a SPL requirements modeling approach, comprised of variability identification at the feature modeling level, as well as of domain requirements description by means of use cases and activity diagrams, and ultimately a (meta)model to relate modeled features and requirements.

So far CVL has not been specifically applied to RE. The approach proposed in this paper, however, is targeted at specifying the C&V of RE concepts. It achieves this because the RSL allows encoding rigorous SRSs, but also because it is possible to associate variability points in a non-intrusive way for many of the RSL constructs (e.g. goal, quality requirements, actors and use cases).

VII. CONCLUSION

This paper proposes a CVL-based approach for modeling variability aspects in SRS documents, integrated in the ITBox system. This proposal uses the SRS template based on the multi-view architecture defined in the RSL language as its base model. 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) [10].

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 and so, its specifications are rigorous and can be validated and transformed automatically. A lightweight tool support is provided based on an RSL's Excel template publicly available at github.

To provide distributed access to its requirements documents and also data manipulating features, the ITBox web platform makes extensive use of two Google web APIs: *Drive* and *Sheets*. Thanks to this, the ITBox variability modeling approach can automatically extract the information present in a specification document, modify it and generate new documents and views as needed. This allowed defining an automatized process for applying CVL concepts in the context of requirements engineering, which was described in Section III.

Future work will focus on expanding the RSL language supported views, allowing to progress from simply modeling variability at the requirements level (*Goals, Functional and Quality requirements*) to the other view, like the *Stakeholders, Entities, Use Cases* etc., enabling a much wider scope of variability points within the spectrum of requirements engineering concerns. Furthermore, the long-term objective is to fully integrate this process within the RSLingo natural language information extraction approach, expanding the entry point of the variability modeling process from formal RE documents to more unstructured Ad-hoc natural language requirements specifications.

ACKNOWLEDGEMENTS

This work was partially supported by national funds under FCT projects UID/CEC/50021/2013 and CMUP-EPB/TIC/0053/2013.

REFERENCES

[1] M. Alferez, J. Santos, A. Moreira, A. Garcia, U. Kulesza, J. Araújo, and V. Amaral. Multi-view composition language for software product line requirements. Lecture Notes in Computer Science, 5969 LNCS:103–122, 2010.

[2] D. Blanes, J. González-Huerta, and E. Insfran. A multimodel approach for specifying the requirements variability on software product lines. 23rd International Conference on Information Systems Development, ISD 2014, (October 2016):329–336, 2014.

[3] J. Bosch, R. Capilla, and R. Hilliard. Trends in systems and software variability. IEEE Software, 32(3):44–61, 2015.

[4] J. Bosch, S. Deelstra, and M. Sinnema. Systems and Software Variability Management. Systems and Software Variability Management, 2013.

[5] A. Davis. Just enough requirements management: Where Software Development Meets Marketing. 2005.

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

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

[8] D. Ferreira and A. R. Silva. RSL-IL: An interlingua for formally documenting requirements. Proceedings of MoDRE'2012, IEEE CS, 2013.

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

[10] A. R. Silva. Linguistic Patterns and Styles for Requirements Specification: The RSL/Business-Level Language, Proceedings of EuroPLOP'2017, ACM, 2017.

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

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

[13] A. R. Silva, et al., XIS – UML Profile for eXtreme Modeling Interactive Systems, in Proceedings of the 4th International Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2007), March 2007 (Braga, Portugal), IEEE Computer Society, 2007.

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

[15] A. Ribeiro, A. R. Silva. Evaluation of XIS-Mobile, a Domain Specific Language for Mobile Application Development, in Journal of Software Engineering and Applications, 7(11), October 2014, Scientific Research Publishing, 2014.

[16] D. Savić, et al., Use Case Specification Using the SILABREQ Domain Specific Language, in Computing and Informatics Journal, 34(4):877–910, 2015.

[17] OMG. Common Variability Language (CVL) OMG Revised Submission, 2012.

[18] B. Moros, C. Vicente-Chicote, and A. Toval. Metamodeling variability to enable requirements reuse. CEUR Workshop Proceedings, 337:140–154, 2008.

[19] J. Verelst, et al., 2013. Identifying Combinatorial Effects in Requirements Engineering, Proceedings of Third Enterprise Engineering Working Conference (EEWC 2013), Advances in Enterprise Engineering, LNBIP, Springer.

[20] A. R. Silva et al., 2014. Towards a System Requirements Specification Template that Minimizes Combinatorial Effects, Proceedings of QUATIC'2014 Conference, IEEE CS.

[21] J. Coplien, D. Hoffman, and D. Weiss, "Commonality and Variability in Software Engineering," IEEE Software, vol. 15, pp. 37-45, 1998.

[22] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, J. H. Obbink, and K. Pohl, "Variability Issues in Software Product Lines," in 4th International Workshop on Product Family Engineering (PFE-4) Bilbao, Spain: Springer-Verlag, 2002.

[23] D. L. Webber and H. Gomaa, "Modeling Variability in Software Product Lines with the Variation Point Model," Science of Computer Programming, vol. 53, pp. 305-331, 2004.

[24] J. Bayer, S. Gerard, Ø. Haugen, J. Mansell, B. Møller-Pedersen, J. Oldevik, P. Tessier, J.-P. Thibault, and T. Widen, "Consolidated Product Line Variability Modeling," in Software Product Lines - Research Issues in Engineering and Management, T. Käköla and J. C. Duenas, Eds. Berlin Heidelberg: Springer-Verlag, 2006, pp. 195-241.