Requirement's Variability in Model Generation from a Standard Document in Natural Language

Juliana Galvani Greghi*†, Eliane Martins†, Ariadne M. B. R. Carvalho†
*Department of Computer Science
University of Lavras
Lavras-MG Brazil
Email:juliana@dcc.ufla.br
†Institute of Computing
University of Campinas
Campinas-SP Brazil
Email: {eliane, ariadne}@ic.unicamp.br

***Abstract*—Requirement documents are, in general, written in natural language and, therefore, may contain problems such as ambiguities and inconsistencies. In some domains, such as the aerospace domain, requirements are obtained from standard documents that describe the set of services to be implemented. The standard document may present variability, with a set of mandatory requirements, which must be always implemented, and a set of optional ones. Also, different applications may require different sets of services. In general, models, such as Extended Finite State Machines (EFSMs), are used to represent the requirements. Because of the variability found in the document, different models can be generated, making manual modeling a difficult and error-prone task. There are many approaches to (semi) automatic generation of models from requirement documents. But, to the best of our knowledge, none is applied to standard documents, or considers variability issues. In this work, a method for semi-automatic generation of EFSMs from a natural language standard document, with the use of variability modeling, is presented. To validate the proposed approach, a prototype to generate different EFSMs for the same service was developed, considering the commonalities and variabilities of the requirements. The document describes services and protocols for applications in the aerospace domain.**

***Keywords*–requirements modeling; variability management; aerospace domain.**

I. INTRODUCTION

The use of models in Software Engineering is not new, because they are less subject to ambiguities and inconsistencies than natural language descriptions. Besides, models can be automatically processed by tools, enabling not only their validation, but also the derivation of code, as well as test cases. However, modeling is a manual task, error-prone [1] due to human interference. Many applications may show some degree of variability, and the definition of commonalities and variabilities enables reuse of software artifacts, ensuring the quality of the developed applications [2]. Reuse decreases development and testing time, and increases reliability of the reused elements, as much as these artifacts will have been validated in previous applications [3].

Many different models may be required in the presence of variability, catering for mandatory requirements, and for many possible combinations of mandatory and optional requirements. In this context, the following research questions must be answered: Can we deal with the large number of different models that may be generated, considering the combination of mandatory and optional requirements? Can these models be semi-automatically generated?

Some approaches for automatic conversion of texts into models [4][5][6] are detailed in Section III-A. But, to the best of our knowledge, the available approaches do not address variability in the requirements, i.e., when optional requirements make it possible to create different behavioral instances, with the same core asset, which allow the final product to have different capabilities.

In this work, product line engineering concepts were applied to a standard document to deal with the variability of the requirements, and to be able to generate models that represent the services described in the standard document. The analyzed standard document was elaborated by the European Cooperation for Space Standardization (ECSS). The document is the *ECSS-E-70-41A - Ground System and Operations - Telemetry and Telecommand Packet Utilization*, called Packet Utilization Standard (PUS), which standardizes telecommand and telemetry packets related to a set of services for handling on-board data software [7]. Those requirements are applicable to every space mission, and they take into account that different missions require different functionalities [7]. Furthermore, variability modeling of the services described in the PUS document helps the expert user, presenting the features that must be included in the implementation, as well as the optional ones, in a self-contained, summarized, and graphical format. Moreover, the model shows relations between features, suggesting features that should be made available together, and the ones that should not, thus reducing the human effort required to identify the information needed to implement a service.

The objective of this work is to show how to use variability modeling to represent functional requirements from standard documents, and to semi-automatically generate models representing the requirements of each service described in the standard document.

It is important to say that the EFSMs [8] are generated from the standard document, not from the variability model. The variability model is used to help understanding the relationship between the services and their functionalities, which are described in the standard document, and in the development of a product line to semi-automatically generate EFSMs representing the requirements defined in a standard document. The proposed approach was evaluated with a prototype to semi-automatically generate EFSMs from standard documents. Furthermore, an empirical work was conducted in which the PUS document [7] was used. For that, we had the collaboration of researchers from the National Institute for Space Research (INPE), and from the Aeronautics Technology Institute (ITA), who provided the requirements for prototype development, and who informally evaluated the generated models. They are very familiar with the PUS document [7] because it is used in the development of some of their projects.

The rest of the article is organized as follows. Section II introduces the main concepts related to software product line, and variability management. Section III presents related work, and Section IV introduces the proposed approach. Section V shows a working example, and Section VI presents the validation of the proposed feature model. Finally, Section VII concludes the article with suggestions for further work.

II. Background

Software Product Line (SPL) is a set of software systems that share common and optional features. It specifies particular needs, obtained from a set of common assets [9]. Software Product Line Engineering (SPLE) is a paradigm for developing software applications using a common structure and satisfying individual client needs. These individual needs are the variability of the system [2].

There are different definitions for variability. In this work, the following definition is used: "Variability is the ability to change or customize a system" [10]. There are several methods for variability modeling, many of them based on features [11]. The concept of feature modeling was first proposed within the Feature-Oriented Domain Analysis (FODA) method [12]. A feature is an important quality or characteristic of a software system, visible to the user. Feature Model (FM) is used to express common and variable system requirements in a certain domain, and the relationship between them [12]. FM is used for product line development, and it represents the properties of the system that are relevant to the end users in a hierarchical form [11].

Feature modeling was chosen because it can be easily understood by stakeholders, and it is widely accepted by software engineers [11]. The notation used, presented in [13] *apud* [14], extends the feature model notation with the feature type *or-feature*, improving the representation of the relationship between features.

III. Related Work

There are many approaches for model generation from natural language documents. The most relevants to this work are presented in Subsection III-A. The application of SPL in the space domain has been encouraged along the years, and some examples of these applications are presented in Subsection III-B.

A. Model Generation from Natural Language Information

The transformation of natural language requirements into models present many challenges. Sometimes documents have to be rewritten, either to correct mistakes, or to express information in a structured format. Moreover, the notation in which the information is expressed, and the kind of model that must be generated, should be taken into account.

The approaches presented here use some sort of Natural Language Processing (NLP) technique for text pre-processing, usually a syntactic analyzer (or parser), and a part-of-speech (PoS) tagger. The parser provides a valid syntactic structure for a sentence. The process is supported by a grammar that defines a set of valid structures in a given domain [15][16]. The PoS tagger associates syntactic and morphological features to the words of a sentence or text [15].

Yue, Briand and Labiche [17][4] present a method, implemented as part of the aToucan tool, which takes use cases in a pre-defined format [18] as input, extracting information and producing Unified Modeling Language (UML) models (class, activity, and state diagrams). Information extraction is performed by NLP tools and transformation algorithms. An extension of this work is presented in [19] to generate Aspect-based State Machines. A product line model and configuration were proposed in [20] to support model-based tests.

Deeptimahanti and Babar [21] transform the requirements document into UML models (class, use cases, and collaboration diagrams), using a tool called UMGAR. The process initiates with requirements being rewritten to remove eventual ambiguities from the sentences. Then, the information is extracted to generate the analysis model. The model can be converted to Extensible Markup Language Metadata Interchange (XMI) files, allowing for requirements traceability [5].

Kof [6] presents an approach to convert textual descriptions of an automaton into a finite automaton model. The text is pre-processed using a PoS tagger. The identification of states relies on search terms, defined by the user, and on specific word categories. The identification of transitions is based on sentence's segmentation, and on the categorization of the sentence segments. A set of rules was defined to extract the text segments that should be used for model generation. The generated automaton is represented in a tabular form.

Kof and Penzenstadler [22] present a method for integration of an NLP approach and a CASE tool. They show a methodology for translation from text to model using a previous work [6]. The training data requires that the user selects the section of the document to be processed. The methodology was integrated with a tool that enables user interaction. The tool learns on the fly about words and grammar constructions that can be used in model generation.

Santiago [23] presents a methodology, SOLIMVA, to generate model-based test cases from requirements in natural language documents. A statechart is produced with the information extracted from the requirements and a tool, called GTSC [24], is used to generate test cases.

In our approach, the input is not restricted to a specific format, as in [4][17], and sentences are not rewritten, differently from [21][5]. The search for patterns is similar to [6], but in addition, our approach uses patterns extracted from the document's structure to obtain information that is used for model generation. Similarly to Santiago's work [23], our approach was applied to the aerospace domain. All of them use NLP tools to extract information from a natural language document.

The novelty of the proposed approach is variability handling: to the best of our knowledge, none of the existing approaches deals with requirements variability during model generation. In our approach, commonalities and variabilities are identified and modeled with an FM, and this information is used to guide model generation.

The information extraction process is briefly described in Section IV.

B. Product Line for Space Application

The use of SPL approaches in space applications has increased along the years due to many factors, e.g., search for more rigorous architectural definition, cost saving, and the need for more systematic approaches for systems development [25][26][27][28].

Many approaches apply product line concepts in the aerospace domain and, in general, the final products are specific software applications [28][29]. Our approach, differently, applies product line concepts for model generation.

IV. Representing variability for EFSM model generation

The goal of this work was to use variability modeling to help the semi-automatic generation of EFSMs. Variabilities were identified to facilitate the generation, from standard documents, of models used for development and testing. To exemplify the entire process, a sample text, with some characteristics often found in standard documents, is presented in Figure 1.

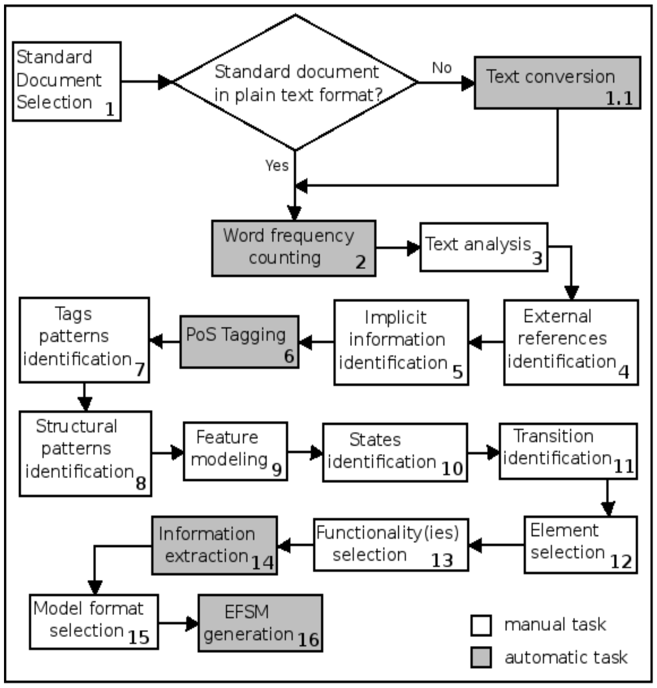
This is a made up data packet validation protocol, illustrating the proposed information extraction methodology.
Standard document for network communication protocol
Chapter 3 - Data packet validation
3-A Description
This chapter defines a protocol for data packet validation. Data is organized in packets, as described in chapter 1, and these packets must be validated before sending. When a packet is sent, a validation activity is initiated. The mandatory fields must be filled always, on each data transfer. Optional fields can be filled according to the application requirements.
3-B Concepts
Data packet validation occurs in two stages:
1) Verification of mandatory fields: the type and the size of data must be verified. Data must conform to the allowed types, and the size must comply with the restrictions of the used type. An error shall occur if any mandatory field is not filled.
2) Verification of optional fields: if one (or more) optional field is filled, the data value must be verified. Data must conform to the allowed values for the field. If a field value is out of range, a failure report must be generated.
3-C Available functionalities
3-C.1 Basic functionalities
Verification of mandatory fields (1,1): the result of the verification must be recorded in a report file.
Verification of optional fields (2,1): the result of the verification can be recorded in a report file if it is required by the application.
In addition to the description, other details about the protocol are described in the last chapters of this manual. For details consult Appendix A.
3-C.2 Additional functionalities
Sending a warning e-mail (3,2)
Registering a log file for validation errors (4,2)
Validation of data packet integrity (5,2)
3-D Dependencies

ID	Name	Requires
1	Verification of mandatory fields	
2	Verification of optional fields	
3	Sending a warning e-mail	
4	Registering a log file for validation errors	
5	Validation of data packet integrity	1,2

Figure 1. Text fragment to illustrate the proposed methodology.

A. EFSM model generation

Figure 2 shows the process diagram for State Machine Generation. The entire process has seven steps, subdivided in sixteen tasks, which are described next. The process of information identification and information extraction was already discussed in a previous work [30].



Standard Document Selection 1
Standard document in plain text format?
No
Text conversion 1.1
Yes
Word frequency counting 2
Text analysis 3
External references identification 4
Implicit information identification 5
PoS Tagging 6
Tags patterns identification 7
Structural patterns identification 8
Feature modeling 9
States identification 10
Transition identification 11
Element selection 12
Functionality(ies) selection 13
Information extraction 14
Model format selection 15
EFSM generation 16
manual task
automatic task

Figure 2. Process Diagram for State Machine Generation.

Step 1 - Semi-automated analysis of the standard document

This step comprises a set of activities performed by an expert analyst. The process initiates with the selection of the standard document to be processed (Task 1 - Figure 2). The selected document must be in plain text format (.txt) to be processed by the NLP tools. If the text is available in a different format, it must be converted to .txt (Step 1.1 - Figure 2). The plain text standard document is then processed by a word frequency counter tool, and the results are used in the analysis of the text (Tasks 2 and 3 - Figure 2). This analysis helps to identify structural patterns in the sentences. Stop words, i.e., words that have a high frequency but no semantic meaning [31], are not taken into account in the analysis. An example of an identified pattern is: most sentences initiating with the word "if" present a condition for the execution of an action, as it can be noticed in the sentence "If a field value is out of range, a failure report must be generated.", presented in Section 3-B.2, Figure 1.

Step 2 - Reading the text

After finding patterns in the sentences, the next task is the careful manual reading of the standard document to identify references to other documents, or to a different section or chapter of the same document (Task 4 - Figure 2). This information is marked for future processing.

The next task is to identify implicit information, i.e, information represented in tables or figures that are not explicit in the text, and that are related to the described service (Task 5 - Figure 2). This information must also be marked and processed to be used in a later stage. In the proposed methodology, information described in different chapters/sections, or as figures or tables, is manually extracted and stored in a database for EFSM generation. This fact can be observed in Section 3-D, line 5, Figure 1: the dependency relation is not explicit in the text, and must be manually extracted and stored in a database to be used during model generation.

Step 3 - Reading the service description

Text is pre-processed using the Stanford PoS Tagger [32] (Task 6 - Figure 2). Each category attributed to a word is a tag that can be used to help the expert analyst in the extraction process. Sequences of tags were already used by Kof [6], and his research is the work mostly related to ours. In Figure 3, we can observe sequences of tags. In the underlined text segments, the sequence "noun (tags initiated by NN) preposition (tag IN) adjectives (tag JJ) noun (tags initiated by NN) noun (tags initiated by NN)" is recurrent and can be used to identify sets of information that must be extracted and used in later stages. The process of finding patterns in the sequences of tags is represented, in Figure 2, by Task 7.

In addition to the categories, the structural pattern identified in the document can also be used during information extraction. The identification of structural patterns is represented, in Figure 2, by Task 8. An example of a structural pattern can be found in Section 3-C.1, Figure 1. There is a description of a functionality, followed by a parenthesis, two numbers separated by a comma, and another parenthesis. The first number is an identification of the functionality, and the second indicates if it is a mandatory functionality (number 1), or an optional functionality (number 2). The pattern "description parenthesis number comma number parenthesis" can be used in the extraction process to help the identification of information. After the analysis, the expert analyst must identify mandatory and optional functionalities described in the text. These functionalities must be represented in a feature model, which is used to guide the selection of services and functionalities in a later stage of the processing (Task 9 - Figure 2).

Step 4 - Identification of states and transitions

The process continues with the identification of states (Task 10 - Figure 2), which relies on the location of the information, e.g., on the section of the document where the information is described, and on the sequence of the tags. In Figure 1, we can see that states are located in the "Concepts" section, identified by the number of the chapter, and by the identification of the section (3-B). We can also observe that the description of the stages starts with a noun (tags initiated with NN), followed by "of" plus and adjective, and finished with a noun. This pattern may be identified in Figure 3, in the underlined text segments. The tasks to identify transitions are very similar to those for state identification, and represented, in Figure 2, by Task 11. A transition is composed of events, actions and conditions, also known as guards. The identification of each transition component can follow a set of rules and patterns.

The identification of conditions and actions, for example, is based on the keyword "if". In Figure 1, we can observe two different situations: the if-clause at the beginning of the sentence (Section 3-B.2, Figure 1 - "If a field value is out of range, a failure report must be generated."), and the if-clause at the end of the sentence (Section 3-B.1, Figure 1 - " An error shall occur if any mandatory field is not filled."). In the first situation, the text segment between the conjunction and the comma is a condition ("a field value is out of range"), and the text segment after the comma is an action ("failure report must be generated"). In the second situation, the action is the text segment until the conjunction ("An error shall occur") and the condition is the text segment after the conjunction ("any mandatory field is not filled."). Patterns are used to create sets of rules to extract information from the standard document. The set of rules must cover the entire description of a service present in the standard document. It is possible that each service may need a specific set of rules, because the methodology is highly dependent on the text structure and on the writing style.

a_DT -RRB-_-RRB-Verification_NNP of_IN mandatory_JJ fields_NNS :_: the_DT type_NN and_CC the_DT size_NN of_IN data/NNS (...)
b_DT -RRB-_-RRB- Verification_NNP of_IN optional_JJ fields_NNS :_: if_IN one_CD -LRB--LRB- or_CC more_JJR -RRB--RRB-optional_VBD (...)

Figure 3. Text segments for tag pattern identification

Step 5 - Selection of the service and functionalities to be modeled

This activity must be performed by an expert user, who is using the requirements described in the standard document for the development of a new system. The user must select the service that needs to be modeled (Task 12 - Figure 2). The mandatory functionalities are selected by default, but the expert user may select the optional functionalities that must be present in the model (Task 13 - Figure 2). The information about what is mandatory or not in a service is provided by the feature models elaborated in Task 9, Figure 2.

Step 6 - Information extraction

The information extraction is an automated task (Task 14 - Figure 2), which uses the sets of rules created during Step 4. The rules are applied in the standard document to extract information about a specific service, selected by the expert user in Task 12, Figure 2. The entire information about the selected service is recovered. After the extraction process is completed, the necessary information to represent only the functionalities selected in Task 13, Figure 2, is filtered and sent to the next step, that is, generation of EFSM. The process of information filtering is based on mandatory and optional features identified through the feature models elaborated in Task 9, Figure 2.

Step 7 - Generation of the EFSM

In the last step of the process, the expert user must select the model format (Task 15 - Figure 2). The text format is generated by default, but the user may request the generation in graphical format too. The last activity is the generation of the model (Task 16 - Figure 2). The extracted information filtered according to the selection made by the expert user is used to generate the EFSM.

B. Variability *Modeling*

As mentioned in Section II, variability is expressed through a Feature Model (FM), following the FODA method [12]. The three steps for feature modeling are described next.

Feature Identification

A feature is an aspect or characteristic of the system considered important by its users [12]. In our proposal, features are defined according to the expert user's viewpoint.

The user may select the standard document version, the text converter and the PoS tagger. The text converter is used to pre-process the standard document, and the PoS tagger is used for annotation. These requirements are considered features because they are directly related to the choices of the expert user that can affect the configuration of the new model.

Regarding the state machine model, the user must select the service(s) to be modeled, the information about the EFSM that will guide model generation, and the output notation of the model. These are the main mandatory features of the system. The sub-features identified for each of these main features are as follows.

Feature modeling

According to the FODA method [12], feature commonalities and variabilities are represented as a tree. Figure 4 presents the feature diagram for the translation system *txt2smm*.

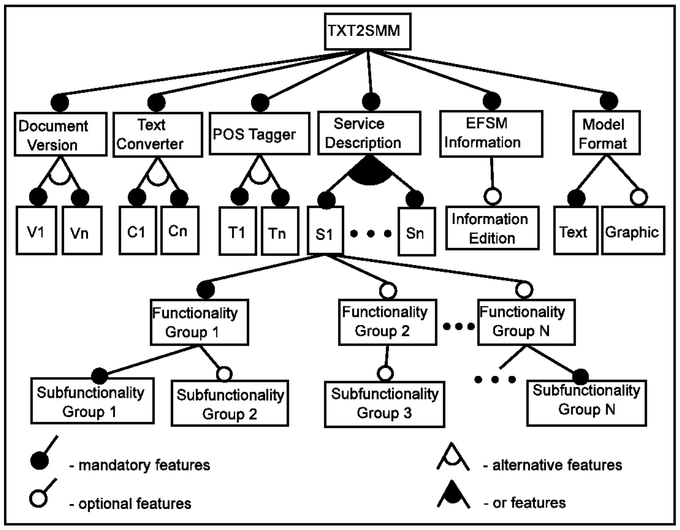


Figure 4. Feature diagram for the txt2smm system.

The generation of a new model requires the configuration of six main mandatory features. Three of them are related to document management: "Document Version", "Text Converter" and "PoS Tagger", each with alternative sub-features. In an alternative relationship, only one sub-feature must be selected.

"Document Version" presents the sub-features "V1" and "Vn", representing that the user must select the version of the standard document. "Text Converter" presents the sub-features "C1" and "Cn", representing that the user must select the converter tool. Finally "PoS Tagger" presents the sub-features "T1" and "Tn", representing the user must select the PoS Tagger to be used for text annotation.

"Service Description", "EFSM Information" and "Model Format" are the next three features, related to the state machine generation. "Service Description" presents the *or-features* "S1" and "Sn", which means that at least one service must be selected. Each service has a set of mandatory functionalities, always available in every implementation of the service, and which are represented by the mandatory feature "Functionality Group 1". Additional functionalities are represented by optional features "Functionality Group N". Each Functionality may describe a set of sub-functionalities such as reports, messages, or requests.

These are represented by "Sub-functionality 1" to "Sub-functionality N", which may be mandatory, optional, alternative, or or-feature, depending on the requirements of each functionality.

The information extracted for state machine generation is presented to the expert user who may edit it. This is represented by the mandatory feature "EFSM Information", and by the optional feature "Information Edition". The state machine model can be generated in textual or graphical notation. This selection is represented by the mandatory feature "Model Format", and the available formats are represented by the mandatory feature "Text", and by the optional feature "Graphic".

Applying feature modeling to the text example in Figure 1, and considering that "Data packet validation protocol" is one of the protocols described in the standard document, the resultant FM would look like the diagram shown in Figure 5, where "Data transfer protocol" is represented by "S1".

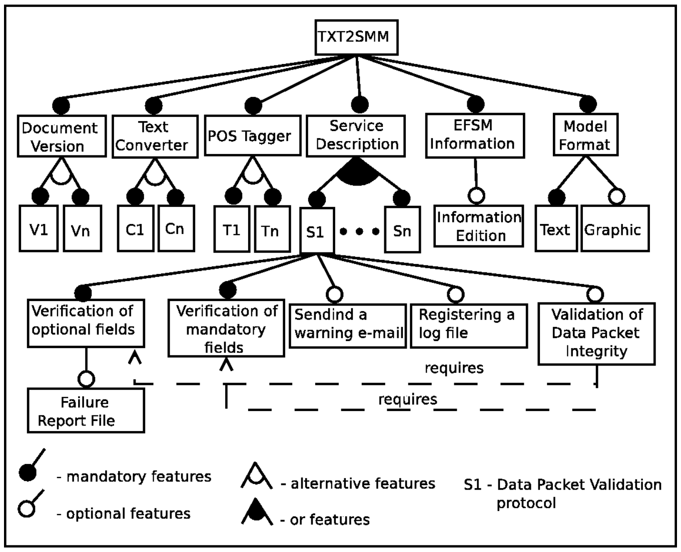


Figure 5. Feature diagram for the text example.

The service described in the text example is a made up data packet validation protocol, and the available functionalities are: Verification of mandatory fields, Verification of optional fields, Sending a warning e-mail, Registering a log file for validation errors, and Validation of data packet integrity. The first two functionalities are mandatory, i.e., they must be always available in every implementation of the protocol, and the last three functionalities are optional. The text also describes the generation of a report file for "Verification of optional fields", which is mandatory. The dependency relation is described in Section 3-D, Figure 1.

Feature model validation and analysis

There are many tools for supporting feature modeling, but some are more adequate for feature checking; to validate and analyze the generated model, one, to be presented later, was chosen. Several operations for automated analysis of feature models are available, like: validation of the feature model, i.e., whether the FM allows for the configuration of at least one valid product; estimation of the number of valid products; computation of all valid products; evaluation of the variability degree of the feature model; checking the consistency of the product, i.e., determining if a specific combination of features is valid, looking for errors in the feature model, and many other operations. The feature model validation process is presented in Section VI.

V. Working Example

The proposed approach was applied to the PUS document [7]. A prototype was developed to extract the necessary information, and to generate different models for the same service, taking into account the commonalities and variabilities of the service. The text was pre-processed using the Stanford PoS Tagger [32].

The PUS document describes a set of 16 services to support functionalities for spacecraft monitoring, but there are no mandatory services for a given mission. Each service is described with a minimum capability set that must be always included in every implementation of the service, and additional capability sets that may be optionally implemented [7]. In addition to the variability, the document also defines other important requirements that must be considered in the implementation of each service, and which may appear in different sections of the PUS document.

An example of an instantiation of the feature model for the generation of a state machine of the Telecommand Verification Service (TVS) is shown. This example was executed using the prototype especially developed for the task.

The TVS is described in four stages, each one with a set of capabilities that might be mandatory or optional. Each capability defines the generation of reports describing the failure or the success of a given stage. According to the chapter that describes the information that can be used for any service, failure reports must always be generated and, according to the description of the service, there will be a dependency relationship between success and failure reports [7].

Regarding document processing, the following features were selected: standard document version: V1, corresponding to the version from 30 January, 2003; Text converter: C1; POS Tagger: T1; Service: TVS; Minimum Capability: Capability Group 1, corresponding to Acceptance of the telecommand; Additional Capabilities: Capability Group 4, corresponding to the Completion of the Execution of the telecommand; Edit Information: No edition; Model Format: text.

With this configuration, an EFSM in text format is generated, and shown here in a tabular form (Table I). The first column shows the state names, extracted from the PUS document. The state "withoutTC" is not presented in the text, but it was included in the EFSM as recommended by the researchers from INPE and ITA who informally evaluated the generated models.

The second and third columns are names used to identify input and output events for each state. These names were semi-automatically extracted from the document during the information extraction phase, presented in Figure 2. This simple example shows the state machines generated to represent the normal behavior of the Telecommand Verification Service. The resultant model can be seen in Figure 6.

The model presented in Figure 6 represents a semi-automatically generated EFSM for the example presented in Table I. It was redrawn, and the natural language conditions were suppressed due to the to lack of space. In this model, the service is represented in two main states: the acceptance of the telecommand, and the completion of the execution of the telecommand. For each of these states, there is an available report. In Figure 6, this report is represented by a natural language parameter, followed by the terms AckON or AckOFF. These terms refer to the possibility of sending or not the report to the control station.

To evaluate the generated model, the model presented in Figure 6 was compared to the EFSM manually generated (Figure 7), available at [33].

These models represent the same capabilities configured in the example (Table I), and some differences between the models were found: an additional state present in Figure 7, which is represented in the semi-automatically generated EFSM as a natural language condition; the information about sending reports to the control station that is represented, in Figure 7, by brackets and hifens. Despite of these differences, the main information were preserved, and the semi-automatically generated model can be used by the expert user as a initial model for the development and testing processes.

This working example helps to answer the first research question presented in Section I: Can we manage the large number of different models that can be generated, considering the combination of mandatory and optional requirements? The product line approach helps to deal with the different models which may be generated for the same service, considering the possible combinations between mandatory and optional features. The implementation of the prototype and the evaluation of the generated models helps to answer the last research question presented in Section I: the models can be semi-automatically generated, and the possible combinations are generated in a controlled way, avoiding invalid feature combination.

TABLE I. OUTPUT FROM THE STATE MACHINE GENERATION SYSTEM

State	Input Event	Output Event
without TC	TC	TC
Acceptance of the telecommand	TC	Acceptance OK
Completion of execution	Acceptance OK	Completion OK

VI. Feature Model Validation

The feature model presented in Figure 4 was validated within the FaMa FW [34] tool, which represents variability via feature models. FaMa FW performs many analysis operations on an FM, for example, checking if the FM has at least one valid product, computing all valid products for a given FM, and the variability degree of the FM.

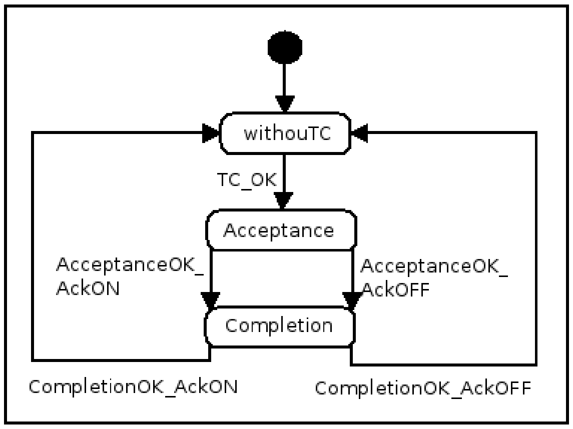


Figure 6. EFSM semi-automatically generated.

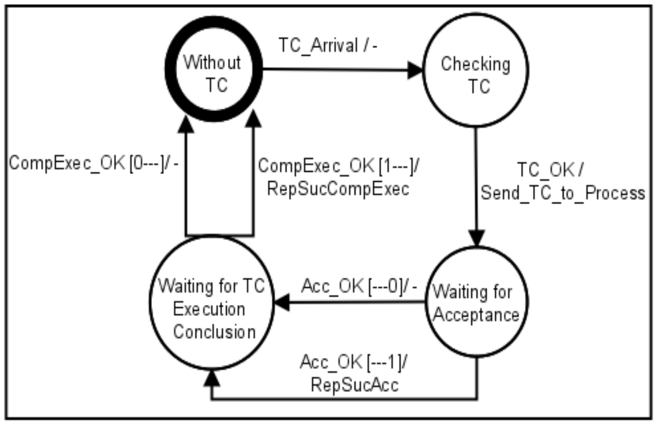


Figure 7. EFSM manually generated. Adapted from [33]

FaMa FW can be used in a shell front-end, or integrated to other applications. In this work, the shell front-end was used. All available operations in the command line interface were applied, and no errors were found.

For instance, the TVS affords two document versions, two converter tools, two PoS taggers, the different combinations of capabilities from TVS, the possibility of editing information, and two model notations to represent the EFSM. Choices between the several alternatives multiply to give 256 descriptions of new products. The number gets even larger if it is taken into account that the PUS document describes 16 services.

The prototype avoids the selection of invalid combinations, for example, automatically selecting mandatory features from a selected service by the expert user, or automatically selecting/deselecting features related to an optional, or alternative feature selected by the expert user.

Regarding the selection used in the working example for the Document Version, the PoS tagger, the text converter and the selected service, there are 8 different possible combinations of additional capabilities for the TVS.

Some limitations were identified in the proposed approach: self-loops and hierarchies cannot be treated unless explicitly described in the text.

VII. Conclusion and Future Work

The use of models to represent requirements is widespread as an alternative to minimize problems during development and testing processes of computational systems. In an attempt to help the semi-automatic generation of EFSMs, considering the variability of requirements in a natural language standard document, the following questions were posed in Section I: Can we deal with the large number of different models that may be generated considering the combination of mandatory and optional requirements? Can these models be semi-automatically generated?

These questions were answered in the article, showing that variability modeling is fundamental to the successful generation of state machine models, not only by allowing for the exploration of numerous combinations, but also for avoiding the invalid ones.

A prototype tool was implemented to semi-automatically generate state machine models. The models were generated according to the proposed feature model, and informally evaluated by the INPE and ITA researchers, who also selected the set of services to be modeled. The state machine models generated with the prototype tool were validated using a model checking tool, and by comparing the semi-automatically and manually generated models for the same services, with the same configuration.

Results showed that the state machines are very similar. The similarities are a strong indication that the generated models could help the analysts in the development and testing processes of new computational systems, reducing time and effort needed for manual modeling.

Although the effort needed to extract information from the document may seem excessive due to the writing style dependency of the standard document, this can be counterbalanced by several aspects: reduction of time consumed in manual modeling, minimization of errors due to human interference, and possibility of automating the testing process. Moreover, a standard document is used by many organizations around the world, and these benefits can be extended to all of them. Another positive trait of the approach is that the use of structural information of the text during information extraction process eliminates the need for rewriting sentences, as required by some available approaches.

Some limitations to the semi-automatic generation are the treatment of self-loops and of the hierarchical relationships, already identified in the literature. In the proposed approach, self-loops could be treated if they were explicitly described in the text.

As future work, we intend to generate models for other services described in the PUS document, and evaluate the prototype from the user perspective, aiming at identifying usability issues and other problems that might be fixed in the final version of the tool.

Acknowledgment

The authors would like to thank Ana Maria Ambrósio, from the National Institute for Space Research, for providing requirements for service implementation, Emília Villani, from the Aeronautics Technology Institute, Cecília M. F. Rubira, from the University of Campinas, Patrick Henrique Brito, from the Federal University of Alagoas, Marco Vieira, from Coimbra University and Raphael Winckler de Bettio, from the University of Lavras, for their valuable help throughout this research. The authors would also like to thank Fundação de Amparo à Pesquisa do Estado de Minas Gerais - Fapemig,- Fundação de Apoio ao Ensino, à Pesquisa e à Extensão - FAEPEX/Unicamp and Institute of Computing - Unicamp, for providing financial support.

References

[1] V. Ambriola and V. Gervasi, "Processing natural language requirements," in In Proceedings of ASE 1997. IEEE Press, 1997, pp. 36–45.

[2] K. Pohl, G. Böckle, and F. J. van der Linden, Software Product Line Engineering: Foundations, Principles and Techniques. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[3] R. Pressman, Engenharia de software. McGraw-Hill, 2006.

[4] T. Yue, L. Briand, and Y. Labiche, "Automatically deriving a UML analysis model from a use case model," Simula Research Laboratory, Tech. Rep. 2010-15 (Version 2), 2013.

[5] D. K. Deeptimahanti and R. Sanyal, "Semi-automatic generation of UML models from natural language requirements," in Proceedings of the ISEC '11. New York, NY, USA: ACM, 2011, pp. 165–174, retrieved: 2015.09.17. [Online]. Available: http://doi.acm.org/10.1145/1953355.1953378

[6] L. Kof, "Translation of textual specifications to automata by means of discourse context modeling," in Requirements Engineering: Foundation for Software Quality, ser. LNCS, M. Glinz and P. Heymans, Eds. Springer Berlin Heidelberg, 2009, vol. 5512, pp. 197–211.

[7] ECSS, Ground systems and operations - Telemetry and telecommand packet utilization. ECSS-E-70-41A. ESA Publications Division, 2003.

[8] V. S. Alagar and K. Periyasamy, Specification of Software Systems. Springer London Dordrecht Heidelberg New York, 2011.

[9] L. M. Northrop, "SEI's software product line tenets," IEEE Softw., vol. 19, no. 4, Jul. 2002, pp. 32–40, retrieved: 2015.09.17. [Online]. Available: http://dx.doi.org/10.1109/MS.2002.1020285

[10] J. van Gurp, J. Bosch, and M. Svahnberg, "On the notion of variability in software product lines," in Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on, 2001, pp. 45–54.

[11] K. Kang and H. Lee, "Variability modeling," in Systems and Software Variability Management, R. Capilla, J. Bosch, and K.-C. Kang, Eds. Springer Berlin Heidelberg, 2013, pp. 25–42.

[12] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11231, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, Tech. Rep. CMU/SEI-90-TR-021, 1990, retrieved: 2015.09.17.

[13] K. Czarnecki, "Generative programming: Principles and techniques of software engineering based on automated configuration and fragment-based component models," Ph.D. dissertation, Technical University of Ilmenau, October 1998.

[14] K. Czarnecki, S. Helsen, and E. Ulrich, “Staged configuration through specialization and multilevel configuration of feature models,” Software Process: Improvement and Practice, vol. 10, 04/2005 2005, pp. 143 – 169.

[15] E. Charniak, “Statistical techniques for natural language parsing,” AI Magazine, vol. 18, no. 4, 1997, pp. 33–44.

[16] D. Klein and C. D. Manning, “Accurate unlexicalized parsing,” in Meeting of the Association for Computational Linguistics, 2003, pp. 423–430.

[17] T. Yue, L. Briand, and Y. Labiche, “An automated approach to transform use cases into activity diagrams,” in Modelling Foundations and Applications, ser. LNCS, T. Kühne, B. Selic, M.-P. Gervais, and F. Terrier, Eds. Springer Berlin Heidelberg, 2010, vol. 6138, pp. 337–353, retrieved: 2015.09.17. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13595-8_26

[18] ——, “A use case modeling approach to facilitate the transition towards analysis models: Concepts and empirical evaluation,” in Model Driven Engineering Languages and Systems, ser. Lecture Notes in Computer Science, A. Schürr and B. Selic, Eds. Springer Berlin Heidelberg, 2009, vol. 5795, pp. 484–498.

[19] T. Yue and S. Ali, “Bridging the gap between requirements and aspect state machines to support non-functional testing: Industrial case studies,” in Modelling Foundations and Applications, ser. Lecture Notes in Computer Science, A. Vallecillo, J.-P. Tolvanen, E. Kindler, H. Störrle, and D. Kolovos, Eds. Springer Berlin Heidelberg, 2012, vol. 7349, pp. 133–145, retrieved: 2015.09.17. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31491-9_12

[20] S. Ali, T. Yue, L. Briand, and S. Walawege, “A product line modeling and configuration methodology to support model-based testing: An industrial case study,” in Model Driven Engineering Languages and Systems, ser. Lecture Notes in Computer Science, R. France, J. Kazmeier, R. Breu, and C. Atkinson, Eds. Springer Berlin Heidelberg, 2012, vol. 7590, pp. 726–742, retrieved: 2015.09.17. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33666-9_46

[21] D. K. Deeptimahanti and M. A. Babar, “An automated tool for generating UML models from natural language requirements,” in Proceedings of ASE’09, Nov 2009, pp. 680–682.

[22] L. Kof and B. Penzenstadler, “Faster from requirements documents to system models: Interactive semi-automatic translation,” in Proceedings of the REFSQ 2011 Workshops REEW, EPICAL and RePriCo, the REFSQ 2011 Empirical Track (Empirical Live Experiment and Empirical Research Fair), and the REFSQ 2011 Doctoral Symposium, B. Berenbach, M. Daneva, J. Dör, S. Fricker, V. Gervasi, M. Glinz, A. Herrmann, B. Krams, N. H. Madhavji, B. Paech, S. Schockert, and N. Seyff, Eds. ICB Research Report- 44, 2011, pp. 14–25.

[23] V. A. d. Santiago Júnior and N. Vijaykumar, “Generating model-based test cases from natural language requirements for space application software,” Software Quality Journal, vol. 20, no. 1, 2012, pp. 77–143.

[24] V. A. d. Santiago Junior, N. Vijaykumar, D. Guimaraes, A. Amaral, and E. Ferreira, “An environment for automated test case generation from statechart-based and finite state machine-based behavioral models,” in Proceedings of ICSTW ’08, April 2008, pp. 63–72.

[25] R. Lutz, “Software engineering for space exploration,” Computer, IEEE Computer Society, vol. 44, no. 10, 2011, pp. 41–46, retrieved: 2015.09.17.

[26] J. Penã, M. G. Hinchey, A. Ruiz-Cortés, and P. Trinidad, “Building the core architecture of a nasa multiagent system product line,” in AOSE, ser. LNCS, L. Padgham and F. Zambonelli, Eds. Springer Berlin Heidelberg, 2007, vol. 4405, pp. 208–224.

[27] K. Weiss, “Reviewing aerospace proposals with respect to software architecture,” in Aerospace Conference, 2007 IEEE, March 2007, pp. 1–20.

[28] J. Fant, H. Gomaa, and R. Pettit, “Software product line engineering of space flight software,” in Product Line Approaches in Software Engineering (PLEASE), 2012 3rd International Workshop on, June 2012, pp. 41–44.

[29] I. Habli, T. Kelly, and I. Hopkins, “Challenges of establishing a software product line for an aerospace engine monitoring system,” in Software Product Line Conference, 2007. SPLC 2007. 11th International, Sept 2007, pp. 193–202.

[30] J. G. Greghi, E. Martins, and A. M. B. R. Carvalho, “Semi-automatic generation of extended finite state machines from natural language standard documents,” in Dependable Systems and Networks Workshops (DSN-W), 2015 IEEE International Conference on, June 2015, pp. 45–50.

[31] C. D. Manning and H. Schütze, Foundations of statistical natural language processing. MIT press, 1999.

[32] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, “Feature-rich part-of-speech tagging with a cyclic dependency network,” in Proceedings of the NAACL ’03 - Volume 1. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 173–180, retrieved: 2015.09.17. [Online]. Available: http://dx.doi.org/10.3115/1073445.1073478

[33] R. P. Pontes, P. C. Véras, A. M. Ambrosio, and E. Villani, “Contributions of model checking and cofi methodology to the development of space embedded software,” Empirical Software Engineering, vol. 19, no. 1, 2014, pp. 39–68, retrieved: 2015.09.18. [Online]. Available: http://dx.doi.org/10.1007/s10664-012-9215-y

[34] Research Group of Applied Software Engineering, “FAMA-FeAture Model Analyser,” http://www.isa.us.es/fama/, retrieved: 2015.09.17.