UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÉNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

BPM Text Model: Automatic Synchronization of BPM Description Artifacts

Raphael de Almeida Rodrigues

Orientadores

Leonardo Guerreiro Azevedo

Kate Cerqueira Revoredo

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO de 2016

BPM Text Model: Automatic Synchronization of BPM Description Artifacts

Raphael de Almeida Rodrigues

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

_____

Leonardo Guerreiro Azevedo, D.Sc. - UNIRIO

_____

Kate Cerqueira Revoredo, D.Sc. - UNIRIO

_____

Fernanda Araújo Baião Amorim, D.Sc. - UNIRIO

_____

Marcelo Fantinato, D.Sc. - USP

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO de 2016

Rodrigues, Raphael de Almeida
R696 BPM Text Model: Automatic Synchronization of BPM Description Artifacts / Raphael de Almeida Rodrigues, 2016.
186f. ; 30 cm + 1 CD-ROM

Orientador: Leonardo Guerreiro Azevedo
Coorientadora: Kate Cerqueira Revoredo
Dissertação (Mestrado em Informática) - Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2016.

1. Processamento de linguagem natural (Computação). 2. Framework (Arquivo de computador). 3. Java (Linguagem de programação de computador). I. Azevedo, Leonardo Guerreiro. II. Revoredo, Kate Cerqueira. III. Universidade Federal do Estado do Rio de Janeiro. Centro de Ciências Exatas e Tecnológicas. Curso de Mestrado em Informática. IV. Título.

CDD - 006.35

For my family, that thought me nothing is impossible.
For my grandmother, that spiritually guided me through this journey.
For my sweet wife Bela, whose love made me a better person.

Acknowledgement

I would like to thank the teachers and staff of the PPGI-UNIRIO, by work to make a great graduate program, especially at a time with ever smaller investments for the development of Brazilian scientific research. In the midst of many difficulties, they had made the best, giving all support for me and my colleagues of PPGI-UNIRIO.

My special thanks go to Kate Revoredo and Leonardo Azevedo, for proof reading my thesis and for their valuable corrections and recommendations.

I want to thank my family for their support and understanding during times of high work load. thank you for your love and warmth, encouragement, patience, and for believing in me.

Thanks for my friend and guide, Leonardo Azevedo, for its dedication, patience and skills which were essential for the development of this thesis, and for its major contribution in my academic and professional career. Thanks for guiding me through my bachelor studies and staying by my side through my master course. Thanks for giving me the set of skills needed to finish this journey.

I would like to thanks my grandmother, that always tried to calm me down and renew my strengths. Thanks for showing me the right path when I could not see. I know that you are watching me from above and I hope that you will be proud to see how your grandson has grown. I love you, and this victory is for you!

I would like to place a special thanks for Isabela Medeiros, who in the beginning of this Master course was my girlfriend and now has become my wife and life-time partner. Thanks for understanding my lack of time, for helping me to overcome several challenges and accepting the challenge to live a new life in São Paulo by my side.

Finally, I would like to thanks Signavio initiative whose support was essential during this whole research.

Rodrigues, Raphael de Almeida. **BPM Text Model: Automatic Synchronization of BPM Description Artifacts**. UNIRIO, 2016. 186 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

ABSTRACT

The proper representation of a Business process is important for its execution and understanding. BPMN has been used as the standard notation for business process models, however domain specialists, which are experts in the business, do not have necessarily the modeling skills to easily read a business process model. It is easier for them to read in natural language. For this reason, both model and text are necessary artifacts for a good communication between business specialists and system analysts. The manual management of both resources may result in inconsistencies, due to unilateral modifications. This research propose a methodology for text generation from process models and vice-versa, while also applying changes made to the text back to the original process model. The technique also enables domain experts to edit formal process models without the efforts of learning a modeling language. The methodology was validated through a language-independent framework, instantiated using Java standard technology. The methodology was evaluated through case studies and experiments. Three main conclusions could be drown from the evaluation. First, textual work instructions can be considered equivalent to the process models in terms of knowledge representation within an acceptable threshold (*74% of the subjects claim the equivalence between both knowledge representations vary from 68% to 100%*). Second, the chosen textual format is good (*86% of the subjects claim the textual descriptions vary from excellent to good*). Third, the knowledge represented by the manually updated text can be considered equivalent to the automatically updated process model after the synchronization within an acceptable threshold (*78% of the subjects claim the knowledge represented by the manually updated text is equivalent to the automatically updated process model*).

Keywords: BPM; BPMN; Natural Language Generation; Natural Language Processing.

Rodrigues, Raphael de Almeida. **BPM Text Model: Automatic Synchronization of BPM Description Artifacts**. UNIRIO, 2016. 186 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO.

RESUMO

A representação adequada de um Processo de Negócio é importante para seu entendimento e execução. BPMN foi utilizado como a notação padrão para modelos de processo, porém especialsitas de domínio, que são os especialsitas do negócio, não detêm as habilidades de modelagem necessárias para facilmente ler um modelo de processo. Para eles, é mais fácil ler um texto em linguagem natural. Por esta razão, ambos modelos e textos são artefatos necessários para a comunicação adequada entre especialsitas de negócio e analistas de sistemas. A gestão manual de ambos recursos pode gerar inconsistências, devido a alterações unilaterais. Esta dissertaão propõe uma metodologia para geração de texto a partir de modelos e vice-versa, aplicando as alterações textuais no modelo de processo original. A proposta foi validada através do desenvolvimento de um framework genérico, instanciado utilizando a linguagem de programação Java. A metodologia foi avaliada através de estudo de caso e experimentos. Três conclusões principais podem ser obtidas atraves da avaliação. Primeiro, descrições textuais de processo podem ser consideradas equivalentes a modelos em termos de representação de conhecimento (*74% dos entrevistados avaliam a equivalencia do conhecimento transmitido por ambas representações entre 68% e 100%*). Segundo, o formato escolhido para representar textualmente os processos é boa (*86% dos participantes avaliaram a qualidade das descrições textuais entre excelente e boa.*) Terceiro, o conhecimento representado pelos descrições textuais que foram manualmente alteradas podem ser consideradas equivalentes aos modelos de processo atualizados autoamticamente (*78% dos entrevistados avaliaram que o conhecimento representado pelo texto alterado manualmente é equivalente ao modelo de processo gerado apos a atualização automática*)

Palavras-chave: BPM, BPMN, Geração de Linguagem Natural, Processamento de Linguagem Natural.

Table of Contents

1 Introduction **1**

1.1 Contextualization 1

1.2 Motivation 2

1.3 Problem Characterization 4

1.4 Proposed Solution 5

1.5 Methodology Applied 9

1.6 Work Structure 11

2 Theoretical Foundation **13**

2.1 Business Process and the BPMN notation 13

2.2 NLG 19

2.2.1 Intermediate Structures Used in NLG 20

2.2.2 Techniques for Translating Machine Artifacts 22

2.2.3 Necessary Steps to Develop Natural Language Generation Systems 24

2.2.3.1 Pipeline Architecture 24

2.2.3.2 Data-to-Text Architecture 26

2.2.4 NLG Steps in the context of BPMN text generation 27

2.3 NLP 30

2.3.1 Syntax Parsing: Part-Of-Speech Tagging (POS) 31
2.3.2 Tokenization 32
2.3.3 Stop Words Removal 33
2.3.4 Anaphora Resolution 34
2.3.5 NLP Steps in the context of BPMN text generation 34
2.4 Chapter Summary 36

3 The Round Trip Approach **37**

3.1 Generic Framework for Natural Language Text 39
3.1.1 The Framework Architecture 40
3.1.2 Framework classes and Interfaces Specification 48
3.2 Model to Text: Natural Language Generation from BPMN process models 53
3.2.1 Text Planing 53
3.2.2 Sentence Planning 54
3.2.3 Sentence Realization 55
3.3 Text to Model: BPMN process model generation from Natural language Texts 56
3.3.1 Text Planing 57
3.3.2 Sentence Text Planning 60
3.3.3 Process Model Realization 62
3.4 Strategy for Incorporating Textual and Model-based Changes 65
3.4.1 Textual Diff 67
3.4.2 Updating the original process model 69
3.5 Framework's activities sequence 73
3.6 Chapter Summary 80

4 Evaluation **81**

4.1 PoC: Proof-of-Concept 81

4.2 Experiment: Business Process Automated Description Quality 86

4.2.1 Experiment Design 86

4.2.2 Analysis and Discussion 92

4.2.3 Threats to Validity 98

4.2.4 Conclusion 101

4.3 Experiment: Text-Model Synchronization 101

4.3.1 Experiment Design 102

4.3.2 Analysis and Discussion 105

4.3.3 Threats to Validity 107

4.3.4 Conclusion 108

4.4 Chapter Summary 108

5 Related Works **109**

5.1 Business Process Understandability 109

5.2 Process Models to Textual Descriptions 111

5.3 Textual Descriptions to Process Models 112

5.4 Comparison 115

5.5 Use of NLP techniques in other contexts 118

6 Conclusion **121**

6.1 Limitations 123

6.2 Further Research 124

A Appendix - Algorithms **139**

B Appendix - Process Models and Textual Descriptions 155

C Appendix - Extending the Framework to new languages 165

List of Figures

1.1 Illustration of the round-trip approach. 6

1.2 Research methodology underlying this master thesis. 10

2.1 Level of Abstractions for process model notations [137]. 14

2.2 BPMN elements summary, adapted from Michele and Alberto [17]. . . . 15

2.3 Considered subset of BPMN symbols, adapted from Michele and Alberto [17]. 16

2.4 Business Process Model sample using the BPMN notation [62]. 18

2.5 (a) A graph and its fragments, (b) a RPST that represents the graph depicted in (a). 20

2.6 Abstract version (Graph) from the process model depicted in Figure 2.4. . 21

2.7 Corresponding RPST from the abstract process depicted in Figure 2.6. . 21

2.8 Simple example of a DSynT Tree, based on the sentence 'The room-service manager takes down the order.' 22

2.9 NLG pipeline approach, defined by Reiter and Dale. 25

2.10 Extended pipeline approach. 26

2.11 Extended pipeline (data-to-text) architecture. 28

3.1 Illustration of the round-trip approach. 38

3.2 NLG Core Architecture - UML Package diagram. 42

3.3 Implementation of the hot spots defined by the architecture. 43

3.4 NLP Core Architecture - UML Class diagram. 44

3.5 Classes used to generate a natural language text from a business process model. Together they form the NLG core. 49

3.6 Classes used to deal with the linguistic rules of a specific language. 49

3.7 Interface responsible for the definition of methods needed to classify a label into one specific style. 51

3.8 Interface responsible for the definition of methods needed to extract information from a specific label style. 51

3.9 Interface responsible for the methods definitions of text analysis inside a process model label. 51

3.10 Interface responsible for all the necessary methods definitions for the storage and retrieval of the label properties. 52

3.11 Interface responsible for the definition of the necessary methods which are called to generate the final text. 52

3.12 Text to model pipeline - Steps performed to generate BPMN model from natural language text. 57

3.13 Process fragment used for illustration purpose. 61

3.14 Example of the relations among the objects seen as a Tree Structure. . . . 63

3.15 *SentenceText* instance A is broken into two parts, to address the Atomicity challenge: *SentenceText* instance A' and *SentenceText* instance B. 64

3.16 Simple process, represented in both text and model, designed only to illustrate a simple role change example. 67

3.17 Updated version of the previous simple process, represented in both text and model. 67

3.18 Original process model and its updated version, after having textual changes reflected to it. 71

3.19 Original process model and its updated version, after removing a gateway sentence from the text. 72

3.20 Activity Insert: Original process model and its updated version, after having textual changes reflected to it. 73

3.21 Gateway-And Insert: Original process model and its updated version, after having textual changes reflected to it. 73

3.22 Sequence diagram representing the steps to generate text from model. . . 76

3.23 Sequence diagram representing the steps to update the model from changes in its text representation. 79

4.1 Evaluation methodology used for the Proof of Concept. 84

4.2 A text-model pair describing a process fragment. 89

4.3 A text-model pair describing a process fragment, which is not equivalent. 90

4.4 Subject's answers distribution among equivalence intervals. 93

4.5 Participant's answers distribution among five groups (Ungrouped). 94

4.6 Participant's answers distribution among two groups (Grouped). 94

4.7 Chart that shows subject's answers distribution among the options available regarding the textual description quality. 95

4.8 Chart that shows subject's answers distribution among the options available regarding the textual description quality. 96

4.9 Two clusters identified by the hierarchical clustering technique. 97

4.10 Chart that shows experienced subject's answers distribution among equivalence intervals. 98

4.11 Chart that shows inexperienced subject's answers distribution among equivalence intervals. 99

4.12 Chart that shows experienced subject's answers distribution among the options available regarding the textual description quality. 99

4.13 Chart that shows inexperienced subject's answers distribution among the options available regarding the textual description quality. 100

4.14 A text-model pair which was used during the experiment. 103

4.15 Subject's answers distribution among the available accordance options (grouped into two groups). 105

4.16 Process fragment which illustrates the scenario where an updated made to a Gateway description should also trigger an event description update. . . 107

4.17 Process fragment which illustrates the optimal scenario where an updated made to a Gateway also update the event description. 107

B.1 Simple Exam application process, written in English. 156

B.2 Simple Exam application process, written in Portuguese. 156

B.3 Simple Secretary process, written in English. 157

B.4 Simple Secretary process, written in Portuguese. 157

B.5 Hotel Service process, written in English. 158

B.6 Hotel Service process, written in Portuguese. 159

B.7 English Bread delivery service subscription in BPMN. 161

B.8 Portuguese Bread delivery service subscription in BPMN. 162

B.9 claims handling process 163

C.1 Dictionary file structure. In this example, it is shown a Portuguese dictionary file. 166

List of Tables

2.1 Steps to the automatic text generation (adapted from Leopold [62]) 29

2.2 Some part-of-speech tags frequently used for tagging English 32

2.3 Steps for Mapping Natural Language Text to Process Models. 35

3.1 Overview of the current template text pattern supported by the NLP Core Component. 47

3.2 Overview of the Process Model elements supported by the NLP Core Component 48

3.3 Supported Labeling styles 54

3.4 Supported Text Pattern received as input. 58

3.5 Overview of Change Operations for Process Model and Text and their Connection, adpated from Kolb *et al.* [51] 66

3.6 Overview of Operations for Updating Links 69

4.1 Overall characteristics of the complete test data set. 82

4.2 Natural Language Text generated by analyzing the process model data and extracting textual information. 85

4.3 Questions about the participant's experience with process models (characterization). 89

4.4 Question about the equivalence between the textual work instruction and the BPMN model, which describes a process fragment. 91

4.5 Question about the textual work instruction quality, which describes a process fragment. 91

4.6 Question about the process model synchronization after having changes made to the original text. 104

5.1 Papers' Comparison 118

B.1 English Natural Language Text generated by analyzing the process model data and extracting textual information. 155

B.2 Portuguese Natural Language Text generated by analyzing the process model data and extracting textual information. 157

B.3 English Natural Language Text generated by analyzing the process model data and extracting textual information. 157

B.4 Portuguese Natural Language Text generated by analyzing the process model data and extracting textual information. 157

B.5 English Natural Language Text generated by analyzing the process model data and extracting textual information. 158

B.6 Portuguese Natural Language Text generated by analyzing the process model data and extracting textual information. 159

B.7 English Natural Language Representation for the Bread delivery service subscription 160

B.8 Portuguese Natural Language Representation for the Bread delivery service subscription 163

B.9 English Natural Language Text generated by analyzing the Claims Handling process model and extracting textual information. 164

List of Algorithms

1 identifySentenceType algorithm, from PortugueseLanguageProcessor implementation . . . 140
2 extractActivityProperties algorithm, from PortugueseLanguageProcessor implementation . . . 141
3 extractEventProperties algorithm, from PortugueseLanguageProcessor implementation . . . 142
4 extractGatewayProperties algorithm . . . 142
5 processTextDiff algorithm . . . 142
6 processRemovals algorithm . . . 143
7 processSentencesRecursivly algorithm, from SurfaceRealizer class . . . 144
8 processInserts algorithm . . . 145
9 removeStopWords algorithm, from ILabelHelper interface specification . 146
10 checkForConjunction algorithm (for Portuguese) . . . 147
11 removeArticleFromBo algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 147
12 getPrepositions algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 148
13 getPrepositions algorithm (for English) . . . 148
14 isAdverb algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 148
15 isVerb algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 149
16 isNoun algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 149
17 isAdjective algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 149
18 getInfinitive algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 150

19 getParticiple algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 150
20 getPresentForm algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 150
21 is3PS algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 151
22 transformToSingularForm algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 151
23 getGender algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 152
24 getArticle algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 152
25 isDefArticle algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 153
26 isPronoun algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 153
27 getPronouns algorithm (for Portuguese) from PortugueseLabelHelper implementation . . . 153
28 posTagging algorithm . . . 154

List of Abbreviations

BPM	Business Process Management
BPDM	Business Process Definition Metamodel
BPMN	Business Process Model Notation
BPMS	Business Process Management Systems
BPQL	Business Process Query Language
DSynt	Deep-syntatic tree
EPC	Eventdriven Process Chain
JSON	JavaScript Object Notation
NLG	Natural Language Generation
NLP	Natural Language Processing
NLU	Natural Language Understanding
OMG	Object Management Group
PoC	Proof-of-Concept
POS	Part-Of-Speech Tagging
RDF	Resource Description Framework
RE	Regular Expressions.
REGEX	Regular Expressions.
RPST	Refined Process Structure Tree
SPARQL	Simple Protocol And RDF Query Language
UML	Unified Modeling Language
YAWL	Yet Another Workflow Language

1. Introduction

This section provides an introduction to this master thesis. Section 1.1 contextualizes the work. After a discussion of the motivational aspects in Section 1.2, the contribution to the body of knowledge of Information Systems research is highlighted in Section 1.3 and Section 1.4. Section 1.5 enlightens the contribution in the perspective of design science. Then, the introduction concludes by providing an outlook on the structure of this master thesis in Section 1.6.

1.1 Contextualization

A model is a representation of an idea, an object or even a process or a system that is used to describe and explain phenomena that cannot be experienced directly. It is an abstraction of the reality. Models are central to what scientists do, both in their research and when communicating their explanations. They guide research through simplified representations of an imagined reality that enable predictions to be developed and tested by experiments [120]. Models can be found throughout several disciplines and in different formats. For instance, although not common to be referenced as such, textual descriptions can be considered as a specific model format since it also abstract the reality through natural language text. Models are used extensively in Computer Science and Information Systems disciplines based in their expressiveness power when compared to other forms of representations, *e.g*, textual descriptions [8].

It is widely accepted that presenting data in the form of models, graphs, diagrams or pictures can enhance data comprehension, decision-making and communication of information about data [14, 124]: "*A picture is worth a thousand words*". The mechanism behind this effect is through inducing cognitive processes, such as visual chunking, mental imagery and parallel processing [139], as an external aid to reduce demands on human memory [12], or to assist mental integration of complex data [100].

However, although the instructional and educational potential of models (or graphs) is widely acknowledged, in several cases, models are not always more effective than other methods of representation. For example, many studies on process model understandability have shown how the comprehension of process models may be complex, even for people who are familiar with process modeling [29, 103]. There are also some studies indicating that it is preferable to discuss the concepts expressed by conceptual models through natural language text [15]. In particular, work instructions that describe tasks at a high level of detail are often documented in the form of textual descriptions, as this format is more suitable for specifying a high number of details [3]. As a consequence, the joint use of process textual descriptions and models is important to achieve better understanding.

Business process models provide an abstract graphical view on organizational procedures by reducing the complex reality to a number of activities. By doing so, they help to foster an understanding of the underlying organizational procedures, serve as process documentation, and represent an important asset for organizational redesign [58]. They have proven to be an effective means of specification [13].

Business process management is a discipline which seeks to increase the efficiency and effectiveness of companies by holistically analyzing and improving business processes across departmental boundaries and plays an important role both in business process redesign initiatives and in the development of process-aware information systems. In order to depict business processes, many companies use specific notations such as BPMN [50], which was developed and standardized by the Object Management Group [88].

1.2 Motivation

The process models resulting from projects for process-aware information systems have to meet high quality standards since the costs of correcting errors grow exponentially if they are discovered at later stages. For this reason, it is of paramount importance to extensively validate the created models already in the beginning of the development project [22]. Nevertheless, process models in practice have a substantial amount of quality issues [75, 76], which can become a threat for the success of the whole modeling project [4]. The reasons for bad quality can be manifold. One of the central issue in this context is the persons involved with a modeling project might not have sufficient experience with modeling [111]. This can be the case when business professionals are asked to model the processes of their department themselves or when process analysts have to interact with business professionals who are not capable or willing to study the created

process models, resulting in a communication gap between the domain expert and the modeling expert [16]. A way to deal with these challenges is to enhance the modeling process with steps for validation and verification. However, while verification has been intensively studied in recent research [128], there has been a notable research gap on how the validation of a process model can be supported by a modeling tool if the modeler is not a modeling expert.

While notations like BPMN have been proven to be useful in many different scenarios, it still represents a challenge for many employees to fully understand the semantics of a process model [29, 103]. For example, often the validation and the usage of process models is hampered by the fact that many domain experts are not able to understand the models in detail [62]. If the reader is not familiar with the wide range of elements and concepts (*i.e.*, gateways, events, or actors), large parts of the process may remain unclear or even be misunderstood by the reader [32]. Training employees in understanding process models is associated with a considerable amount of time and money and can hardly be considered an option for the entire workforce of a company.

For this reason, modeling experts are required to iteratively formalize and validate the models in collaboration with the domain experts. But, this traditional procedure of extracting process models through interviews, meetings, or workshops tends to be cost and time intensive due to the informal setting and ambiguity or misunderstandings between the involved participants [113, 102]. Therefore, the initial elicitation of conceptual models presents a knowledge acquisition bottleneck.

To reduce the process knowledge bottleneck, many so-called verbalization techniques have been proposed that allows mapping conceptual models to natural language [86]. For data modeling, verbalization techniques provide a direct mapping from model to natural language assisting the discourse between system analysts and domain experts in the context of requirements engineering [132, 86, 38]. For process modeling, verbalization techniques are responsible for transforming process models to natural language text, which can be read and understood by persons without specific modeling skills [110, 62, 63, 33, 34]. Nevertheless, while these works allows for generating text from a process models, it does not provide the possibility to generate process models from natural language text, nor to reflect edits in generated text back to the process model. As a consequence, domain experts are not able to alter or create process models without relying on modeling experts.

1.3 Problem Characterization

Many companies maintain both process models and textual work instructions to depict its processes [129]. The use of both knowledge representations is needed to address specific audience. While domain experts are usually not qualified for reading process models, having to rely on textual descriptions, modeling experts prefer using the model representation. This, however, means that companies face redundant effort for updating both process knowledge representation artifacts.

For instance, when a domain expert needs to update a process description, it must: (i) fetch its natural language representation; (ii) update the textual description with the desired changes; and, (iii) update the original process model, reflecting the process changes. The first task, *i.e.*, generating the textual description from the original process model, can be completely automatized by using the proposed process verbalization techniques. Nevertheless, the last task must be manually performed by someone with modeling skills and experience with the specific process model notation. As most domain specialists do not have such set of skills and experience, the last task must be delegated to system analysts [32]. This is prone to several inconsistencies problems, as only one of the artifacts is modified or updated.

Thus, in order to enable domain experts to create or update formal models simply through a textual description and to leverage the information potential of already existing text documents, an automatic transformation technique is needed. The transformation technique must be also capable of preserving the process verbalization techniques, integrating it with process model generation. By providing automation support, it will be possible to achieve substantial savings, solve the mentioned inconsistency problem and to enable a quicker realization of BPM-projects and their benefits.

Therefore, the goal of this master thesis is to develop an automated transformation technique, deploy it in a prototype implementation, and to evaluate it.

The main research question addressed by this master thesis is: *How may organizations maintain both business process representations (models and textual descriptions) automatically synchronized?*

1.4 Proposed Solution

The goal of this work is to define a technique that is able to generate natural language texts from process models and also apply changes to the model from editions on the text, *i.e.*, a `round-trip` approach (model-text-model). To accomplish such goal, our technique is based on correlating elements from the process model to text elements (sentences) in the natural language text. Based on this, we are capable of generating text elements from business process elements and storing its relations in intermediate structures. Through these intermediate structures, we explore the information and map them as natural language sentences. Changes on to the natural language text are reflected back to the original process model, due to the tracking of changes in both artifacts. In other words, if the user change the process model, the text is updated accordingly and if the text is updated by the user, the process model is also updated. We called this technique as Round-trip, because it is capable of closing a synchronization cycle between both knowledge representation artifacts. The designed technique can be adapted to different languages, *e.g.*, English, Portuguese, and German.

Figure 1.1 details the proposed solution, which is explained in more details in Chapter 3. Using our approach domain specialists do not need to expend training time in a specific notation or in process modeling discipline. They are able to update process model artifacts by editing a natural language text derived from the original process model. As to system analysts, they are relieved from the time-intensive modeling task by writing and updating natural language texts.

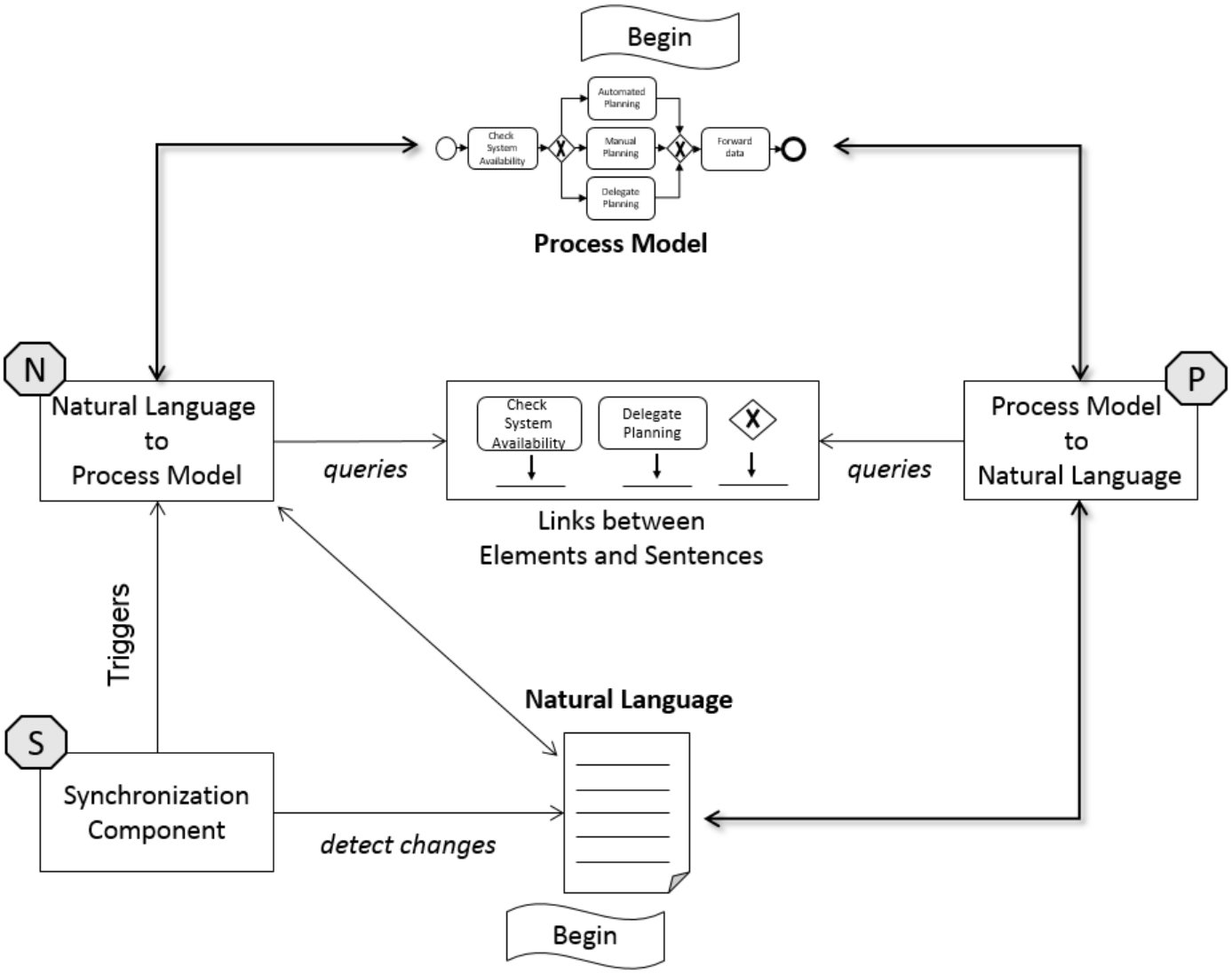


Figure 1.1: Illustration of the round-trip approach.

This master thesis provides the following contribution to the body of knowledge of Information Systems research:

- **Literature Review -** In-depth study of the works done in the area, including concepts, case studies and techniques related to their original themes. These works were evaluated regarding their strengths and weaknesses.

    1. Research on the state of the art in NLG (Natural Language Generation), specially regarding natural language text generation from business process models;
    2. Research on the state of the art in NLP (Natural Language Processing), specially regarding process models and the problem of automated business process model generation;

3. Research on the state of the art in Model learning and refinement from natural language text;

- **Categorization of Issues -** Based on these findings, a list of challenges, which are relevant for a business process model transformation approach, is presented to guide future works on this particular research field.

- **Novel Transformation Approach -** A framework was developed based on the list of challenges and the associated benefits of a round-trip. It is the first which does permit the user to generate natural language texts from a machine artifact (*e.g.*, BPMN process model) and to synchronize BPMN models through textual operations made to natural language texts, in the same environment. Furthermore, allows the extension and customization of natural language processing algorithms and strategies through the use of specific interfaces. Finally, it covers a wider spectrum of modeling constructs then other approaches and includes support for multiple languages through the definition of a natural language-independent module.

- **Prototype -** The proposal was implemented and tested in a research prototype. The source code is available at https://bitbucket.org/rar150/unirio-workspace/src.

- **Extensible and Language-Independent Framework -** Several interfaces were defined in the core framework's components, making it generic and extensible for new implementations that may be better suitable for specific scenarios that were not defined beforehand. This allows the framework's reuse by implementing specific interfaces methods.

- **Modeling Language Independence -** In industry, several modeling languages are used. Frequently found examples include BPMN, EPCs, and UML. Existing solutions often only focus one of these languages (see *e.g.*, [84, 60]). The business process model specific generation component (BPMN Model Generation) was designed as a separate independent module. It can be replaced by a custom module to offer support to new notations. As a result, we are able to define a round-trip approach that can be employed for commonly used modeling languages.

- **Evaluation Approach -** To assess the accuracy and quality of the transformation approach, we created an evaluation methodology based on a PoC (Prof of Concept) and experiments.

- **Technological and Scientific Contributions**:

  - If business users are not familiar with business process modeling notations, they can edit a natural language text created from the process model. Then, the updated text can be automatically transformed back into an updated version of the process model.
  - If one artifact is changed, the changes can be automatically reflected in the other artifact.
  - It is even no longer necessary to maintain both artifacts. A company can simply maintain a repository of process models. The corresponding natural language texts can be automatically generated at any time and are always consistent with the process models.
  - There is no need to train employees in BPMN notation. If the training is necessary, natural language texts can be used to short the learning curve.
  - The communication between business specialists and system analysts flow more naturally due to the use of specific artifacts according to their specific skills or knowledge.

- **Publication of partial and final results -** During this research, several questions concerned with linguistic analysis in process models were addressed. The following papers document the results:

  - *An Experiment on Process Model Understandability Using Textual Work Instructions and BPMN Models* - Paper that describes an experiment to address if there are significant differences in terms of process understand ability depending on whether textual work instructions or process models are used to represent a business process. The experiment can be classified as *"true experimental design"* [101], because randomization techniques were applied during the sample selection, to ensure the groups were similar among each other. In total, 73 individuals (students and IT professionals) were selected to participate in the experiment. The results and data gathered from the experiment are depicted in more details in [108].
  - *A Tool to Generate Natural Language Text from Business Process Models* - Paper that describes the initial version of the proposed framework, before concluding the round-trip approach. The presented version was capable of generating full natural language text from process models. The tool usage was demonstrated for the audience through live sessions during the conference [110].

– *Text Generation from Business Process Models* - Paper that document the main challenges associated with generating natural language from machine artifacts, presents two types of architectures for developing NLG systems and describes several systems which are based on these architectures, specially regarding business process models [109].

– *Automatic Synchronization of BPM Description Artifacts: Methods and Applications* - Paper that describes a theorical methodology for text generation from process models and vice-versa. It presents the main concepts for developing a round-trip approach [98].

1.5 Methodology Applied

The structure of this methodology is composed by a four step approach and is illustrated in Figure 1.2. By following this methodology, the contributions of the research conducted in this master thesis follows the paradigm of design science, as defined by Hevner [45]. The steps are detailed as follows:

- **Test Data Sets (Step 1)** The first step was to collect initial test data, consisting of several process models and natural language process specifications. They provided insights on how to translate business process models to textual descriptions and vice-versa.

- **Syntactic and Semantic Patterns (Step 2)** by analyzing the linguistic patterns we were able to map text constructions to their corresponding modeling fragments. This also revealed issues regarding the quality of the syntax parse and the text itself. We systematically categorized these issues and developed a conceptual solution strategy.

- **Transformation Rules (Step 3)** To effectively mitigate the detected issues, appropriate transformation heuristics were defined and refined iteratively. The rules were then implemented in our research prototype to assess the output.

- **Evaluation (Steps 4 and 5)** against several process models (detailed below) providing important information about the framework's behavior. The gathered information could then be used in another iteration to refine or discover new patterns and rules and to improve the performance regarding the test data set.

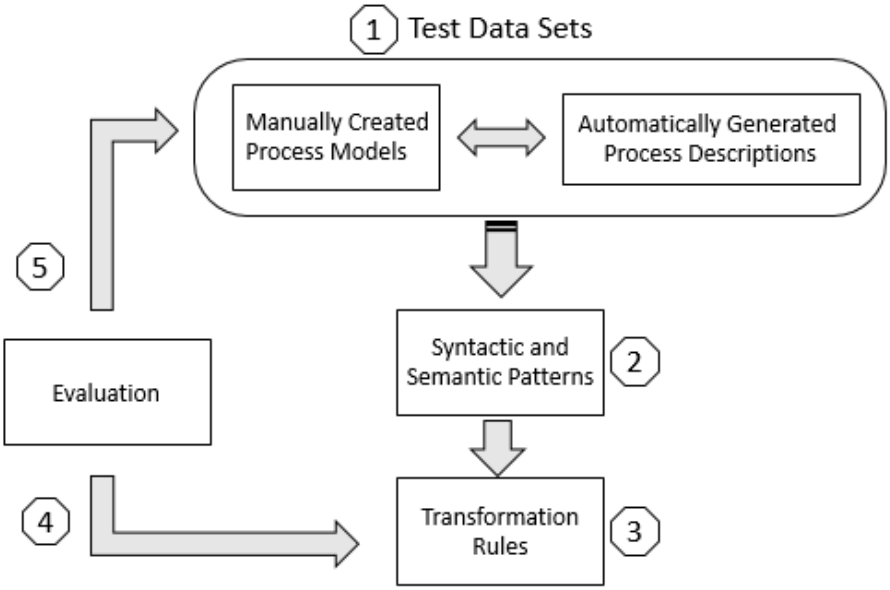


Figure 1.2: Research methodology underlying this master thesis.

The research methodology applied to evaluate the proposed solution was based on the development of a prototype which serve as a proof-of-concept, specification of experiments involving qualitative and quantitative researches [134]. The research methodology is detailed below:

1. Basically we conducted a two-step evaluation. At first, artificial data were generated, which represented a control experiment. This experiment was composed by exclusively made-up models[1] (written by the authors) to stress specific conditions (scenarios) that should be covered by our solution. The framework was designed to support multiple languages, thus the set of process models were written in both English and Portuguese. During this evaluation, two analysis were conducted:

    - The output texts were compared with the models used as inputs, a set of text metrics to investigate in how far the generated texts are comparable to manually created texts and if the textual description is capable of representing the same process knowledge as a business process model;
    - After asserting the texts were suitable, several textual operations (*e.g.*, add, change and remove text fragments) were applied to output texts. Two snapshots were taken from the original process model, the first before submitting the changes and the other after the synchronization of textual changes. The snapshots were then compared to the textual operations to investigate whether

[1] This set of process models (artificial data) are available in the Appendix (Section A)

the framework was capable of correctly mapping text-based changes to model-based changes.

After the initial evaluation with artificial data, real data was used through a second set of business process models[2] gathered from universities and from companies to stress real scenarios.

2. Afterwards, some research questions were defined to serve as a guide during the execution of the experiment, which had two main objectives: validate and evaluate the natural language text produced as the framework output when given a process model instance as input. Basically, it was an exploratory research to investigate weather the generated text is capable of transmitting the same knowledge as compared with a Business Process Model. It was also possible for the participant to leave comments (*i.e.*, feedback) about the generated text structures.

3. Finally, a third experiment was run to evaluate the synchronization components through editions made to the original text and asking the business process expert to evaluate whether the changes were reflected correctly to the original process model. During this experiment, the participant was asked to leave his opinion (*i.e.*, feedback) about the synchronization process.

The results of this master thesis are twofold. On one hand, algorithms were developed to generate texts from process models, to detect changes made to the generated text, and to allow automatic updating of the original model considering these detected changes. On the other hand, these algorithms were packaged in a flexible tool (*i.e.*, framework), capable of producing texts from process models written in different languages, allowing automatic update of the original model according to changes in the text.

1.6 Work Structure

The remainder of this work is structured as follows.

Chapter 1 corresponds to this introduction. Chapter 2 presents the background, including business process concepts, the BPMN notation and some strategies used to develop natural language generation systems. Some challenges in the context of BPMN model generation are illustrated for both, NLG and NLP research areas. Chapter 3 presents the

[2]Due to copyright reasons, the whole set is not available. Nevertheless, some process models are available in the Appendix (Section A)

implementation of the proposed `round-trip` technique (Figure 1.1). Chapter 4 corresponds to the evaluation of the methods developed and implemented within the language-independent framework.

Chapter 5 presents related articles, which describe approaches similar to the one presented in this master thesis. These approaches are explained and their differences are highlighted. Finally, Chapter 6 presents the conclusions, proposals for future work and highlights this research's limitations.

2. Theoretical Foundation

This master thesis covers an interdisciplinary topic. It relies on concepts and research in the areas of Conceptual Modeling, Natural Language Generation (NLG), Natural Language Processing (NLP) and Business Process Modeling Notation (BPMN). This section presents some business process concepts, the BPMN notation and some strategies used to develop natural language generation systems. These strategies are generic and widely used for the development of tools aimed at dealing with manipulation of texts in natural language. Afterwards, some steps in the context of BPMN model generation are illustrated for both, NLG and NLP research areas. These steps are addressed by this master thesis and explained in details in Chapter 3.

2.1 Business Process and the BPMN notation

The need for well defined and flexible business process is growing in many organizations [50]. Companies have to put considerable effort into the design, implementation, execution, monitoring, and evaluation of processes and the corresponding data. Business Process Management Systems (BPMS) are often employed to provide the necessary software support for those activities. Weske defined BPMS as "*A generic software system that is driven by explicit process representations to coordinate the enactment of business processes*" [137]. Such explicit representations are conceptual models, which are the result of the design phase as depicted in the BPM life-cycle. In the area of BPM, different kinds of conceptual models are required, including representations of functions, data, organization, system landscapes, and perspective process models [113]. Such process models provide an abstracted representation of several business process instances which are the interactions within a company conducted to create value. It includes activities conducted by humans and/or software systems and is able to show interdependencies.

There are several notations to model business process, such as: UML AD (UML Activ-

ity Diagrams) [125], BPDM (Business Process Definition Metamodel) [88], EPC (Event-driven Process Chain) [114], BPQL (Business Process Query Language) [88], BPMN (Business Process Modeleling Notation) [88] and YAWL (Yet Another Workflow Language) [127]. These languages define the meta-model (Figure 2.1), *i.e.*, the rules and structure the model has to follow. Due to the existence of these several notations, a standard is necessary to model company business processes [50].

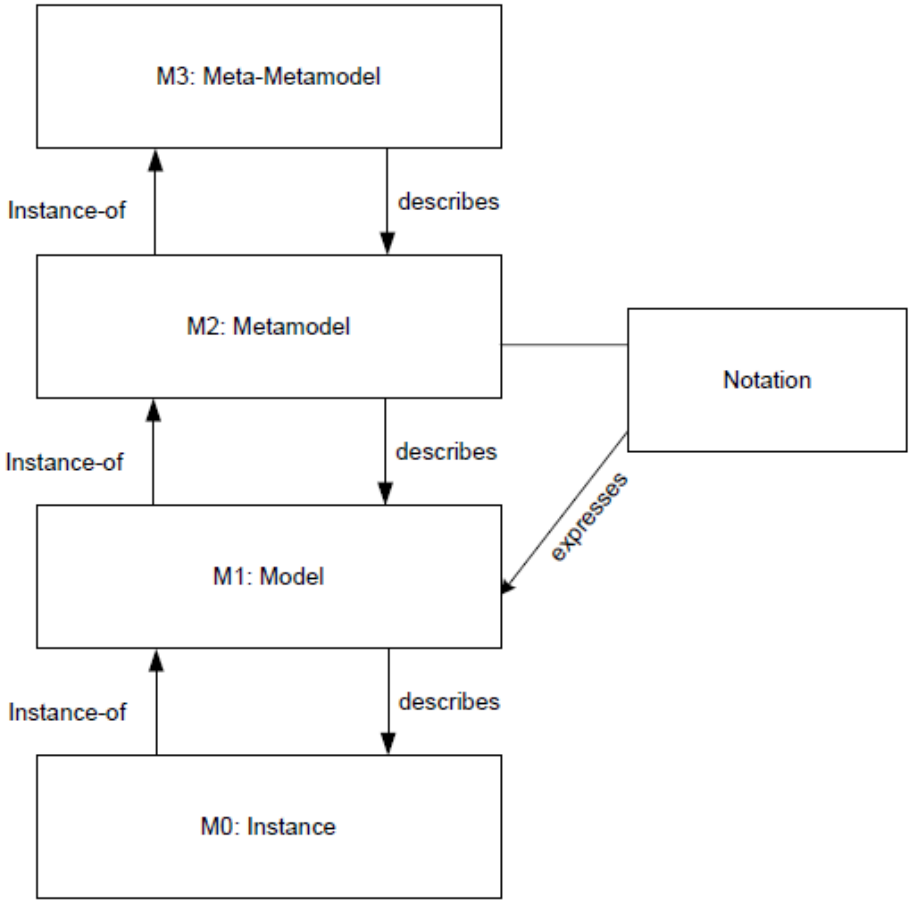


Figure 2.1: Level of Abstractions for process model notations [137].

For the approach developed in this research we decided to use BPMN 2.0[1] as language or metamodel for the resulting process models [88]. BPMN is an official standard supported by the Object Management Group (OMG) for process modeling, providing rich expression capabilities. Furthermore, an evaluation of several notations was conducted by Ko *et al*. [50], which pointed out the BPMN as the most used one. They also depict the trend BPMN becomes the *de facto* standard for business process modeling.

The Business Process Model Notation (BPMN) is a graphic process modeling language used for the specification of business process and has a big set of specific symbols

[1]For more detalis, refer to: http://www.omg.org/spec/BPMN/2.0/

for the modeling task. Currently, the BPMN is in its 2.0 version, published by OMG [88]. With the introduction of BPMN 2.0, each element of a BPMN process diagram has clearly defined semantics. The BMPN works as a proxy between all the organization's areas, lowering the distance between the specification and the execution of the process [62].

In general, graphic notations are easier for business users to understand and use. Models can make explicit several process's patterns, flaws in process cycles and even bottlenecks or deadlocks. But, it is required that all the people involved with the process model have the necessary knowledge about the model notation. Besides, it is important to define a set of business process models samples that could be used as guides for the process model developers [62].

Figure 2.2 depicts all symbols from the BPMN 2.0 specification. As can be observed, the symbol set of BPMN covers four types of elements: flow objects (activities, events, gateways), swimlanes (pools and lanes), artifacts (*e.g.*, data objects and annotations) and connecting objects (sequence and message flow associations) [88]. This master thesis works with the core subset of the BPMN symbols. This subset is depicted in Figure 2.3, and explained below.

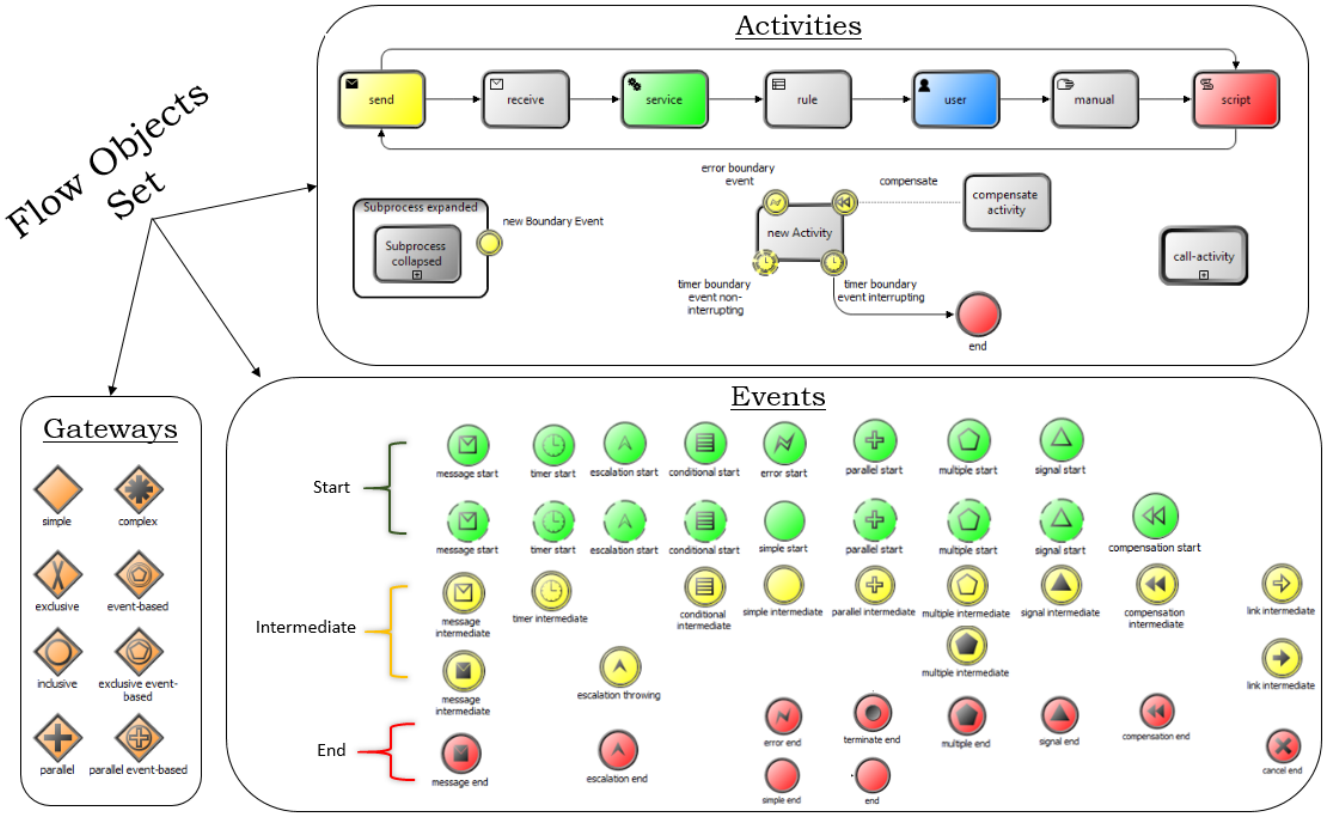


Figure 2.2: BPMN elements summary, adapted from Michele and Alberto [17].

A Task element is an atomic Activity within the process flow. Whenever a group of

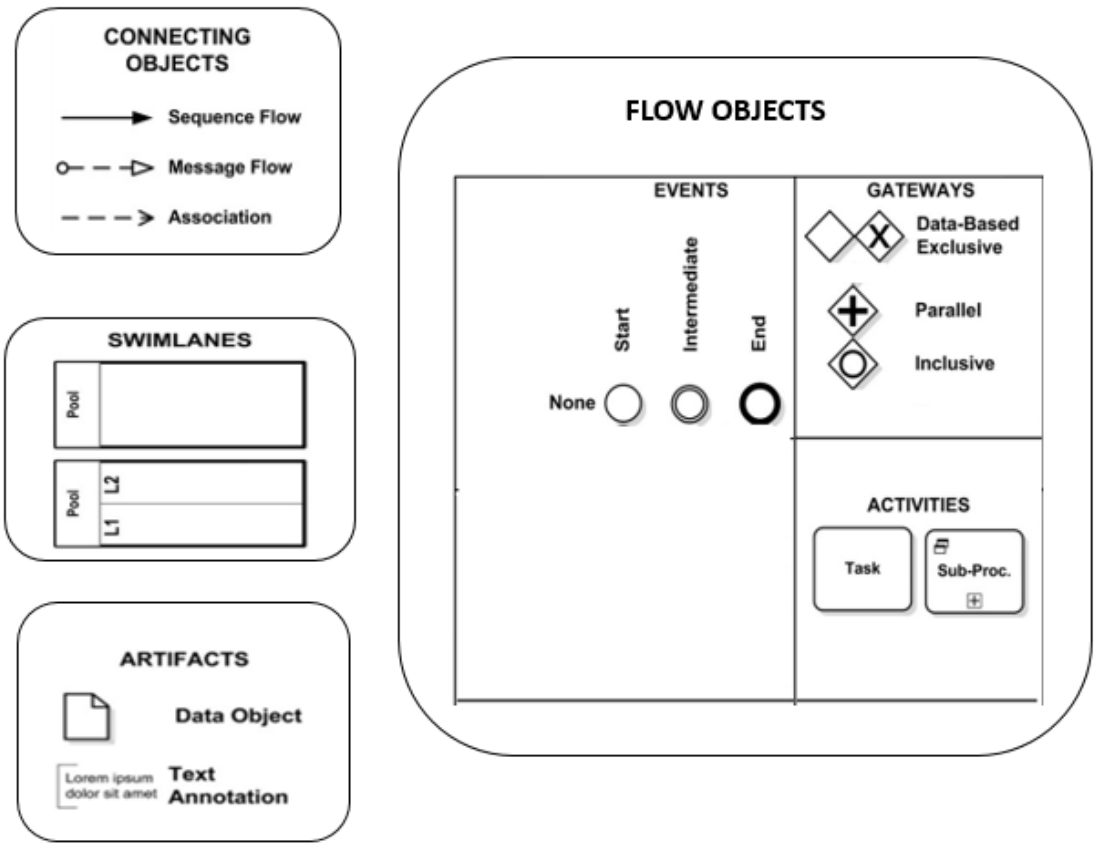


Figure 2.3: Considered subset of BPMN symbols, adapted from Michele and Alberto [17].

activities is combined or the reuse of a process fragment is intended, a Subprocess can be used instead of a Task. A Subprocess can be collapsed and expanded (hence the plus sign at the bottom) to either show or hide its details.

Gateways enable the process flow to be split and joined. An exclusive Gateway (or XOR Gateway) is used to model a decision. Out of all its incoming or outgoing edges only one path will be selected. An event-based exclusive Gateway shows the same behavior, but requires that all of its successors are events. The semantics of a parallel Gateway are different as it will activate all of its outgoing paths or requires that all of its incoming paths are activated. Thus, it can be used to model concurrent behavior. The inclusive Gateway (also OR Gateway) can activate 1-n of its in/outgoing edges or requires them to be activated in order to proceed. It is thus more versatile, but also more complex [82].

Event nodes can be used to denote the start or end of a process. Additionally, intermediate Events can be used within the process flow to make clear that the process will halt and wait for the expected Event to occur. The nature of the Event can be signified by additional symbols, *e.g.*, for Messages, Time or Exceptions as shown in Figure 2.2. BPMN specifies 13 different event types which are fully listed in the BPMN 2.0 specification [88].

Lastly, Pools and Lanes represent actors or roles participating in the process. A Pool can represent a human, organization, or software system involved in a business process. While a Pool is used to show the boundary of an organization and to determine the involved process participants, a Lane can be used to partition the Pool and show the different process participants within that body. Again, this could be different individual, organizational units or software systems. Whenever the behavior of an involved participant is supposed to be left unspecified, a Black Box Pool can be used. This way other process participants and the interactions with them can be shown in a diagram without the need to specify their behavior directly.

BPMN distinguishes between three types of edges or Connecting Objects. Sequence Flows are used to connect Flow Objects within the same Pool. Message Flows have to be used whenever an edge crosses the boundary of a Pool. Therefore, it can be used to visualize the interactions between several process participants in a Collaboration Diagram [17]. Associations are used to connect Artifacts to Flows or Connecting Objects.

According to zur Muehlen [87], only few BPMN diagrams use more than 15 different elements. The subset we defined, presented in Figure 2.3, covers all these 15 elements. Therefore, we are confident that the required elements for the majority of BPM projects can be provided by our transformation procedure as it covers the most important and widely used elements [87].

Figure 2.4 depicts an example of a business process in a hotel represented as a BPMN model. It describes how an order is handled. The process has four lanes. The activities performed by the different actors are depicted as boxes with round corners, the diamond shaped gateways define the routing behavior. The meaning of the symbols inside the diamond are: plus "+": parallel execution; a circle: inclusive choice (one or more branches may be executed); and, a cross: exclusive choice (only one branch can be executed). The process can be described as follows:

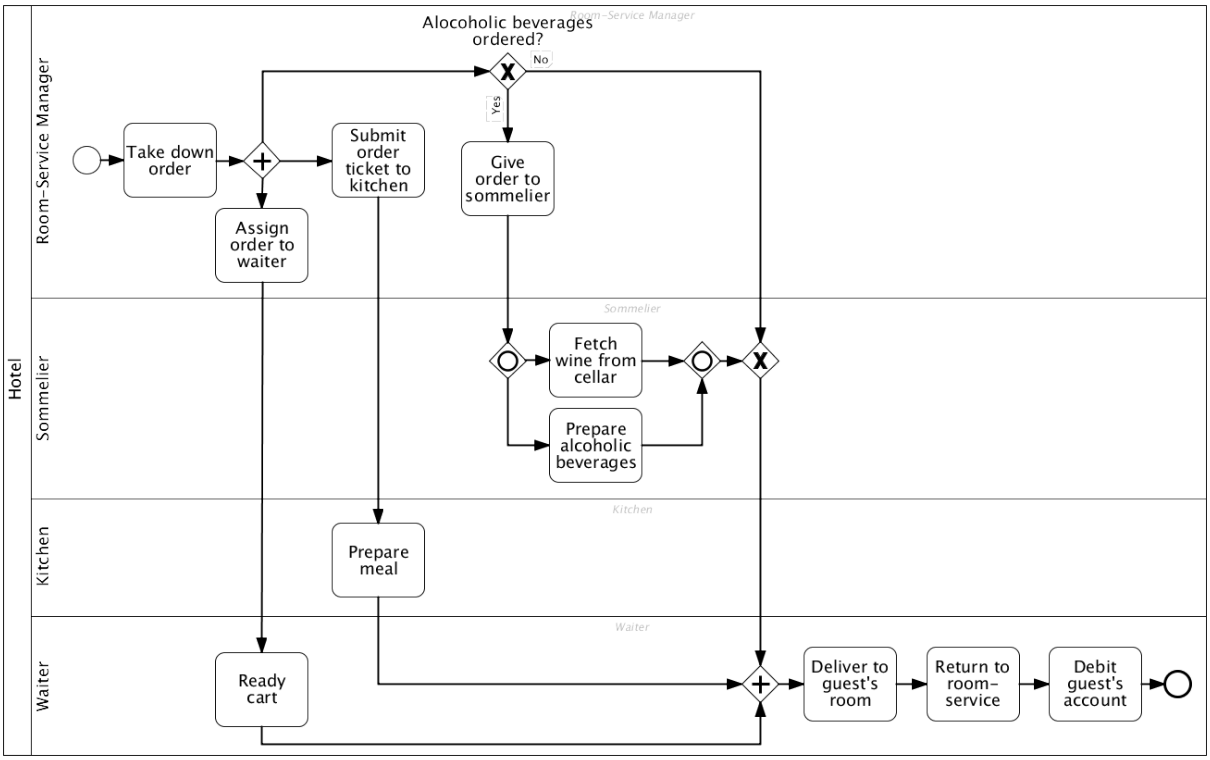


Figure 2.4: Business Process Model sample using the BPMN notation [62].

The process starts when the Room-Service Manager takes down an order. Afterwards, three streams of action occurs in parallel:

- In case alcoholic beverages are ordered, the Room-Service Manager gives order to the sommelier. Afterwards, one or more of the following paths are executed:
  - The Sommelier fetches wine from the cellar.
  - The Sommelier prepares the alcoholic beverages.
- The Room-Service Manager submits the order ticket to the kitchen. Subsequently, the Kitchen prepares the meal.
- The Room-Service Manager assigns the order to the Waiter. Then, the Waiter readies the cart.

As long as all the streams of actions were executed, the Waiter delivers to the guest's room. Afterwards, the Waiter

```
returns to the room-service. Subsequently, he debits from
the guest's account. Finally, the process is finished.
```

In order to be effective, the notation must be known and well understood by business executives, specialists, analysts, and users. Without enough knowledge about the used notation, wrong decisions may be taken due to misunderstandings. This problem can be solved by specialized systems which can automatize the interpretation of business process models. For example, there are works based on an interface that is capable of reading a model (written in English) and extract all the relevant business information, in the same way a human would [110, 62]. The information extracted is then used for generating process textual description, which is an alternative process knowledge representation used by domain experts. This kind of representation is prefarable to discuss the concepts expressed by conceptual models [15]. Nevertheless, these works are not capable of supporting multiple languages nor generating process models from texts. Thus, it is this master thesis's goal to fill this research gap. Section 5 provides a detailed analysis of several works related to this thesis.

2.2 NLG

This section presents Natural Language Generation (NLG) concepts, which are specific to the development of NLG systems. This master thesis proposes a framework that is capable of generating natural language text from business process models. All the NLG algorithms that were deployed within this framework were built based on the concepts presented in this section. In particular, regarding the text generation from a BPMN process model, we have implemented the data structures presented in Section 2.2.1, we have applied the technique of natural language generation presented in Section 2.2.2 and generated the sequence flow described in Section 2.2.3.1.

Several techniques are proposed in the literature to generate natural language text, but only NLG is considered as a true natural language generation technique [20]. Without NLG, it would not be possible to generate natural language text from a process model (*i.e.,* machine artifact). After presenting the techniques for translating machine artifacts, we present the proposed steps necessary for building NLG systems.

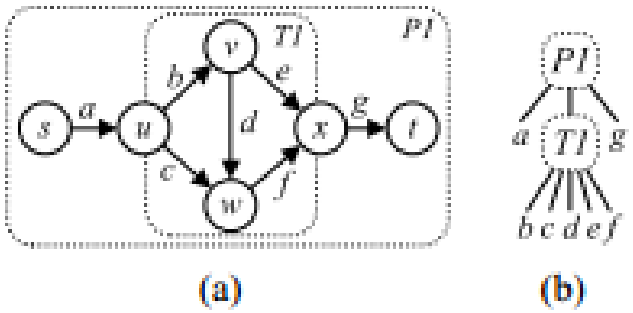


Figure 2.5: (a) A graph and its fragments, (b) a RPST that represents the graph depicted in (a).

2.2.1Intermediate Structures Used in NLG

RPST - Refined Process Structure Tree. It is a parse tree containing a hierarchy of subgraphs derived from the original model. The RPST is based on the observation that every work-flow graph can be decomposed into a hierarchy of logically independent subgraphs having a single entry and single exit. Such subgraphs with a single entry and a single exit are referred to as fragments. In a RPST, any two of these fragments are either nested or disjoint. The resulting hierarchy can be shown as a tree where the root is the entire tree and the leaves are fragments with a single arc [95, 131]. Figure 2.5 depicts a simple RPST tree and its according graph.

In total, we may encounter four different fragment classes: trivial fragments (T), bonds (B), polygons (P) and rigids (R). Trivial fragments consist of two nodes connected with a single arc. A bond represents a set of fragments sharing two common nodes. In BPMN process models, this generally applies for split and join gateways, including more complex split and join structures such as loops. Polygons capture sequences of other fragments. Hence, any sequence in a process model is reflected by an according polygon fragment. If a fragment cannot be assigned to one of the latter classes, it is categorized as a rigid. Although the original version of the RPST was based on graphs having only a single entry and exit point, the technique can be easily extended to compute a RPST for arbitrary process models. Figure 2.6 and Figure 2.7 illustrate the concepts using an abstracted version of the hotel process and its corresponding RPST.

DsynT - Deep Syntactic Tree. It is a dependency representation introduced in the context of the Meaning Text Theory [74]. In a deep-syntactic tree each node is labeled with a semantically full lexeme, meaning that lexemes such as conjunctions or auxiliary verbs are excluded. Each lexeme carries grammatical meta information, which includes voice and tense of verbs or number and definiteness of nouns. The advantages of deep-syntactic

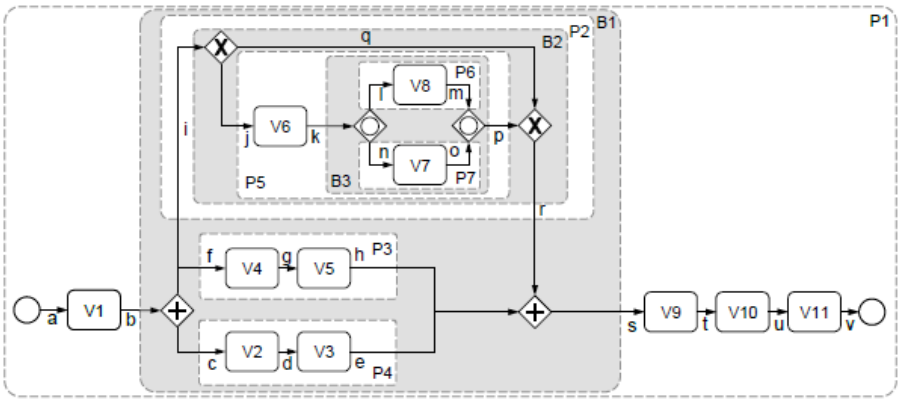


Figure 2.6: Abstract version (Graph) from the process model depicted in Figure 2.4.

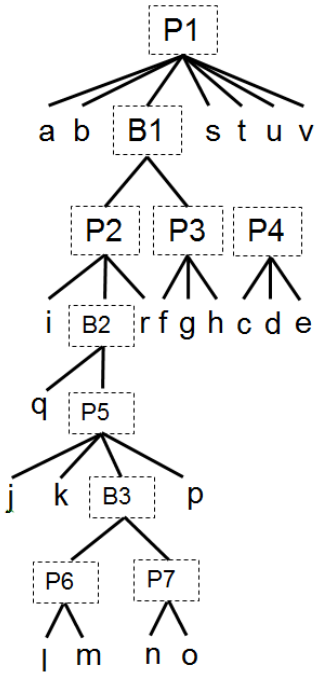


Figure 2.7: Corresponding RPST from the abstract process depicted in Figure 2.6.

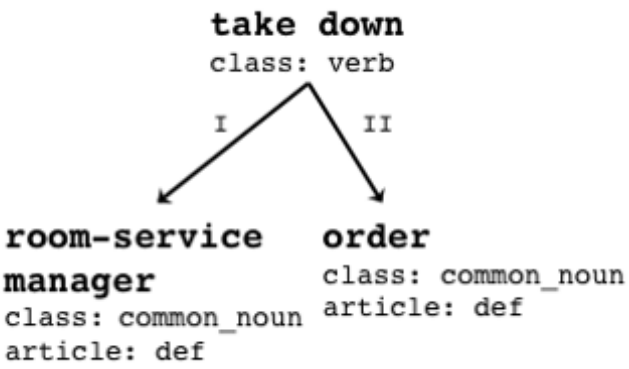


Figure 2.8: Simple example of a DSynT Tree, based on the sentence 'The room-service manager takes down the order.'

trees are the rich but still manageable representation of sentences and the existence of of-the-shelf surface realizers which take deep-syntactic representations as input and directly transform it into a grammatically correct sentence.

Using the annotation from the RPST, it can be directly mapped to a DSynT. For illustrating this procedure, consider the first activity from the example process (Figure 2.4). Using the action *take down* as the main verb, the role *room-service manager* as subject and the business object *order* as object, we can derive the DSynT depicted in Figure 2.8 representing the sentence *The room-service manager takes down the order*.

2.2.2Techniques for Translating Machine Artifacts

There are three main techniques for translating machine artifacts to text (*e.g.*, translate a business process model to a human-readable text in natural language): non-language techniques, mail-merge techniques and language techniques [20].

The simplest techniques, normally termed *non-language techniques*, are based on canonical texts or predefined templates. In the *canonical text* technique, the input data is directly mapped to a predefined sentence. For example, a system translating data on climate prediction to a text in natural language could use the sentence "The weather will be good today." to represent a warm and sunny day. The *predefined template* technique inserts some extra information in the predefined sentence. For example, in the template "Today there is an X% probability of rain", the X could be replaced by the probability of precipitation retrieved from a database [62]. These types of techniques are not considered truly language, since manipulation of the text is based on manipulating character within a sentence previously defined [104].

The *mail-merge* techniques are more complex than the *non-language* techniques. They are found in Microsoft Word and other popular text editors. In this case, there are techniques that insert data into predefined spaces in a standard document, as well as techniques which are essentially programming languages allowing the output text to vary arbitrarily, according to the input data [20]. As an example, consider a Word document that has its output according to the content of the file used as a data source (*e.g.*, an Excel spreadsheet). Each predefined space, like `CONTACT_FIRSTNAME` and `CONTACT_LASTNAME` will be replaced by the respective value defined in the file used as input. Suppose that the input file is some Excel sheet with the following columns: FirstName, LastName etc. Through the use of *mail-merge* techniques, the data of each column can be mapped to the predefined spaces in the Word document, according to the mapping defined by the user (*e.g.*, FirstName column corresponds/maps to `CONTACT_FIRSTNAME`).

Language techniques, or real generation of natural language, use intermediate structures to represent the text in more details. These structures normally specify the main *lexemes* for each sentence. Lexeme is a set of words with the same morphological class that are distributed in a complementary form and differ morphological from each other only by flexion and suffixes. The words that make up a lexeme are called inflections of lexeme [73]. For example, the words *singer* and *sing* do not make up a lexeme because they do not belong to the same morphological class, noun and verb, respectively. On the other hand, the words *singer* and *singers* compose a lexeme because they belong to the same morphological class (nouns), and they differ only by suffixes.

The core of *language* techniques, is the use of an intermediate structure to store the messages before they are transformed into natural language sentences. The advantage of this procedure is the significant gain in maintenance and flexibility. In a system based on templates, each template must be manually modified when a different output text is required.

In a language-based approach, the generated text can be changed by setting a different value to a parameter in the intermediate structure. For example, the sentence “The train will depart soon” can be easily transformed into “The train is departing now” by modifying the main verb time of the intermediate structure [62].

Approaches based on templates and canonical text are not considered as good as linguistic approaches, in maintenance terms, quality of the output text and variation of the text. Systems using templates based approaches are more difficult to maintain and modify, and produce outputs with inferior quality than NLG (Natural Language Generation) systems [20]. Systems using a non-language approach are not able to incorporate generic

language aspects [11].

On the other hand, approaches based on templates and canonical text also have advantages. Systems based on templates require less time to be developed and do not need to be fully complete for a correct execution. Besides, they may be extended by inserting new templates. Another advantage is the flexible adaptation to a new domain. When applying a new domain to the system based on templates, many of the templates should be rewritten. However, the mechanism used to generate the text, will require little or no modification. The fact that templates can be manually specified is advantageous over NLG only when good linguistic rules are not yet available or have very specific conditions for their use [126].

As a consequence, Reiter and Mellish propose a cost-benefit analysis, seeking an approach that makes use of advantages and eliminates disadvantages of other works [35, 106]. As a result, many systems for natural language generation use hybrid approaches where linguistic techniques are combined with canonical texts and templates [107].

2.2.3Necessary Steps to Develop Natural Language Generation Systems

This section presents two sets of steps (*i.e.*, processes) used to develop systems that generates natural language from machine artifacts.

2.2.3.1Pipeline Architecture

Many natural language generation systems follow a pipeline approach (Figure 2.9) consisting of three main steps described as follows [27, 20]:

- **`Text Planning`**: This task converts the system input to the specific kind of data objects (or "messages") that serves as a basis for the subsequent generation tasks. Furthermore, it is specified in which order this information will be conveyed and the structure of the output text. The result is a text plan, often in the form of a tree structure representing the order and grouping of the messages, and the relations between them.
- **`Sentence Planning`**: This task chooses the right words to express the input information. Often, a concept can be expressed using different words or phrases. So, in the sentence planning, the lexicalization procedure is used to choose the most appropriate way in the given context to express words or phrases. If applicable, messages are aggregated and pronouns are introduced in order to obtain variety. The sentence planning also can use the *Aggregation* procedure to decide which informa-

Figure 2.9: NLG pipeline approach, defined by Reiter and Dale.

tion to put in one sentence. Pieces of information that form separate input messages may be joined together and expressed using one sentence which, for instance, contains a conjunction or a relative clause. *Referring expression generation* can also be used to create phrases to identify domain entities. This involves choosing the type of expression (*e.g.*, a pronoun or a definite description) and in the case of definite descriptions, choosing the properties to include in the description.

- **`Sentence Realization`**: This task creates grammatical sentences. This involves the application of syntactic and morphological rules that determine aspects like word order and agreement.

In general, the text planning tasks (step 1) are language-independent but domain-specific, whereas sentence realization (step 3) is language-specific, but can in principle be done in a domain independent fashion. This difference in domain-dependence of the different tasks largely explains why reusable software packages exist for linguistic realization, but not for document planning. The tasks associated with sentence planning (step 2) require both domain and language-specific knowledge. For instance, the generation of referring expressions requires knowledge about the application domain (*e.g.*, information about domain objects and their properties) and about the application language (*e.g.*, the syntactic properties of modifiers expressing certain properties).

The first and second step of the pipeline can be further decomposed into several micro steps as follows (Figure 2.10).

- **`Text Planning`**: The text planning step can be decomposed into three steps. The first step (*Linguist Information Extraction*) is responsible for extracting the linguist information (*e.g.*, words of labels) from the machine artifact received as input by the system (*e.g.*, a business process model written using BPMN (Business Process Model Notation) [50]). In particular, the information is extracted and organized in a graph to maintain the information's dependencies and hierarchy. The second step (*Annotated RPST Generation*) is responsible for storing the linguistic information in RPST (Refined Process Structure Tree, section 2.2.1) tree nodes [131]. In particular, it reads the graph structure created in the previously step and transforms it in a RPST tree. Finally, the third step (*Text Structuring*) is responsible for adding structuring

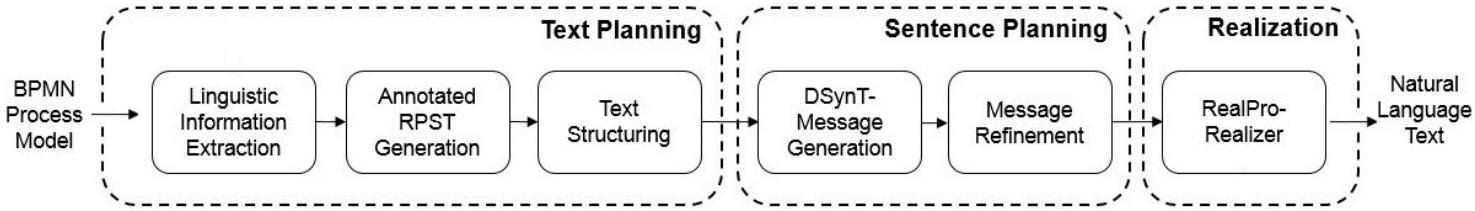


Figure 2.10: Extended pipeline approach.

data inside the nodes. For example, the addition of paragraphs, text indentation and markers, allowing the text to be shown in a fashion format and improving the text quality and readability. Basically, the RPST tree nodes are traversed and their structuring properties are set. Furthermore, it is specified in which order this information will be conveyed.

- **`Sentence Planning`**: The sentence planning phase can be decomposed into two steps. The first step (*DSynT Message Generation*) transforms the previously created RPST tree into a list of intermediate messages. In this step, the linguistic information of the model is not directed mapped to the final text, rather to a conceptual representation that is still suitable for changes. In particular, each sentence is stored into a DSynt tree. Afterwards, the second step (*Message Refinement*) is triggered. This step is needed to perform the message aggregation task (*e.g.*, aggregate correlated sentences into one bigger message separated by coma).

2.2.3.2Data-to-Text Architecture

The pipeline proposed by Reiter and Dale [20] is suitable for most NLG systems. Nevertheless, there are systems that produces text from non-linguistic input data. In this case, the linguistic information extraction executed during the first step (*Text Planning*) of the pipeline process (Figure 2.9) is not required. These systems are called data-to-text systems. The information needed have to be extracted from raw data, which is typically numerical, *e.g.*, sensor data, graphics and event logs. The data is extracted through data analysis aligned with linguistic processing. To address such challenge, Reiter and Dale proposed a new architecture (*data-to-text architecture*), which can be considered as an extended version of the pipeline architecture [105]. The extended architecture is suitable for many data-to-text systems and is being used, *e.g.*, in the BabyTalk project which employ NLG techniques to provide decision support in a Neonatal Intensive Care Unit (NICU) [36].

As patient care standards improve, the demand for continuous monitoring and data collection is on the increase in these units. Therefore, medical staff need to process large

quantities of information in order to ensure that clinical decisions are maximally beneficial to an infant. The systems developed by the BabyTalk project aim to reduce this information overload through the use of NLG techniques. Moreover, they target different user groups, namely nurses, doctors and family members or friends. These groups have different information requirements and may also have different levels of expertise. The main source of data is non-linguistic (raw data), through graphics and tables.

Perhaps the biggest difference between data-to-text systems and NLG systems whose input is a knowledge base is that data-to-text systems receive raw data (non-linguistic information) and must analyze and interpret their input data, as well as decide how to linguistically communicate it. For example, it must interpret a heart rate of 80 (*e.g.*, through image processing) per minute and decide weather it is within an acceptable frame and communicate with a natural language sentence as "*The patient heart rate is within an acceptable frame*".

The data-to-text architecture is composed by a 4-stage pipeline (Figure 2.11) which are described as follows.

1. ***Signal Analysis***: This step analyzes numerical and other input data, looking for patterns and trends.
2. ***Data Interpretation***: This step identifies more complex (and domain-specific) messages from the patterns and trends detected in Signal Analysis and also identifies causal and other relations between messages.
3. ***Document Planning***: This step decides which of the above messages should be mentioned in the generated text, and creates a document and theoretical structure around these messages.
4. ***Micro-planning and Realization***: This step creates the text that communicates the document plan.

2.2.4NLG Steps in the context of BPMN text generation

As described in Section 2.2.3.1, there is a set of steps for the automatic generation of text from process models. Table 2.1 shows a summarized view of these steps and some authors that published papers about the specific difficulties related to it. This work considers these steps which are based on the three-step approach [27], consisting of activities related to automatic text generation, which are presented as follows.

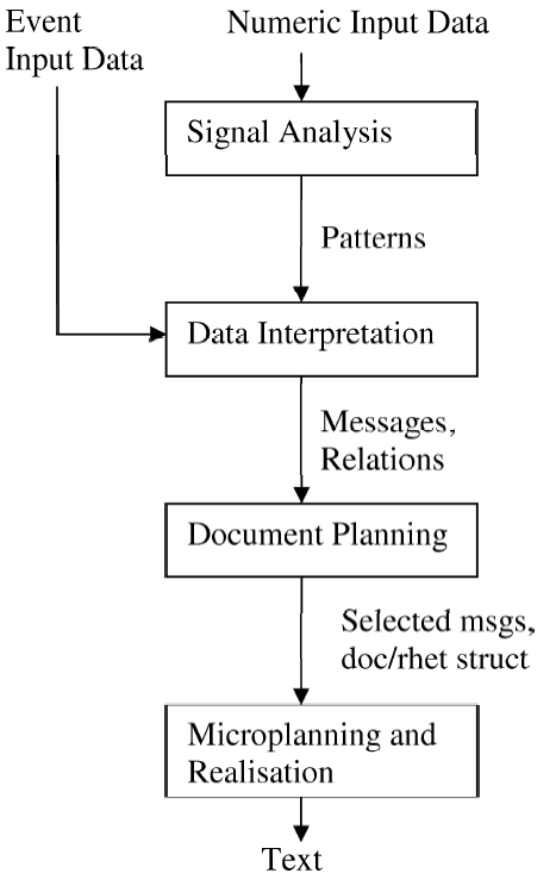


Figure 2.11: Extended pipeline (data-to-text) architecture.

Table 2.1: Steps to the automatic text generation (adapted from Leopold [62])

Macro and substeps of NLG approach
1 - Text Planning
a. Extraction of linguistics information [95].
b. Linearization of the model [131].
c. Structuring the text [83].
2 - Sentence Planning
a. Lexicalization [21].
b. Messages refinement [49]
3 - Message Realization [59, 11]

Regarding **Text Planning**, there are three great steps related to the activities involved.

- **Extract linguistic information from elements in the process model.** For example, the activity *Receive order* has to be divided automatically into action *receive* and in business object *order*. Without this separation, it would not be clear which of the two words define the verb representing the action of the activity. The analysis of labels (or names) of process models becomes even more complicated due to the small size of process models' labels [26].
- **Linearization of process models into a sequence of sentences.** Process models rarely consist of only one sequence of tasks. They also include parallel branches and decision points.
- **Structure and format text.** Natural language text are structured in order to facilitate reading. Some structures, such as paragraphs and markers, must also be present in the natural language representation of the process model.

The **Sentence Planning** step encompasses the tasks of lexicalization and refinement of messages and has the following challenges:

- **The appearance of lexicalization refers to the mapping of BPMN process models to specific words.** In this case, the challenge is the need for integration of language information extracted from elements of process models with control structures, division and bonding so that the process would be described in an easily and understandable manner.

- **The aspect of message refinement refers to the construction of the text.** The related challenge involves the aggregation of messages, as the introduction of referring expressions (for example, pronouns) and also the insertion of discourse markers, such as, thereafter and subsequently. In order to consolidate the messages in a proper way, the choice of aggregation have to be first identified and then decide which can be applied to improve the quality of the text. The introduction of terms of reference requires the automatic recognition of entity types. For example, the actor *kitchen* should be referenced by the pronoun *it* while the actor *waiter* should be referenced by the pronoun *he*. The insertion of discourse markers[2] should increase the readability and the variety of text. Examples of these markers are: "*afterwards*", "*As a consequence*", "*In parallel*".

In the context of **Messages Realization**, the activity to generate grammatically correct sentences is performed and it includes, among other tasks:

- Determination of an appropriate order of words;
- Classification of the gender;
- Introduction of articles;
- Transformation of the words in upper-case or lower-case according to the context in which they operate (*e.g.*, beginning of a sentence, noun etc.).

In addition to the core activities of natural language generation, flexibility is also an important resource. As we do not expect the input models to be defined according to a specific agreement, we have to deal with different characteristics of each artifact used as input. Diverse scenarios should be covered with different implications for text output. For example, if a model uses lanes and consequently provides a description of the actors, the sentence can be represented in the active voice, *e.g.*, *The manager checks the system*. On the other hand, if actors are not defined for the specified task, the description should come in passive voice, *e.g.*, *The system is verified*.

2.3 NLP

Methods in the area of Computational Linguistics and Natural Language Processing, which are a branch of artificial intelligence, try to analyze and extract useful information

[2]Discourse markers, in this context, are words or expressions used for explicitly correlate text's sentences.

from natural language texts or speech. Therefore, it is concerned, with the recognition and synthesis of speech in natural languages (*e.g.*, English) [48]. An exemplary area of application is sentiment analysis, where the goal is to automatically determine the attitude or opinion towards a product or company from online articles [92].

The field of Natural Language Processing (NLP) aims to convert human language into a formal representation that is easy for computers to manipulate, *i.e.*, the ultimate goal of research on Natural Language Processing is to parse and understand language. Automatic text, or document, retrieval has recently become a topic of interest for those working in NLP. Current end applications include information extraction, machine translation, summarization, search and human-computer interfaces. Researchers have taken a divide and conquer approach and identified several sub-tasks useful for application development and analysis. These range from the syntactic, such as part-of-speech tagging, chunking and parsing, to the semantic, such as wordsense disambiguation, semantic-role labeling (named entity extraction) and anaphora resolution. These techniques can be used altogether or combined within a specific set to enable natural language processing of texts and documents [71, 97].

This section presents NLP concepts, which are specific to the development of algorithms that can process and parse natural language texts to gather relevant data. The techniques described were essential to the development of the round-trip approach described in details in Section 3. As mentioned earlier in Section 2.2, this master thesis presents a framework that can generate natural language text from business process models through NLG. Based on the generated text, it is possible to navigate in the opposite direction. In other words, to generate a business process model from the natural language text. In the context of this master thesis, NLP techniques were implemented by the framework to enable the extraction and parsing of natural language texts, in order to gather business process data and present this data through a machine artifact format (*e.g.*, BPMN process model). In total, four standard NLP tasks are of vital importance to achieve this goal and will be described in more detail in the upcoming subsections. In Section 3.3, these techniques are referenced in the context of business process, explaining the specific strategies used along with each technique to extract the information from the business process descriptions.

2.3.1Syntax Parsing: Part-Of-Speech Tagging (POS)

NLP has on intermediate tasks that make sense of some of the structure inherent in language without requiring complete TAGGING understanding. One such task is *part-of-speech tagging*, or simply *tugging*. Tagging is the task of labeling (or tagging) each word in

a sentence with its appropriate part of speech. It aims at labeling each word with a unique tag that indicates its syntactic role, *e.g.*, plural noun, adverb. For illustration purpose, consider the sentence "*The representative put chairs on the table.*". After applying POS technique, we would have the following tags for the words: The(AT); representative(NN); put(VBD); chairs(NNS); on(IN); the(AT); table(NN).. The part-of-speech tags assigned to each word are described in table 2.2 and follows the Brown/Penn tag sets [72].

Table 2.2: Some part-of-speech tags frequently used for tagging English

TAG -> PART-OF-SPEECH
AT -> article
BEZ -> the word is
IN -> preposition
JJ -> adjective
JJR -> comparative adjective
MD -> modal
NN -> singular or mass noun
NNP -> singular proper noun
NNS -> plural noun
PERIOD -> .:?!
PN -> personal pronoun
RB -> adverb
RBR -> comparative adverb
TO -> the word to
VB -> verb, base form
VBD -> verb, past tense
VBG -> verb, present particple, Gerund
VBN -> verb, past participle
VBP -> verb non-3rd person singular present
VBZ -> verb, 3rd singular present
WDT -> wh- determiner (what, which)

2.3.2Tokenization

Normally, an early step of processing is to divide the input text into units TOKENS called tokens where each is either a word or something else like a number or a punctuation mark. This process is referred to as *tokenization*. The main strategy to split sentences into an array of tokens is to use the occurrence of white-space (a space or tab or the beginning of a new line between words) as a delimiter to distinguish words within the sentence. But

even this signal is not necessarily reliable because words are not always surrounded by white space. Often punctuation marks attach to words, such as commas, semicolons, and periods (full stops) [72].

At first, it seems easy to remove punctuation marks from word tokens, but the treatment of punctuation may vary greatly depending on what we want to extract from the natural language text used as input. For instance, depending on the primary objective, it may be important to keep sentence boundaries [72]. In other cases, sentence-internal punctuation can be just stripped out. Nevertheless, recent work has emphasized the information contained in all punctuation [65]. No matter how imperfect a representation is, punctuation marks like commas and dashes give some clues about the macro structure of the text. For example, punctuation marks like period are often end of sentence punctuation marks but they also can mark an abbreviation such as in ”etc.”. These abbreviation periods presumably should remain as part of the word, and in some cases keeping them might be important so that, *e.g.*, we can distinguish “Wash.”, an abbreviation for the state of “Washington”, from the capitalized form of the verb “Wash”. Note especially that when an abbreviation like “*etc.*” appears at the end of the sentence, then only one period occurs, but it serves both functions of the period, simultaneously. Besides periods, there are also other challenges while defining the best strategy for the Tokenization phase (*e.g.*, single apostrophes, hyphenation, homograph, word segmentation, etc.) [71, 72].

2.3.3Stop Words Removal

Sometimes, some extremely common words which would appear to be of little value in helping select documents matching a user need are excluded from the vocabulary entirely. These words are called *stop words*, which are filtered out before or after processing of natural language data (text). Any group of words can be chosen as the stop words for a given purpose. The general strategy for determining a stop list is to sort the terms by collection frequency (the total number of times each term appears in the document collection), and then to take the most frequent terms, often hand-filtered for their semantic content relative to the domain of the documents being indexed, as a stop list, the members of which are then discarded during indexing. Using a stop list significantly reduces the number of postings that a system has to store. The reason why stop words are critical to many applications is that, if we remove the words that are very commonly used in a given language, we can focus on the important words instead. Keyword searches for terms like *the* and *by* do not seem very useful. However, this is not true for phrase searches. The phrase query “*President of the United States*”, which contains two stop words, is more precise than President AND “*United States*”. The meaning of flights to London is likely

to be lost if the word to is stopped out [71, 72].

2.3.4Anaphora Resolution

Another problem which has to be addressed to produce conceptual models from text is the resolution of anaphoric references. Anaphoras include possessive pronouns (*e.g.*, "*my*", "*your*", "*her*"), personal pronouns (*e.g.*, "*I*", "*you*", "*she*"), certain determiners ("*this*", "*that*"), relative pronouns ("*who*", "*which*") or phrases describing the object under investigation with different expressions (*e.g.*, "*Steve Jobs*", the "*CEO of Apple*"). A simple algorithm for the resolution of pronouns is proposed by JR Hobbs [46]. It was later extended by Niyu Ge *et al*. [37].

Jurafsky and Martin also mention further possibilities of restricting the selection process through the usage of parallelisms, verb semantics, and selectional restrictions [48]. Examples for implementations of anaphoric reference resolution algorithms are the GuiTAR Framework [94], BART8 [133] or the Reconcile framework [122]. Although these libraries are written in Java, they require a special XML-Format as input and are not seamlessly usable with the output provided by our framework.

Section 3.3 describes how anaphoric references are tackled within the context of business process model generation.

2.3.5NLP Steps in the context of BPMN text generation

Mapping natural language text to process models can be assigned to four broad categories: syntactic leeway; atomicity; relevance; and, referencing. Table 2.3 gives an overview of the difficulties (bold font) and the respective approaches addressing them (standard font).

Table 2.3: Steps for Mapping Natural Language Text to Process Models.

Challenges
1 - Syntactic Leeway
a. Active-Passive [1].
b. Rewording/Order [142, 31].
c. Implicit Conditions [39, 40].
2 - Atomicity
a. Complex Sentences [31, 66].
b. Action Split over Sentences [118]
c. Relative Clauses [66]
3 - Relevance
a. Relative Clause Importance [66].
b. Example Sentences [52]
c. Meta-Sentences [66]
4 - Referencing
a. Anaphora [118, 23].
b. Textual Links [30]
c. End-of-block Recognition [66, 52]

Syntactic Leeway relates to the mismatch between the semantic and syntactic layer of a text. For example, the grammatical role of an actor differs for active and passive sentences. In active sentences, such as "*The clerk enters the data*", the actor *clerk* is a subject. In passive sentences, by contrast, the actor is often only mentioned in the prepositional phrase. As an example, consider the passive version of the former sentence "*The data is entered by a clerk* ". Another problem relates to implicit conditions as in the sentence "*For new patients, a patient file is created*". This sentence implies that a new file is only created for new patients. However, the automated detection of such implicit conditions is not trivial.

Atomicity deals with the question of which sentences correspond to process model activities and which sentences provide additional information. Hence, automated techniques must, for instance, detect whether a sentence represents an activity or rather a meta-description of the control flow. In addition, a techniques must take into consideration that a single activity may correspond to multiple sentences and vice-versa.

Relevance relates to the challenge of detecting whether a sentence is relevant for the corresponding process model or not. The problem is that many natural language texts contain example sentences for illustrating the discourse. As an example, consider the sentence "*The edit function can be used to correct errors*".

Finally, *Referencing* addresses the question of how to resolve relative references between words and between sentences. The challenge is to correctly associate personal pronouns such as "*he*" or "*she*" to the corresponding entity in former sentences. Also determiners such as "*that*" or "*this*" need to be adequately resolved.

2.4 Chapter Summary

This chapter presented three main concepts which the framework proposed by this master thesis is build on. These concepts are essential to understand how the framework works. It implements a round-trip approach, described in the next chapter, which cycles between the generation of natural language text from BPMN process models and the generation of BPMN process models through natural language texts. In particular, NLP techniques (Section 2.3) were used to generate BPMN process models (section 2.1) from the natural language text and NLG techniques (Section 2.2.2) were used to generate natural language text from BPMN process models. As described earlier in this chapter, several steps (refer to 2.2.4 and 2.3.5) had to be addressed to instantiate the round-trip approach.

3. The Round Trip Approach

This chapter presents the `round-trip` technique besides its implementation. The technique was implemented in a language-independent framework, through the development of algorithms capable of generating business process models using natural language text as input and the integration of algorithms responsible for generating natural language text from business process model. The result is a framework with features that enable the synchronization between changes made to text back to the original process model and changes in the model to reflect in the corresponding text. Hence achieving the desired goal of maintaining both business process representations (models and textual descriptions) automatically synchronized. The risk of flaws regarding the information's consistency in the manual process is also mitigated by the proposed approach.

Figure 3.1 gives an overview of the round-trip technique. The links between elements and sentences are previously defined through pattern matching and stored for further querying by both, Process Model to Natural Language (labelled as "*P*") and Natural Language to Process Model (labelled as "*N*"). The pattern matching rules are defined in a template stored by the framework. Since it is a round-trip, the technique can be triggered from different scenarios. These use-case scenarios are better described below:

- **Generate Text from Process Model:** The first scenario is the natural language generation from a given process model. The framework receives the model as input, triggers the Process Model to Natural Language which queries the linkage component and assemble a whole text from the natural language sentences.
- **Generate Model from Process Textual Descriptions:** This scenario is represented by the generation of a process model from a given process textual description. The framework receives the text as input, triggers the Natural Language to Process Model which queries the linkage component and assemble a whole model from the mapped elements.

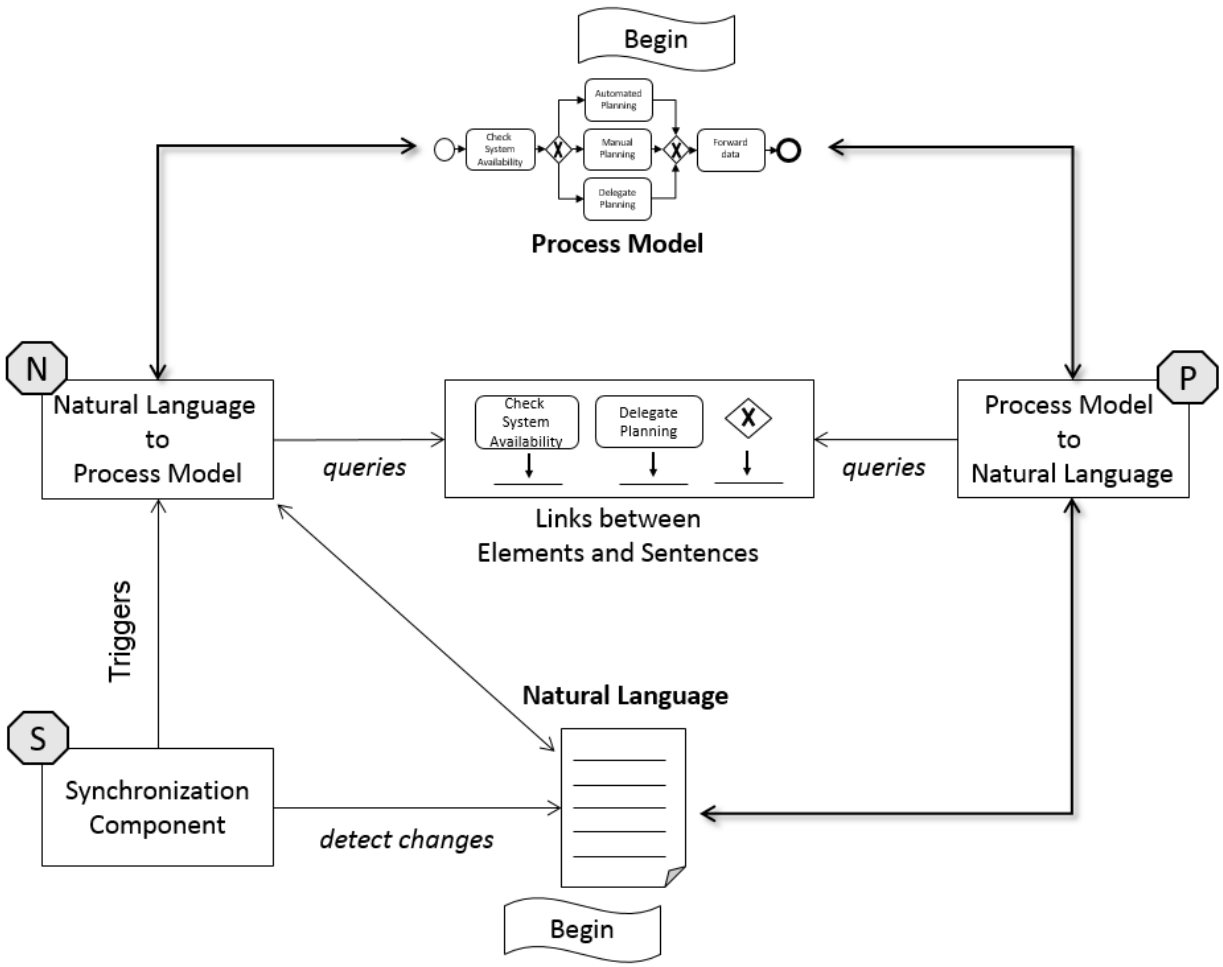


Figure 3.1: Illustration of the round-trip approach.

- **Update a Model from Text based changes:** This scenario is triggered when a process textual description is manually updated. The framework's Synchronization Component (labelled as "*S*") detects these changes and reflect them to a process model. This is done by triggering the Natural Language to Process Model and thus generating a updated version of the original model. This scenarios only reflect the changes, updating only the process model elements that had changes.

- **Update a Text from Model based changes:** The text update is treated as a new text generation scenario. In other words, the whole model is submitted as input and the whole text is regenerated. The decision to generate the whole text again instead of updating only sentences which had changes, was based on cost-benefit analysis which considered the performance gain and development effort.

- **Round-Trip:** This scenario is represented by a generation cycle. For illustration purpose, consider the process model as starting point. The model is submitted as the framework input and the text is generated. Then, the text is manually changed and the original process model is automatically updated. In the same cycle, the updated model is submitted again to assert whether it matches the text description. At this point, the user can go in both directions, generate text from model or model from the text.

Section 3.1 describes the framework architecture, packages and the main classes. Section 3.2 describes the steps and algorithms implemented to generate natural language text from business process models used as input, through a specialized NLG pipeline. Section 3.3 describes in details the generation of business process models using natural language text as input, through the definition and implementation of a specialized pipeline to deal with the NLP process. Section 3.4 details the algorithms responsible for the integration between the NLG and NLP pipelines described in Sections 3.2 and 3.3, enabling the implementation of the `round-trip` approach. Section 3.5 illustrates the framework usage through two UML sequence diagrams. These diagrams show how the components and classes interacts to deliver the process output in text or model format. Finally, Section 3.6 presents the summary for this chapter.

3.1Generic Framework for Natural Language Text

In prior work, a tool capable of generating natural language texts from BPMN process models was presented [110]. The tool implements the NLG architecture proposed by Reiter and Dale (Section 2, Figure 2.10) and makes use of an extended version of the original

NLG pipeline process (Section 2, Figure 2.9). In the present research, the formalism of this tool is further improved towards a generic infrastructure (*i.e.*, framework) that allows the addition of new languages belonging to the Romanian and Germanic sub-branches of the Indo-European language family. Moreover, the logic to handle Portuguese process models was implemented in order to evaluate the framework language-independent infrastructure. Hence, new algorithms have been developed to handle and map the Portuguese grammar (semantic and syntactic aspects). The framework may significantly increase the audience of process models as an understanding of process models is no longer bound to the knowledge of a specific notation.

Although English is the predominant business language, several companies typically model their processes in native language, partially driven by legal requirements [61]. As a consequence, any company that model their process using other language, rather than its native, would not be able to benefit from the framework features. Another drawback is the complexity of the framework implementation. Natural language generation is a complex subject and, in order to use our technique, it would be necessary to understand all the NLG and NLP pipeline steps. This could take some time for people without the required knowledge on this particular field. To overcome this issue, the framework was designed based on generalization of language-specific components, making them generic to support multiple languages. It hides the language-independent logic and provides a public and generic infrastructure for the language-specific logic. In other words, our implementation encapsulates algorithms that are exactly the same for any language, and provides generic infrastructure that guides the implementation of algorithms that deal with the syntactic and semantic rules of a given language. This infrastructure defines all the contracts and operations needed in order to read and produce language-specific texts from business process models. Thus, making it easier to add support to new languages.

The technique generalization made it possible to abstract a lot of complex and common aspects from NLG and NLP pipeline steps, such as, the planning phase and sentence planning phase. These phases are totally encapsulated by the framework, reducing complexity of implementation to handle new languages, and maintaining the focus into the language specific operations.

3.1.1The Framework Architecture

According to Pree, the proposed framework can be classified as an application framework. "*Application frameworks consist of ready-to-use and semi-finished building blocks. The overall architecture is predefined as well. Producing specific applications usually*

means to adjust building blocks to specific needs by overriding some methods in subclasses" [96]. This framework is composed by two main components: The NLG core component and the NLP core component. Both components are based on a languange-independ module, which permits the addition of multiple languages without altering the two core components.

The NLG core component (Figure 3.2) is composed by several ready-to-use building blocks (known as Frozen spots) and defines interfaces which must be implemented to support specific languages. Each interface represents a hot spot, because they are flexible to satisfy specific needs (in our case, generate text in a specific language). The architecture's frozen spots are represented by classes, while the hot spots are represented by interfaces (elements stereotyped as *«interface»*) [96].

The `GeneralLanguageCommon` package (Figure 3.2) is the generic (language-independent) module. It includes the interfaces definitions, which must be implemented for a specific language in order to generate natural language text, *i.e.*, it is the Natural Language Generation core. It contains the necessary infrastructure to work with the NLG and NLP pipeline process. It includes the data structures, and it knows exactly which and when an object must be called to deal with a specific phase of the pipeline. For example, regarding the *Localization strategy* (represented by the classes of the `Localization` package), the module knows when to call the `LocalizationManager` object to translate a specific message, retrieved from the `LocalizationMessages` enumeration (keys) during the text information extraction. *E.g.*, for the key *PROCESS_BEGIN_WHEN*, the returned text would be "O processo começa quando" for Portuguese. Analogously, it knows when to trigger each interface method implemented for a given language at runtime.

The `GeneralLanguageCommon` is composed by several sub-packages, which are defined as follows.

- **`Fragments`**: represents sentences in natural language patterns. The classes defined in this package are frozen spots.
- **`DSynt`**: maps the information from the process into DSynT trees. Classes defined in this package are frozen spots.
- **`Localization`**: defines the logic needed to access specific language dictionaries. Besides, it has the common functionality for fetching the translation of a given word. For example, the `LocalizationManager` class is used to fetch messages from the dictionary, which will be used in the final text representation.

- **`LanguageRealizer`**: corresponds to packages containing the classes for the concrete implementation of interfaces and abstract classes defined in `GeneralLanguageCommon` package for each language (*e.g.*, `EnglishRealizer` and `PortugueseRealizer` depicted in Figure 3.3).

- **`LanguageConfig`** in the **`MultiLanguageProject`** package: is a Factory[1] that creates objects from the classes that implement language-specific logic interfaces for a given language (*e.g.*, Portuguese or English).

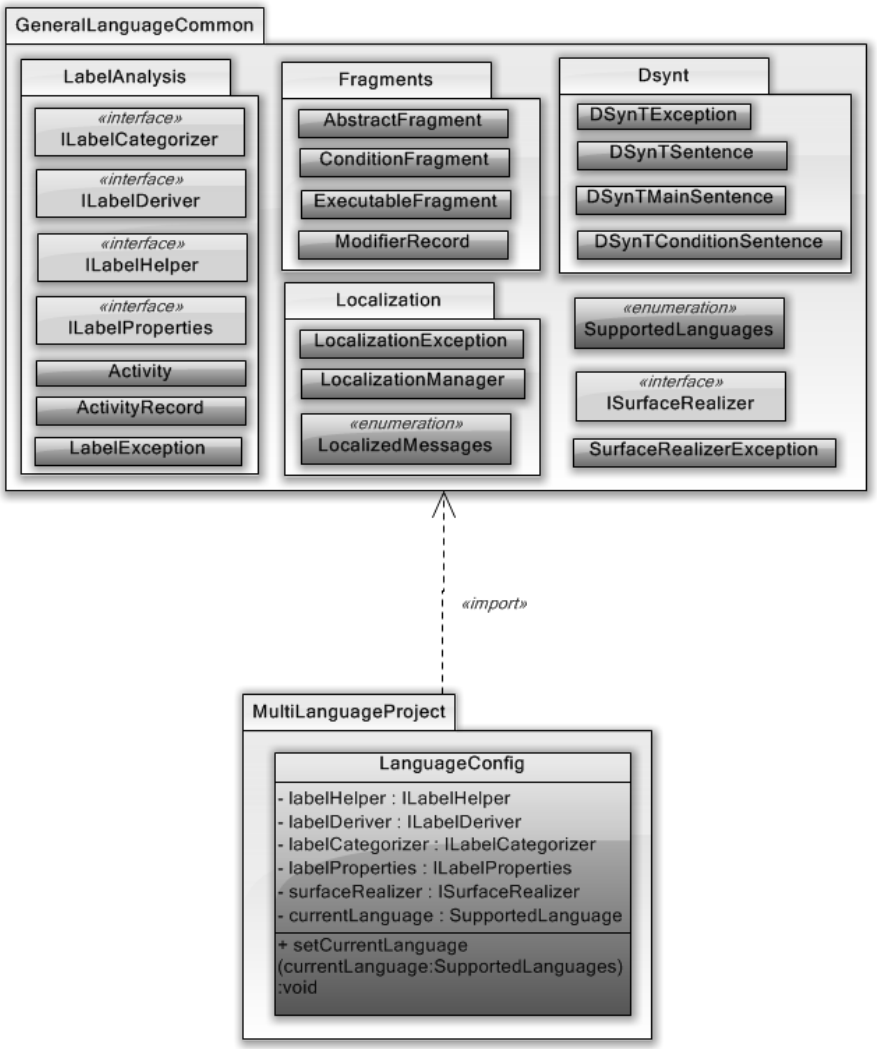


Figure 3.2: NLG Core Architecture - UML Package diagram.

Figure 3.3 presents a package diagram of the hot spots implementations for Portuguese and English. They are named as *Realizer* since they realize the implementations of `GeneralLanguageCommon` package interfaces. Each language has its own specific implementation (*e.g.*, `PortugueseLabelHelper` class implements `ILabelHelper`).

[1] A factory is a program component which main responsibility is the creation of other objects.

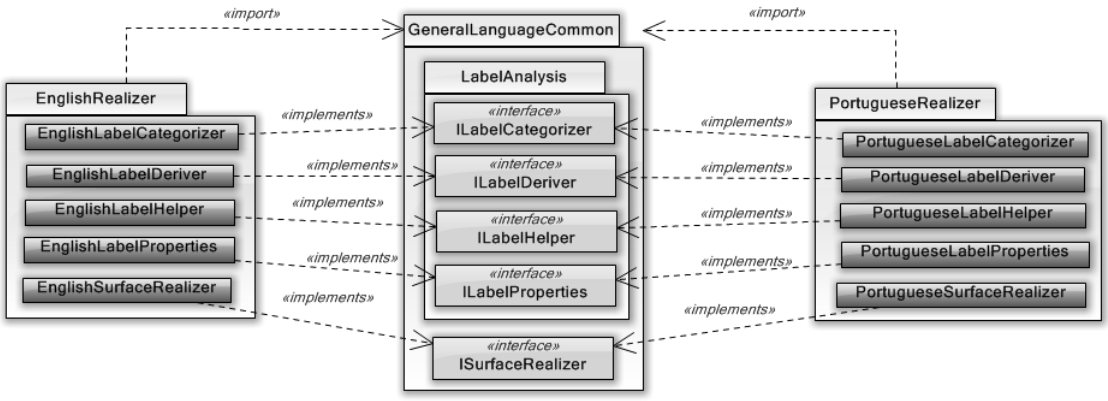


Figure 3.3: Implementation of the hot spots defined by the architecture.

So, `PortugueseRealizer` and `EnglishRealizer` classes implement hot spots and use frozen spots to accomplish necessary tasks.

The `NlpCore` package (Figure 3.4) is the generic (language-independent) NLP module. It includes the interfaces definitions, which must be implemented for a specific language in order to generate process models from natural language text. In other words, it is the `Natural Language Processing (NLP) core`. It contains the necessary infrastructure to work with the NLP pipeline process described in Section 3.3. It includes the data structures, and it knows exactly which and when an object must be called to deal with a specific phase of the pipeline. For instance, regarding the *Text Cleaning Strategy*, the module knows when to call the correct `ITextFormatter`'s implementation object to format a specific text, received as input. For example, for texts which follows a given template, the returned object would be an implementation of `TemplateTextFormatter`. Analogously, it knows when to trigger each interface method implemented for a given text format at runtime.

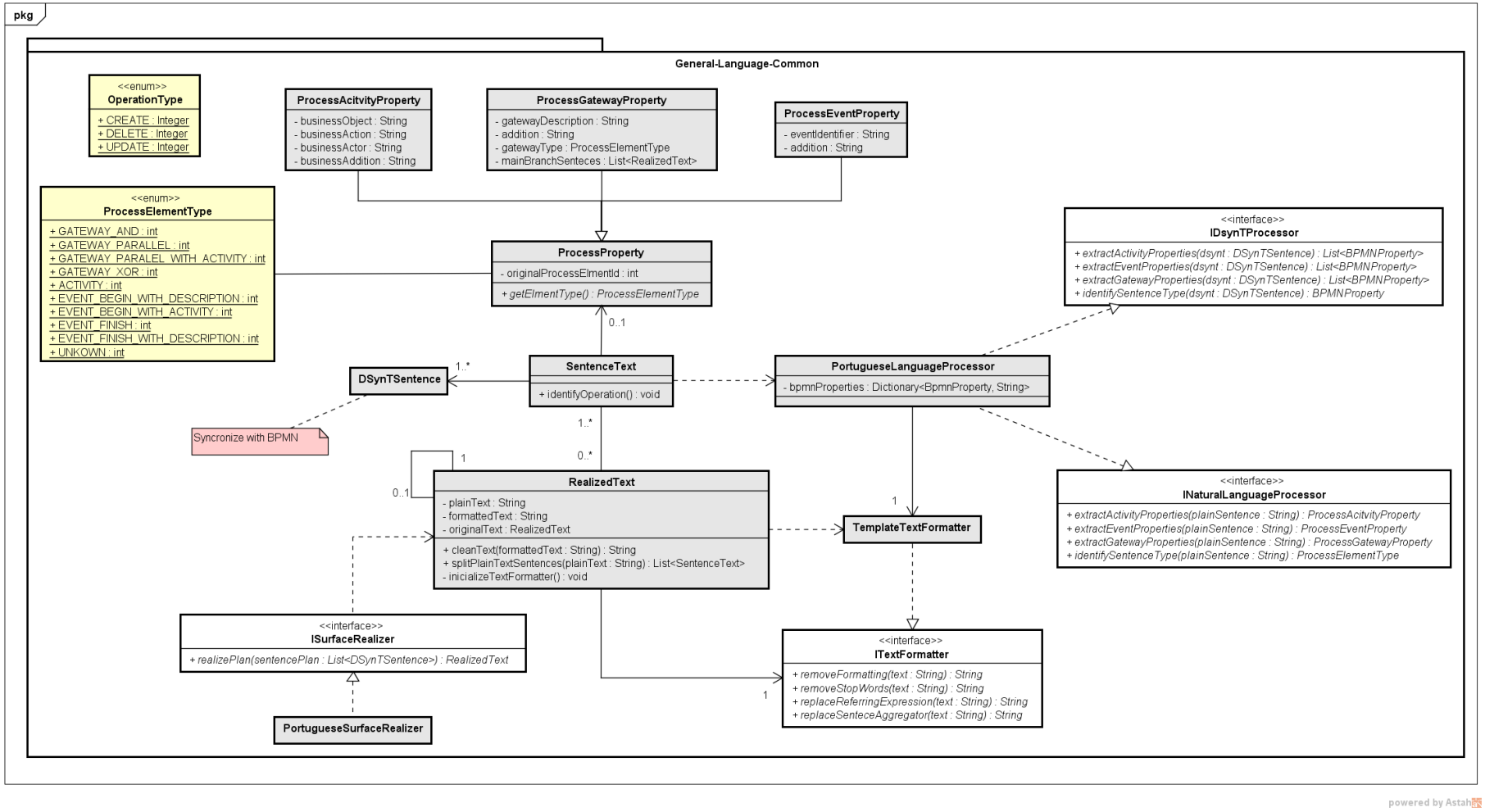


Figure 3.4: NLP Core Architecture - UML Class diagram.

With the definition of the NLP core module, the developer does not need to know in details how the NLP process works and the developer is free to choose weather he will use the self-package NLP algorithms or if he prefers to develop new algorithms to best fit his needs. The module assures that all the business process properties will be correctly extracted from the natural language text, as long as interfaces and their methods are implemented according to the specification. The components of this model are described as follows:

- `ITextFormatter:` The objective of this interface is to remove the stop words (*i.e.*, words and phrases that must be filtered out before the processing of natural language text) and to remove any special character used for formatting purpose (*e.g.*, underscore, excessive white spaces, bullets etc.). It is also responsible for undoing aggregation tasks. For example, consider the following text fragment: "- *The customer pays with credit card. Afterwards, he signs the retrieval form.*" would be formatted to "*Customer pays with credit card. Customer signs retrieval form.*". In this case, the stop words "*Afterwards*", "*The*" and the formatting character "-" were removed from the text fragment. Note that the word "*he*" was replaced by the actor's name "*Customer*" due to the undoing of referring expression generation (Section 3.2.2).

- `INaturalLanguageProcessor:` This interface defines the necessary methods which are called to identify the words (or plain sentences) that represents relevant business process data. It is also responsible for extracting all the process model elements properties according to its respective type (*e.g.*, gateway, event or activity). For instance, the implementation for *identifySentenceType* method must be capable of mapping specific language patterns (*e.g.*, IF <condition> THEN, <activity>) to specific process model elements (*e.g.*, Gateway XOR). After identifying the element type, it calls the extraction method (*e.g.*, *extractGatewayProperties*) to store its data. For illustration purpose, the following sentence should be mapped to a XOR Gateway accompanied by an activity: "*If payment is authorized, then the Financial Department notifies the payment.*". In this case, the Gateway label is represented by the clause "*payment is authorized*" and the activity by the clause "*Financial Department notifies the payment*".

- **`RealizedText`**: This class is initialized with the natural language text and starts the processing to generate a plain text, without any formatting and stop words. The result is a list of several plain sentences (without any formatting or stop words).

- **`TemplateTextFormatter`**: Concrete implementation of the *ITextFormatter*

interface. This implementation is specific to Template descriptions that were automatically generated from a BPMN model. The current supported template is presented in Table 3.1.

- **`SentenceText`**: This class represents a plain sentence, extracted from a business process textual description. It has two main responsibilities:
  - Store whether the sentence is a new sentence or if it was removed from the original description.
  - Given a String (plain sentence) it must identify which process model element it represents, *i.e.*, if the sentence represents an activity, gateway or an event. To accomplish this goal it must rely on several NLP techniques, which are injected during runtime through the *INaturalLanguageProcessor* interface.
- **`ProcessProperty`**: Domain object responsible for storing the general properties for a process model element, which are: its ID and its type (*e.g.* Event, Activity or Gateway).
- **`ProcessActivityProperty`**: Domain object responsible for storing the specific properties for an activity model element. The activity's specific properties are: business object, action, actor and addition.
- **`ProcessElementType`**: Enumeration (keys) that represents all the process model elements currently supported by the NLP core module. Table 3.2 details the elements that are supported by the module.
- **`ProcessEventProperty`**: Domain object responsible for storing the specific properties for an event model element. The event's specific properties are: event type (which is an instance of *ProcessElementType*), identifier and addition.
- **`ProcessGatewayProperty`**: Domain object responsible for storing the specific properties for a gateway model element. The gateway's specific properties are: gateway type (which is an instance of *ProcessElementType*), description and a list of RealizedText objects which stores the branches, each identified by a bullet character in the text. Section 3.3.2 illustrates a textual gateway with several branches.
- **`OperationType`**: Enumeration with all possible operations that can trigger the synchronization between a business process textual description and its respective model.
- **`PortugueseLanguageProcessor`**: Concrete implementation of the `INaturalLanguageProcessor`, considering the Portuguese language. It implements

several procedures which are essential for extracting process semantic from natural language texts. Examples of these procedures are: Algorithm 1, 2, 3 and 4.

Table 3.1: Overview of the current template text pattern supported by the NLP Core Component.

Patterns Mappings Supported
ACTIVITY-PASSIVE = BO + [ADDITION] + ACTION + ACTOR **ACTIVITY** = [DISOCURSE_MARKER], ACTOR + ACTION + BO + [ADDITION]
EVENT-BEGIN-1 = [The] process begins [when there is] BEGIN_DESCRIPTION. **EVENT-BEGIN-2** = [The] process begins [when] ACTIVITY. **EVENT-FINISH-1** = Finally, the process finishes. **EVENT-FINISH-2** = Once all the brances are executed, the process finishes with FINISH_DESCRIPTION.
GATEWAY-XOR-1 = IF + CONDITION_CLAUSE + [THEN] + ACTIVITY. **GATEWAY-XOR-2** = IF + CONDITION_CLAUSE + [THEN] + ACTIVITY. OTHERWISE + ACTIVITY. **GATEWAY-PARALEL** = DISOCURSE_MARKER + the process is divided into x parallel branches: **GATEWAY-AND** = DISOCURSE_MARKER + the following branches are executed:

The framework is composed by several classes which can be grouped according to three perspectives: (i) The `NLG core` (Figure 3.5), which contains the classes responsible for generating natural language text from business process models; (ii) The `Language core` (Figure 3.6), which contains the classes responsible for dealing with the linguistic aspect (morphology, semantics and syntactic rules) of specific languages; and (iii) The `NPL core` (Figure 3.4) which contains the classes responsible for dealing with natural language processing (*e.g.*, extract relevant business process information from the text).

With the definition of the NLG and NLP core module, the developer does not need to know in detail how the NLG or NLP process works. The module assures that a natural language text will be produced for the given language, as long as interfaces and their methods are implemented according to the specification (`Language core`, Figure 3.6). Regarding the interfaces (hot spots), there will be `n` implementations of the same interface, where `n` is the number of different languages currently supported.

Table 3.2: Overview of the Process Model elements supported by the NLP Core Component

Process Model Element	Description
GATEWAY_AND	Represents an AND gateway element (*control-flow*)
GATEWAY_PARALLEL	Represents a parallel gateway element (*control-flow*)
GATEWAY_XOR	Represents a XOR gateway element (*control-flow*)
ACTIVITY	Represents an activity element
EVENT_FINISH	Represents an end event element
EVENT_FINISH_WITH_DESCRIPTION	Represents an end event element with a descriptive label
EVENT_BEGIN_WITH_DESCRIPTION	Represents a begin event element with a descriptive label
EVENT_BEGIN_WITH_ACTIVITY	Represents a begin event element followed by an activity element
UNKNOW	Type assigned to those sentences that could not be mapped to any of the above elements

3.1.2Framework classes and Interfaces Specification

This section aims to details the most important interfaces and packages, providing the necessary documentation to add support to new languages.

- `LabelAnalysis`: This package is responsible to extract linguistic information from process model labels. Interfaces defined in this package (hot spots) must be implemented for each supported language, *i.e.*, all the linguistic classification algorithms must be implemented for each language. For example, algorithms to identify that *assess* is the verb in the label *application assessment*.

- `ILabelCategorizer`: This interface (Figure 3.7) defines all the methods for the label classification into one specific style, which can be: Action-Noun (AN), Verb-Object Style (VOS) or Descriptive Style (DES). There are more styles that

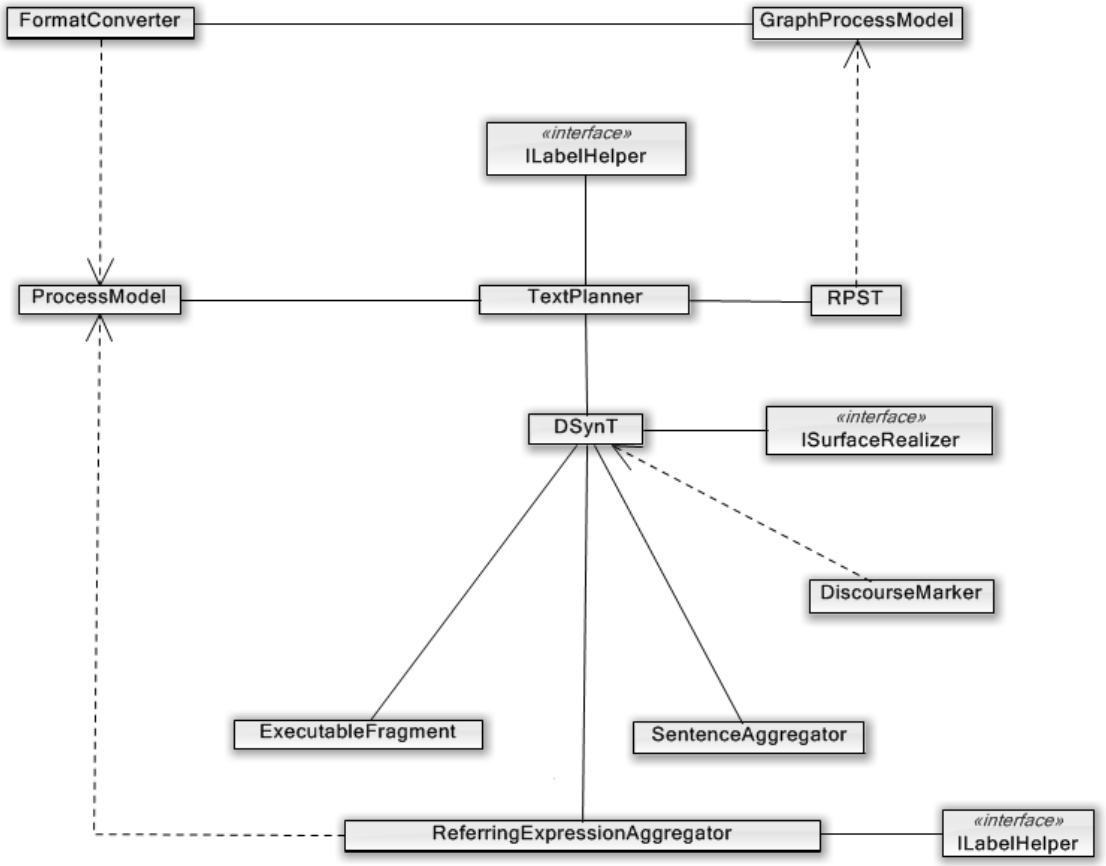


Figure 3.5: Classes used to generate a natural language text from a business process model. Together they form the NLG core.

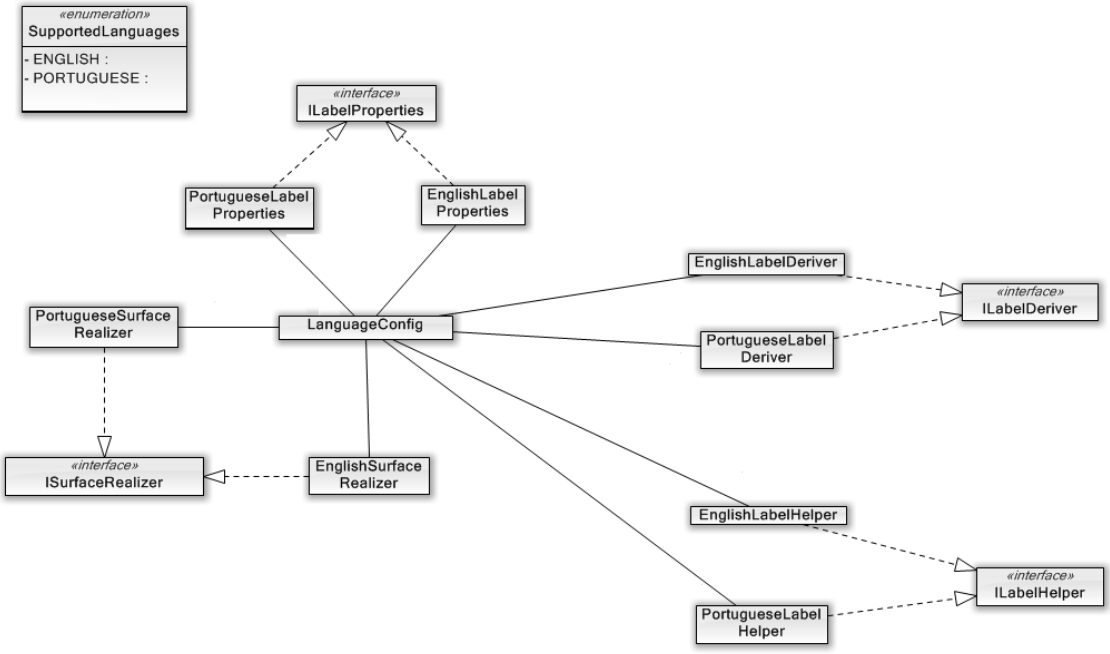


Figure 3.6: Classes used to deal with the linguistic rules of a specific language.

can be used to create labels in process models, but they are not covered by this paper. For further reference about labeling styles, please refer to [61].

- `ILabelProperties:` This interface (Figure 3.10) contains all the necessary methods definitions for the storage and retrieval of the label properties. Each process model label, have its according ILabelProperties object. This object gather all the textual information extracted from the process label, which contains important information, *e.g.* Actors, Business Objects and Actions. In other words, it is used as a container for the information which will be needed to generate a complete natural language sentence. When the relevant information is extracted form the process label, it is stored within as within the object property. Afterwords, the information is retrieved by querying the respective property.

- `ILabelHelper:` This interface (Figure 3.9) contains all the necessary methods definitions for the analysis of text inside a process model label. For example, in order to find out if the first word of a label is likely an action, the method *isVerb* is called. If the first word is a verb, then the Action property of the ILabelProperties object will be set to the respective word. Regarding semantic aspects, this interface defines the most important methods for a specific language. All the linguistic specific logic will be centralized in this interface implementation. The algorithms must be able to, *e.g.*, given a specific word (string), identify if it is a verb, noun, adjective, adverb etc. The main algorithms that were developed considering the Portuguese language can be found in the Appendix A.

- `ILabelDeriver:` This interface (Figure 3.8) is used to derive all the information from a specific label category, known as Verb-Object Style (VOS) [61]. When a VOS label is found, during the analysis of the business process model, the information extraction for the given label starts. The objective of the implementation for this interface is to populate the ILabelProperties object with all the relevant textual information read from the Label object. In order to find out which information are relevant and their linguistic semantic, the implemented methods from the ILabelHelper must be called.

- `ISurfaceRealizer:` This interface (Figure 3.11) defines the necessary methods which are called to generate the final text. Basically, the most important implementation is the realizeSentence method. This method receive as input a DSynT tree with all the textual information gathered during the textual planning and sentence planning phases (which contains data about the activities sequences). Through the DSynT tree object's properties, it is possible to assemble the respective sentence.

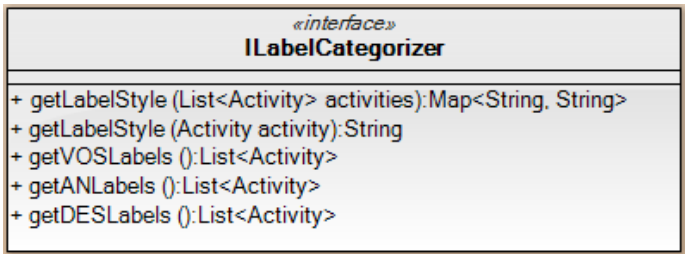


Figure 3.7: Interface responsible for the definition of methods needed to classify a label into one specific style.

Figure 3.8: Interface responsible for the definition of methods needed to extract information from a specific label style.

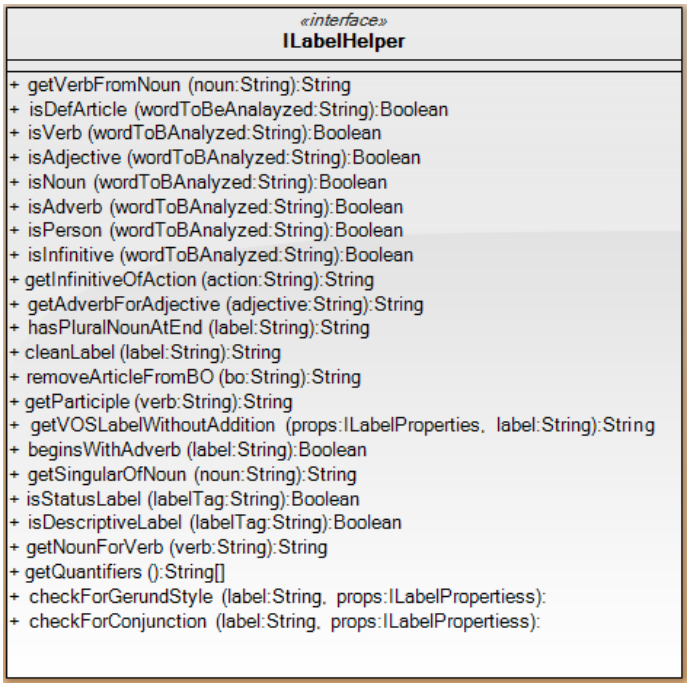


Figure 3.9: Interface responsible for the methods definitions of text analysis inside a process model label.

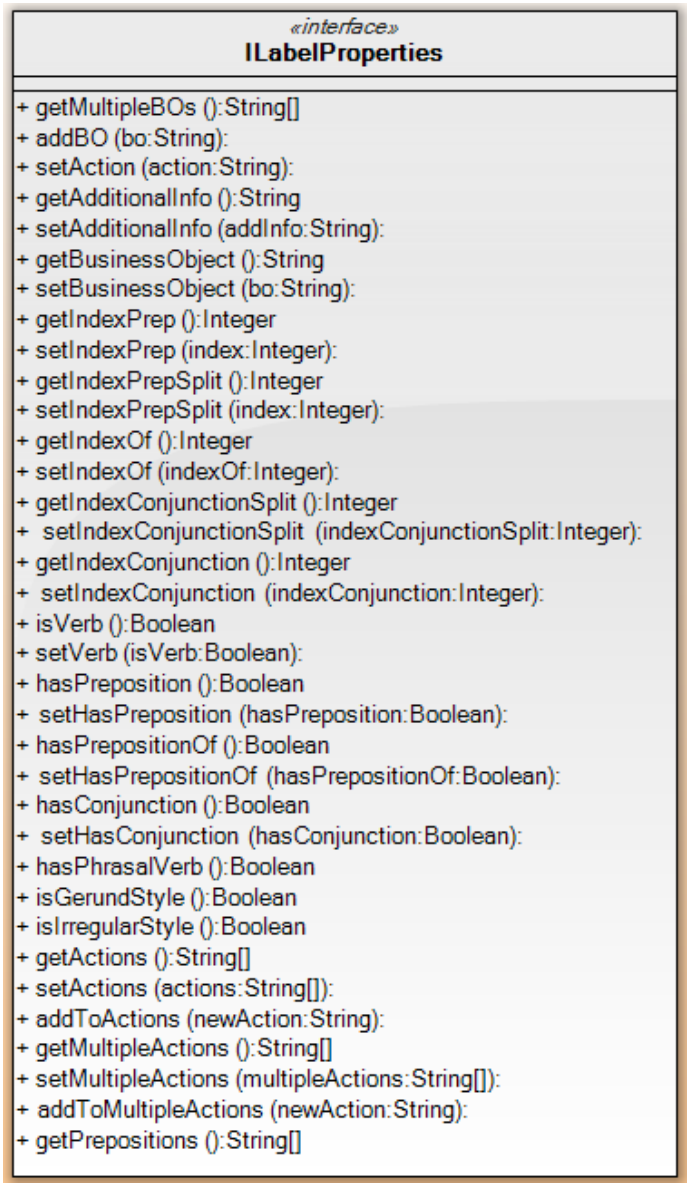


Figure 3.10: Interface responsible for all the necessary methods definitions for the storage and retrieval of the label properties.

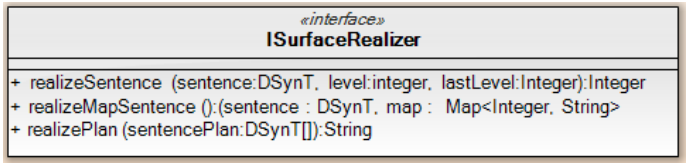


Figure 3.11: Interface responsible for the definition of the necessary methods which are called to generate the final text.

3.2Model to Text: Natural Language Generation from BPMN process models

This sections aims at describing in details how each NLG component works, presenting the set of algorithms deployed for each component. By running the NLG pipeline, natural language texts can be produced from process models. The only pre-requisite for parsing the models is that they must be compliant to the main process modeling and labeling style guidelines [80, 79, 61].

3.2.1Text Planing

This section details the implementation regarding the first phase of the NLG pipeline process (depicted in Figure 2.10). To treat the Text Planing step, the following implementations were considered.

Linguistic Information Extraction The main goal of this component is the correct inference of the linguistic information from all the labels (*e.g.*, name of the activities, descriptions, event labels, gateways labels and actors) in a business process model. This component was built on process modeling guideline premises. Thus, it is essential that the process models guidelines were followed while designing the model. The package Label-Analysis of the GeneralLanguageCommon module (Figure 3.2) contains all the methods that are used during the information extraction phase. In particular it is capable of parsing labels written using the naming conventions (*i.e.*, label styles) presented at Table 3.3 [61]. In a nutshell, it reads all the label from the process model and populates, for each label, the respective ILabelProperties object (Figure 3.10) with the label's properties (*e.g.* actor, action, business object). The identification of the semantic function of a specific word is given by the ILabelHelper implementation for the specific language (Figure 3.9). In special, the following methods are essential for the information extraction: isVerb , isNoun , checkForConjunction , isDefArticle, removeArticleFromBO and getPrepositions .

Annotated RPST Generation This module is responsible for creating the RPST tree representation (class RPST depicted in Figure 3.5) of the process model (class Process-Model, depicted in Figure 3.5) received as input by the system. It receives as input a graph structure (class GraphProcessModel, depicted in Figure 3.5) that represents the process model. The graph is generated in the previous step of linguistic information extraction.

Table 3.3: Supported Labeling styles

Labeling Style	Core Structure	Example
Verb–object VO	A (imperative) + O	Create invoice, Crie nota fiscal
Infinitive Style IS	A (infinitive) + O	Create invoice, Criar nota fiscal
Action-noun AN (np)	O + A (noun)	Invoice creation
Action-noun AN (of)	A (noun) + "*of*" + O	Creation of invoice
Action-noun AN (gerund)	A (gerund) + [article] + O	Creating invoice
Descriptive DES	[role] + A (3P) + O	Clerk creates invoice

Text Structuring The main responsibility of this component is the addition of paragraphs, text indentation and markers, allowing the text to be shown in a fashion format and improving the text's quality and readability. Basically, the RPST tree's nodes are traversed and their structuring properties are set. For example, if an activity must start a new paragraph, the attribute *hasParagraph* of the respective node is set to true. Others properties, like *hasBullet* and *senLevel* are used to help with the indentation of the final text. These properties are read by the Message Realization phase to generate the text with the correct indentation for activities that are executed in parallel, for example.

3.2.2Sentence Planning

This section details the implementation regarding the second phase of the NLG pipeline process (depicted in Figure 2.10).

DSynT Message Generation This component transforms the previously created RPST tree into a list of intermediate messages, *i.e.*, the linguistic information of the model is not directed mapped to the final text, rather to a conceptual representation that is still suitable for changes. In particular, each sentence is stored into a DSynt tree. The Dsynt package (depicted in Figure 3.2) have all the classes that are used in this phase. Basically, the package contains the specific DSynt types, which can be a conditional sentence (originated from, for example, a XOR-Gateway) or a regular sentence. The DSynTSentence class is abstract, hence it cannot be instantiated. It was created only to store all the DSynT properties that are common for both, conditional and regular sentences. Also, it provides default methods for accessing some of the mandatory properties.

Message Refinement This component is used to the message aggregation task. The need for message aggregation arises when the process contains a big sequence of activi-

ties. In this case, we can use three types of aggregations techniques: aggregation by actor, by business object and by action. There are two classes involved in the aggregation task: *SentenceAggregator*, which aggregate sentences executed by the same actor (role) and *ReferringExpressionAggregator*, which aggregate sentences through the addition of referring expressions, *i.e.*, addition of pronouns to avoid unnecessary repetition of actors. Both classes are depicted in Figure 3.5. The message refinement must use several methods defined by the ILabelHelper interface, in order to generate a sentence correctly: *isPronoum*, *getGender* and *getPronouns*.

3.2.3Sentence Realization

This section details the implementation regarding the last phase of the NLG pipeline process (depicted in Figure 2.10). This component is responsible for generating the natural language text, which represents the process model in a textual format. The complexity of the message realization task lead to the development of public available tools, like TG/2 and RealPro[2] Realizer [11, 60] for English. Due to the lack of an adequate tool to the message realization process in Portuguese, specific algorithms to treat the Portuguese sentences were implemented.

It was observed, based on the NLG pipeline, that aspects and operations were common and general to any language belonging to the Romanian and Germanic sub-branches of the Indo-European language family. Hence, a generic mechanism that accepts the implementation of a message realization module in a given language, without the need to change the others systems modules, was created (interface ISurfaceRealizer from the GeneralLanguageCommon package - Figure 3.2). In a nutshell, the implementation of this interface for a specific language must be able to, given a DSynT tree, read the textual information from the nodes and assemble a grammatically correct sentence (this process is triggered by the *realizeSentence* method).

After the identification of the necessary operations to elaborate the Portuguese module, the creation process of the necessary algorithms to the correct execution of the operations was straightforward. First, a verb database was created with verbs gathered from the tool KonjugationsHase from the German organization Cactus2000[3], totalizing 15310 different verbs in Portuguese. Then, a second database was created for the identification and manipulation of others syntactic functions that the words of a process model can assume, like: nouns, adjectives, pronouns, articles and adverbs. The necessary data was colected from

[2]For more information, please refer to http://www.cogentex.com/technology/realpro/

[3]The verbs can be downloaded from http://www.cactus2000.de/de/software/conjughase.php

Floresta[4] corpus [2] for Portuguese, Totalizing 1.640.000 words. All the words within the Floresta corpus were tagged according to the predefined categories that a word can assume (*e.g.*, nouns, adjectives, pronouns, articles and adverbs). Some minor changes were done to the original corpus database, enabling it to be read by the framework's classes. Hash tables were used to map the words to their respective category (*i.e.*, syntactical function). The grater the word database is, the bigger is the coverage for words found withing process models. Regarding the English language, a Java WordNet library was used for querying the syntactical function of words. These syntactic functions can take different shapes in a process model, they can be a business object, an actor, or express some condition or a complement. With all the necessary and already structured data, the methods to generate a grammatically correct message were developed. Some of them are responsible for classifying a given word, extracted from the process model, according its respective syntactic function through a simple search in the database for the respective word. Some of the operations that represents these functionality are exposed by the ILabelHelper interface, such as *isAdverb* , *isVerb* , *isNoun* and *isAdjective*.

Others methods are responsible for the transformation of a given word in some specific form, for example, transformation of the verb that represents an action extracted from an activity of the model in the infinitive form for its respective conjugation in the 3rd person singular. Other example is the transformation of the gender and form of a given noun, to its plural or singular form, as well as to the masculine or feminine if it is necessary. The related methods are: *getInfinitive* , *getParticiple* , *getPresent* , *is3PS* and *transformToSingularForm*.

Based on these methods, for each DSynt tree, the correct article to the respective nouns are added, the verb is conjugated in the correct form according to the context and with the noun it is related to. Besides, the necessary connectives are added, like comas, spaces and word capitalization, if it is necessary. Some of the related methods are: *getGender*; *getArticle*.

3.3Text to Model: BPMN process model generation from Natural language Texts

The round-trip approach is represented by the capability of generating natural language text from process models and generating updated versions of the process models from

[4]Floresta is a corpus (big collection of articles) composed by almost 95.000 sentences (approximately 1.600.000 words) gathered from CETENFolha corpora (texts from the Brazlian newspaper Folha de São Paulo) and CETEMPúblico (public portuguese diary, dated from 1991 to 1998). The entire corpus was automatically analyzed by PALAVRAS syntatc analyzer [2].

modified natural language texts.

This section details how NLP techniques were implemented by the NLP Core component deployed within framework, enabling the extraction and parsing of natural language texts, in order to gather business process data and present this data through a machine artifact format (*e.g.*, BPMN process model).

The NLP process follow the steps presented in Figure 3.12. The following sections present details about each step and when each method is triggered within the pipeline execution.

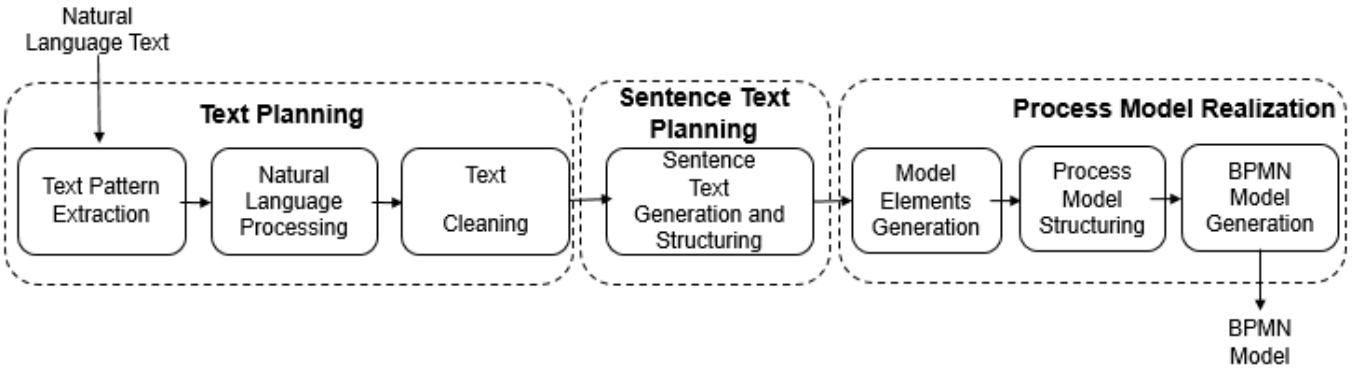


Figure 3.12: Text to model pipeline - Steps performed to generate BPMN model from natural language text.

3.3.1Text Planing

This section details the implementation regarding the first phase of the NLP pipeline process (Figure 3.12). This module finds the right strategy to extract linguistic data from the text, decides what to do with unnecessary data (*e.g.*, words that has no semantic relevance) and also deals with referencing problems between sentences. It does not have any process model specific logic, being suitable for reuse in other applications which needs NLP processing capabilities. To treat the Text Planing step, the following implementations were considered.

Text Pattern Extraction The main goal of the component `Text Pattern Extraction` is the correct inference of textual pattern represented by the text received as the input. The text input (*e.g.*, the text presented in Table 3.4) is received as parameter by the RealizedText object, which retrieves text patterns from the database (*i.e.*, implementations for ITextFormatter interface) and runs its inicializeTextFormatter method (depicted in Figure 3.4) against the current framework's configuration. If any match is found, it in-

stantiates the specific implementation mapped to the given pattern. Otherwise, it returns an error message to inform the user that the given pattern is not currently supported by the framework and thus aborting the pipeline execution.

Table 3.4: Supported Text Pattern received as input.

The process begins when the Room-Service Manager takes down an order. Then, the process is split into 3 parallel branches: - In case alcoholic beverages are ordered, the Room-Service Manager gives order to the Sommelier. Afterwards, one or more of the following paths are executed: * The Sommelier fetches wine from the cellar. * The Sommelier prepares the alcoholic beverages. - The Room-Service Manager submits the order ticket to the Kitchen. Subsequently, the Kitchen prepares the meal. - The Room-Service Manager assigns order to the Waiter. Then, the Waiter readies the cart. As long as all the 3 branches were executed, the Waiter delivers to the guest's room. Afterwards, the Waiter returns to the room-service. Subsequently, the Waiter debits from the guest's account. Finally, the process finishes.

To better illustrate this component usage, consider that the framework's configuration for the text format option is set to "*TemplateTextFormatter*". The *RealizedText* object would search for classes named "*TemplateTextFormatter*" that implement *ITextFormatter* interface. This component serve as a initial validation for the textual pattern, avoiding unnecessary processing by the NLP core, thus saving both user's time and system processing resources. The validation consists of verifying if the text received as input follow any supported text pattern.

Natural Language Processing The main goal of the `Natural Language Processing` component is to analyze the text's semantic and extract relevant linguistic information from the text. It is in this phase that the main NLP concepts (detailed in Section 2.3) are employed in specific algorithms to recognize specific patterns (*e.g.*, end of a line), splits the text into sentences, the sentences into words array and classify words according to their respective syntactic rule.

The first step is the stop words removal, which consist of removing words that are of little value when building up a business process model (*e.g.*, articles and discourse markers

such as "*Then*", "*Afterwards*"). These stop words are stored within a file or in the database and may vary according to the text's current language or text structure. Therefore, the method is defined in the `ITextFormatter` interface and can thus be overridden with custom code if the self-package algorithms does not fit the developer needs.

The second step is the Part-Of-Speech Tagging (POS), which consist of tagging each word (within the word array) with its respective syntactic role (*e.g.*, plural noun, adverb). For each word, it tries to identify its syntactic role. If no such role is found, then the word is stored into an ignored words list and it is excluded from the word array.The ignored word is stored because it is further referenced by the `Model Elements Generation` component.

Text Cleaning The main goal of the component `Text Cleaning` is to undo message refinement step executed by the NLG pipeline (Section 3.2.2). Undoing message refinement is responsible for replacing referring expressions and sentences aggregators. As an illustration example, consider the following text:

```
In case alcoholic beverages are ordered, the Room-Service
Manager debits from the customer account and gives order to
the Sommelier. Afterwards, he fetches wine from the cellar
and prepares the alcoholic beverages.
```

If the Text Cleaning component is triggered for the above text, the result would be:

```
In case alcoholic beverages are ordered, the Room-Service
Manager debits from the customer account. The Room-Service
Manager gives order to the Sommelier. The Sommelier fetches
wine from cellar. The Sommelier prepares alcoholic beverages.
```

Here some challenges arises, for example, how to replace correctly the referring expression object. This challenge is known in NLP research area as the "*Referencing*" challenge (see Section 2.3.5). This component addresses this challenge by using a simple heuristic: if a referring expression is found, then it looks for the previous role (*i.e.*, actor) and replaces the pronoun by the role found. If no such role is found, then it returns an error message stating that a referencing problem has been found and it must be fixed manually by the user before submitting the text again.

The use of a simple heuristic is justified because the texts used as input follows specific patterns and are not totally unstructured. Despite of not being generic enough to deal with virtually any text pattern, the NLP Core Component architecture is flexible and can be

extended to support any text pattern, as long as the interface methods are implemented according to the specification.

3.3.2Sentence Text Planning

This section details the implementation regarding the second phase of the NLP pipeline process (Figure 3.12). This module uses the text treated by the previous module to generate several *SentenceText* objects to store textual data and the correct structure so that the text structure can be mapped to sentences in the right order. To treat the Sentence Text Planing step, the following implementations were developed.

Sentence Text Generation and Structuring The main goal of the `Sentence Text Generation and Structuring` component is to generate the *SentenceText* objects for the original *RealizedText* created from the text received as input, while preserving the description structure. In this context, description structure means: (i) the activities sequence orders; and, (ii) the correlation between them. For example, consider the following text fragment extracted from Table 3.4:

`In case alcoholic beverages are ordered, the Room-Service Manager gives order to the sommelier. Afterwards, one or more of the following paths are executed.`

- `The Sommelier fetches wine from the cellar.`
- `The Sommelier prepares the alcoholic beverages.`

`Then, the Waiter delivers the beverages to the guest.`

In the above example, "*In case alcoholic beverages are ordered*" express a condition (*i.e.*, a control-flow) and the bullet indicates that the associated sentence is one possible output for the control-flow. This structure is particular important regarding business process models. If the same text were written as:

`In case alcoholic beverages are ordered, the Room-Service Manager gives order to the sommelier. Afterwards, one or more of the following paths are executed. The Sommelier fetches wine from the cellar. The Sommelier prepares the alcoholic beverages. Then, the Waiter delivers the beverages to the guest.`

It would be not easy to determine which sentences were suitable outputs for the control-flow. For instance, identify the sentence "*Then, the Waiter delivers the beverages to the guest*" be one possible output for the control-flow. It is easier to notice that this sentence is not a possible output if we look at the first text (with formatting characters), whereas this is not clear looking at the second text (without any formatting characters). Figure 3.13 illustrates the correct mapping between this set of sentences and the corresponding BPMN elements.

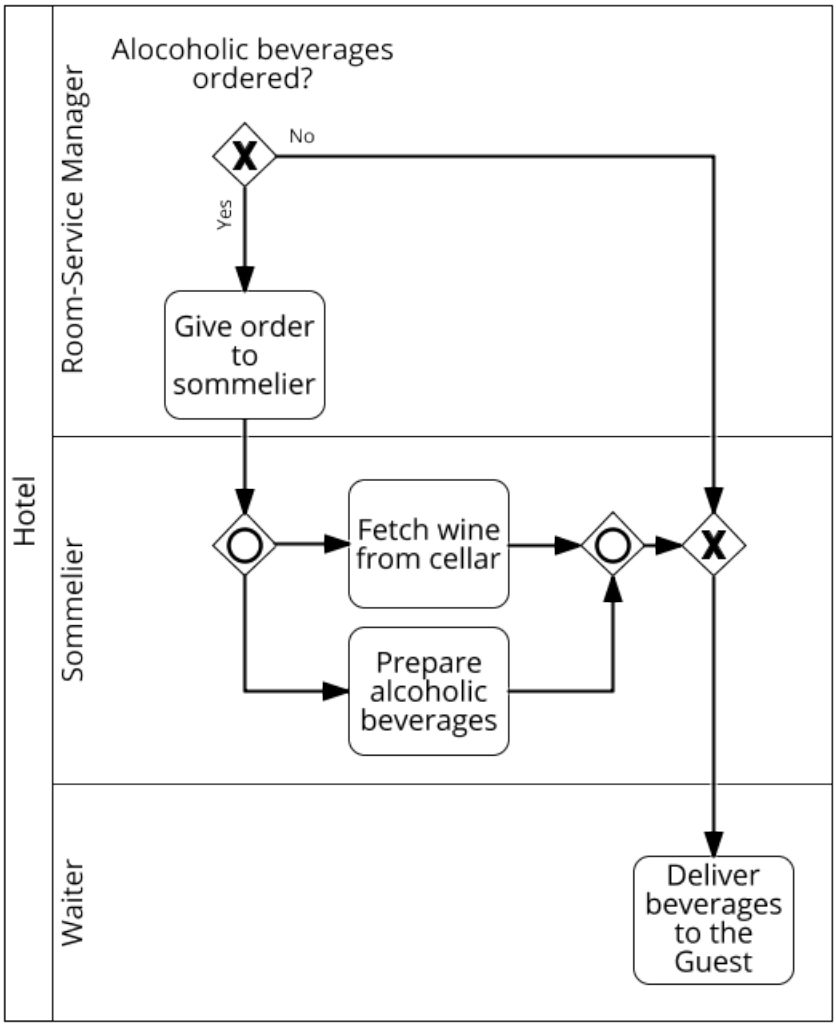


Figure 3.13: Process fragment used for illustration purpose.

The strategy (*i.e.*, implementation) for generating *SentenceText* may vary according to the *ITextFormatter* implementation. For illustration purpose, consider the following text fragment extracted from Table 3.4 and assume the *TemplateTextFormatter* as the default implementation for *ITextFormatter* in this case:

```
The Room-Service Manager gives order to the Sommelier.
```

```
Afterwards, one or more of the following paths are executed.

  • The Sommelier fetches wine from the cellar.
  • The Sommelier prepares the alcoholic beverages.

Then, the Waiter delivers the beverages to the guest.
```

The text input is obtained by the RealizedText's *getPlainText* method and it is then split by sentence period limiter ('.'). The *SentenceText* objects are created in the same order which they appear in the original text, thus preserving the activities sequence orders. If a bullet is found, then a special treatment is applied. Another *RealizedText* object instance is created, to simulate a new branch level, and all the sentences within the same bullet context are added to this secondary *RealizedText* object sentence list. This secondary *RealizedText* is added to the branch list of the last *SentenceText* from the main *RealizedText* object. Please note that this procedure may be applied recursively if a sub bullet is found within a main bullet sentence. At the end of these steps we would have a tree structure (similar to the RPST data structure). The Algorithm 7 describes the above procedure in pseudo-code. Figure 3.14 illustrates how the sample text is mapped to the objects described above and how their relations enable them to be disposed as a tree structure.

This sentence structuring was designed to enhance and facilitate the process of generating process models from textual descriptions, as storing the elements through a tree data structure enables to traverse it using Depth-first search in pre-order [28]. In the example depicted by Figure 3.14 the correct visiting order would be: Tree Node 1 -> Tree Node 2 -> Tree Node 3 -> Tree Node 4 -> Tree Node 5.

3.3.3Process Model Realization

This section details the implementation regarding the last phase of the NLP pipeline process (Figure 3.12). This module is based on the objects created by the previous module and has all the process model generation specific logic. To treat the Process Model Realization step, the following implementations were considered.

Model Elements Generation The `Model Elements Generation` component is responsible for generating the model elements from the *SentenceText* objects. The biggest challenge for this step is to offer support to multiple text structures and formats. To address this challenge, the `INaturalLanguageProcessor` interface was defined. By implementing the interface, the developer is capable of defining its own custom method

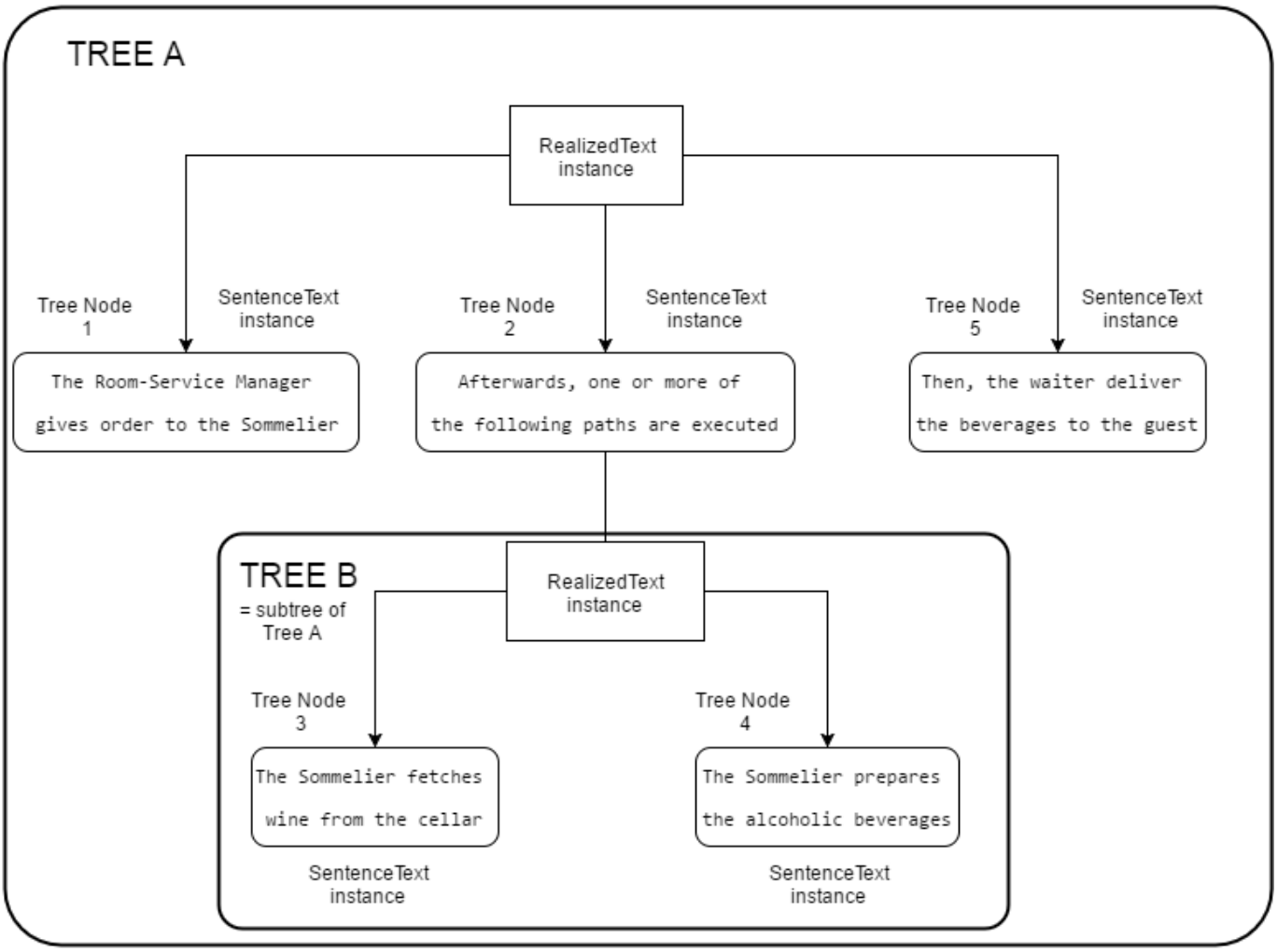


Figure 3.14: Example of the relations among the objects seen as a Tree Structure.

for generating the model elements from the *SentenceText* data, instead of using the self-packaged algorithms. The default framework implementation is based on a multi-language template pattern, with its own syntax (*TemplateTextFormatter*, Section 3.1.1). It can be extended to cover new patterns and thus, offer support for mapping different text sentences to the same model element type. Table 3.1 depicts the implemented template, which is language-independent. As can be observed, the template offer support to specific text structures, which can then be mapped to process model elements. After mapping each sentence to a *SentenceText* object instance, the model elements generation phase starts. For each *SentenceText*, the remaining formatting characters (*e.g.*, bullets) are removed and the sentence is mapped to a process model element. After identifying the model element that the sentence represents, a link is created between the *SentenceText* and the *ProcessProperty* object to act as a glue between the language level and process model level analysis (both classes are depicted in Figure 3.4). With this link, it is possible to track back the original sentence associated with a process model element and vice-versa.

The Atomicity challenge (described in Section 2.3.5) is also addressed by this component. For example, it must take into consideration that a single activity may correspond to multiple sentences and vice-versa. Consider the following sentence: "*The process begins when the Room-Service Manager takes down an order.*". Two process model elements can be identified when analyzing this sentence: (a) "*The process begins*" represents a begin-event and "*Room-Service Manager takes down an order*" represents an activity. This component is capable of identifying such scenarios and handle them accordingly. For this specific example, after identifying that the sentence represents a begin event associated with an activity, it stores the event textual description in one *SentenceText* object and creates a new *SentenceText* object to store the activity textual description and updates the links between these sentences (Figure 3.15).

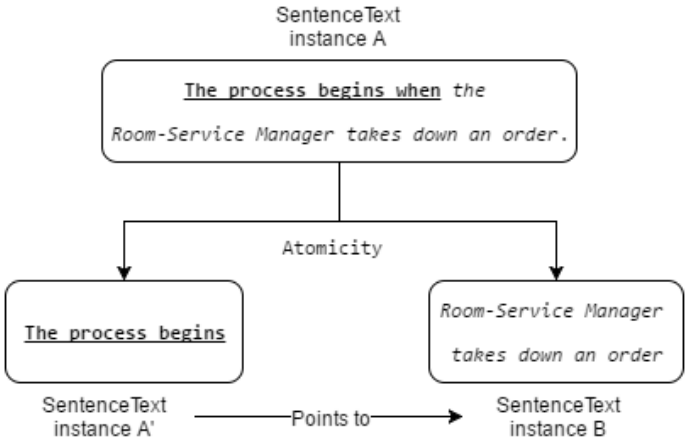


Figure 3.15: *SentenceText* instance A is broken into two parts, to address the Atomicity challenge: *SentenceText* instance A' and *SentenceText* instance B.

Process Model Structuring The model structuring refers to structuring the generated elements according to the order which thy appear within the text and also respecting sentences correlations, in the same manner as it was described in *Sentence Text Generation and Structuring* component. Taking advantage of the link between the *SentenceText* and the associated model element, we also benefit from the initial sentences structuring. In other words, the process model elements structuring is the same as the sentence structuring. Each *SentenceText* is mapped only to one model element, and we just need to traverse through the sentences tree nodes, fetch its associated model element and create the ProcessModel object. With the process model object (which is not notation specific), it is possible to translate it to any specific notation and print out the whole process model to the desired output (*e.g.*, JSON, xml, bpmnl, etc), which is done by the next and final

NLP's pipeline component.

BPMN Model Generation The main goal of the `BPMN Model Generation` component is to generate a business process model in BPMN standard notation. It is important to notice that, the *ProcessModel* object generated in the previous step is not notation specific. It can be instantiated to any notation, as long as this component is extended to support the chosen notation. The default generation algorithm generates the model as a machine artifact, more specifically it generates a JSON file that represents the process model written using the BPMN notation. The JSON file is then read by a BPMN graphic modeler tool to dispose all the elements visually for the end user. In particular, the Signavio[5] online platform is used to read the JSON file and output the graphic process model to the user. As flexibility is the key design point of the framework, it supports generation to other machine artificats formats (*e.g.*, xml) through the use of interfaces. If needed, the developer just need to implement the interface methods to add support for generating a BPMN model in another file format.

3.4Strategy for Incorporating Textual and Model-based Changes

The discussion in the Section 2.2.4 (NLG) and Section 2.3.5 (NLP) illustrates that a mapping between text and process model is associated with many challenges. In order to adequately implement the round-trip, we hence have to define which change scenarios exist and how they can be implemented for model and text. For process models, change scenarios have been intensively discussed by Weber et al. and Kolb et al., which also discuss how the basic change operations for process models translate into textual changes [136, 51]. Table 3.5 gives an overview of change operations for process models, texts and their connection.

Using the above table as a guide, we defined the main operations that should be supported by our round-trip technique. These operations were implemented within the framework, enabling it to synchronize textual changes back to the original process model. The operations must be inferred by analyzing the differences between the original text and the changed version. For example, if a sentence is removed from the text, it must be tagged as an activity/event removal. Afterwards, the correct business process element (*e.g.*, Activity, Event, Gateway and Role) is tracked through the data structure links between sentence elements and process elements. Finally, the operation is performed in business process

[5]available at www.signavio.com

Table 3.5: Overview of Change Operations for Process Model and Text and their Connection, adpated from Kolb *et al.* [51]

Operation on Model	Operation on Text
Add Activity / Event	Add Clause / Sentence
Add Gateway	Add Meta Clause / Meta Sentence
Add Business Object	Add Object to Clause / Sentence
Add Addition	Add Adverbial Phrase to Clause / Sentence
Add Role	Add Subject of Clause / Sentence
Change Activity / Event	Change Clause / Sentence
Change Gateway Label	Change Meta Clause / Meta Sentence
Change Action	Change Verb of Clause / Sentence
Change Business Object	Change Object of Clause / Sentence
Change Addition	Change Adverbial Phrase of Clause / Sentence
Change Role Label	Change Subject of Sentence
Remove Activity / Event	Remove Clause / Sentence
Remove Gateway	Remove Meta Clause / Meta Sentence
Remove Business Object	Remove Object of Clause / Sentence
Remove Addition	Remove Adverbial Phrase from Clause / Sentence
Remove Role	Remove Subject of Sentence

model level, removing the element from the original process model. These steps are the same for changing sentences on the original text.

For sentences addition, it has to perform another step after having identified the text changes (Section 3.4.1), because the link between sentence and model still does not exist. As seen earlier, the model-sentence links are automatically created from a process model as input. In this case, there is no such process model yet. Thus, the links and the process' model elements itself must be created using the same strategy described in sections 3.3.2 and 3.3.3. After creating the model element, the links are added to the data structures and the model is then updated (Section 3.4.2).

Depending on the context, there are some operations that can trigger other operations in order to complete successfully. For example, let's consider a simple process that has just one role and two activities. Figure 3.16 illustrates the process through both knowledge representations – model and text. If the role "*Secretary*" is changed to "*Intern*" in the sentence "*Afterwards, the secretary writes down the message*", the role element's label cannot simply be replaced to "*Intern*". Otherwise, the updated model would result in

two activities being executed by the "*Intern*" which is not the desired result. Instead, the correct output should be the addition of a new role and mapping the activity "*Write down message*" to this role's lane. Figure 3.17 illustrate the correct output for the sample process. So, despite having changed only one word in the original text (second occurrence of "*Secretary*" to "*Intern*"), the changes reflected to the original model were much more complex than simply updating the role's label. The framework algorithms are capable of dealing with these scenarios as explained in Section 3.4.2.

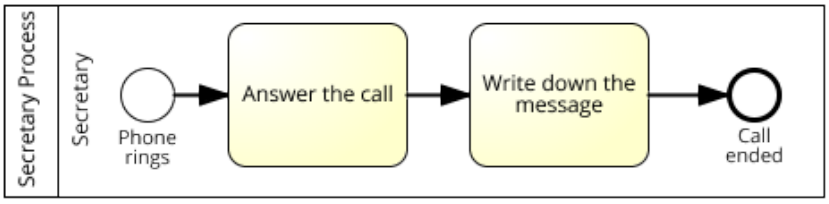


Figure 3.16: Simple process, represented in both text and model, designed only to illustrate a simple role change example.

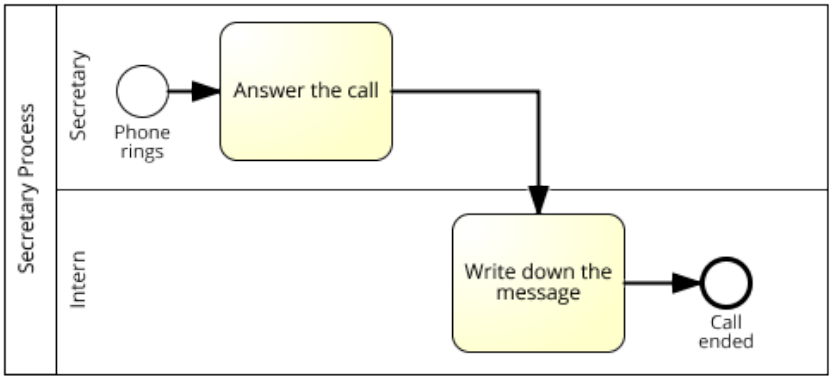


Figure 3.17: Updated version of the previous simple process, represented in both text and model.

3.4.1Textual Diff

This section details the strategy developed for identifying changes in the original text. First, it was reasoned about weather using an out-of-the-shelf solution or if a new one should be developed to identify text changes in specific textual patterns. Most of the available tools works with the Levenshtein distance between two strings, which gives the number of edit operations that are necessary to modify one string to obtain another string [54].

For this case it is not necessary to know how many character does the string differ from another string. It is enough to know that they are not the same. As a result, it is not necessary to use such algorithms for now. Besides, an additional effort would be necessary to map the result to the specific data structures already developed. Thus, a custom technique was preferred to best fit the change operations requirements (Table 3.5). The algorithm details (in pseudo code) are presented in the Appendix (Algorithms 5, 6 and 8). Nevertheless, an abstracted explanation of the technique can be found next. Basically, the change operations can be grouped into two groups: insert and delete. The changes (*i.e.*, updates) can be seen as a special case, where the original sentence is tagged as "*Deleted*" and the new one is inserted in its place. In other words, there is no need to identify weather only a specific word has changed within a sentence. Instead, a binary evaluation is performed, tagging the sentences as deleted (0) or inserted (1). While marking the sentences as removed or inserted, the algorithm also calculates the suitable position that this operation should be performed in the original text. The example below illustrates the technique usage considering 0 based indexes (*i.e.*, sentence 0 is the first text's sentence):

- **Original Text** = `O processo comeca quando o Secretario recebe o pedido de compra. Finalmente, o processo é terminado.`
- **Changed Text** = `O processo comeca quando o Secretario recebe o pedido de compra. Em seguida, o gerente finaliza o pedido. Finalmente, o processo é terminado.`
- **Result** = "*`Em seguida, o gerente finaliza o pedido`*" marked as inserted with index 1. In other words, to update the original text it would be needed to insert the string as the second sentence in the original text, and shift forward all remaining sentences. In this case, the sentence "*`Finalmente, o processo é terminado`*" index would be shifted from 1 to 2.

The development of the textual diff consisted of three main classes, which are described as follows:

- **`DiffAnalyzer`**: responsible for the heavy work during the diff analysis. It implements the algorithms described above. In a nutshell, breaks the texts into sentences, process the removals, insertions and then updates the original text applying the changes detected. The result (output) is a list of differences (insert or remove) between the texts' sentences.

- **`SentenceDiff`**: domain object responsible for storing the sentence data (*i.e.*, String) that was changed, the type of operation performed (insert or delete) and the index where the operation should be performed in the original text.

- **`SentenceOperationType`**: enumeration (enum data structure) which lists the supported operations. Current values are: insert and delete.

3.4.2 Updating the original process model

After having identified the sentences which must be synchronized with the original process model (*e.g.*, removed or inserted), the process information is derived from the text sentence using the techniques described in Section 3.3.3. Afterwards, with the *SentenceText* instance, the process' properties are extracted and passed to the next module, which is responsible for updating the *ProcessModel* object and all the related objects instances, if necessary. Three specific scenarios for updating links[6] between process model elements exist. Table 3.6 depicts these scenarios, which are roughly the same as inserting elements into a linked list data structure [25].

Table 3.6: Overview of Operations for Updating Links

Element Position	Result (Delete Operation)	Result (Insert Operation)
First (*e.g.*, index=1)	Before: A -> B After: B	Before: A -> B After: A' -> A -> B
Middle (*e.g.*, 1 < index < n)	Before: A -> B -> C After: A -> C	Before: A -> B -> C After: A -> B -> B' -> C
Last (*e.g.*, index=n)	Before: A -> B After: A	Before: A -> B After: A -> B -> B'

The first scenario consists of inserting a new element as the first element within the process pool. In this case, all the elements must be shifted forward. This is done by adding a link between the new element and the previous first element. To illustrate this strategy, suppose that you have three activities: A -> B -> C. A new activity "*NEW*" must be inserted into the first position. In order to do so, the first activity must be located. In this example, the first activity is represented by "*A*". Then, a link is created between the "*NEW*" activity and the first activity "*A*". By doing so, activity "*NEW*" has become the first activity from the sequence, as follows: NEW -> A -> B -> C.

The second scenario consists of inserting a new element between the first and last element within the process pool. To illustrate this strategy, suppose that you have three

[6] In this case, links represent the sequence flow elements (arrows) that connects each process model element to the next one.

activities: A -> B -> C. A new activity "*NEW*" must be inserted into the second position. Then, the activity located into the previous position must be found. In this example the previous position corresponds to the first, which is occupied by activity "*A*". Activity "*A*" is linked to activity "*B*" (A -> B), but now it must be updated so that its link points to the "*NEW*" activity (A -> NEW). Afterwards, the "*NEW*" activity link is created pointing to activity "*B*" (NEW -> B). The result would be four activities: A -> NEW -> B -> C.

The third scenario consists of inserting a new element as the last element within the process pool. In this case, all the elements must be shifted backward. This is done by adding a link between the new element and the previous last element. To illustrate this strategy, suppose that you have three activities: A -> B -> C. A new activity "*NEW*" must be inserted into the last position. First, the last activity must be located. In this example, the last activity is represented by "*C*". Then, a link is created between the last activity "*C*" and the "*NEW*" activity. By doing so, activity "*NEW*" has become the last activity of the sequence, as follows: A -> B -> C -> NEW.

Reflecting deleted elements to the original model Reflecting deleted sentences, that represent activities, back to the original model is quite straightforward. Through the process elements properties (obtained from the *SentenceText*), the algorithm searches for the respective process element in the original model. When the original element is found, it is removed from the model and the links are updated. If no such element is found, then the operation was performed successfully. To remove only a business object from an activity, it is necessary only to remove the business object from the sentence and not the whole sentence itself (otherwise it would remove the whole activity). By doing so, the algorithm is capable of updating the original sentence by removing the business object from the activity's label.

Note that to remove a role (*i.e.*, actor) from the original model, it is necessary to remove all the references (activities) to it. This is done by removing all the sentences which the role is the main object or by replacing it by another role, which would transfer all the activities to this second role and remove the role's lane from the original model. The example below illustrates this scenario:

1. **Original Text:** `The process begins when the secretary answers the call. Afterwards, the customer requests the order status. Then, the intern checks the order status. The secretary informs the order status. Afterwards the intern writes down the message. Finally, the process`

`finishes.`

2. **Changed Text:** `The process begins when the secretary answers the call. Afterwards, the customer requests the order status. Then, the secretary checks the order status. The secretary informs the order status. Finally, the process finishes.`

3. **Updated Model:** Figure 3.18 illustrates this procedure. In this example, the role's references were removed by replacing the first occurrence of the role "*intern*" by "*secretary*" and removing the remaining activity, which the intern participates.

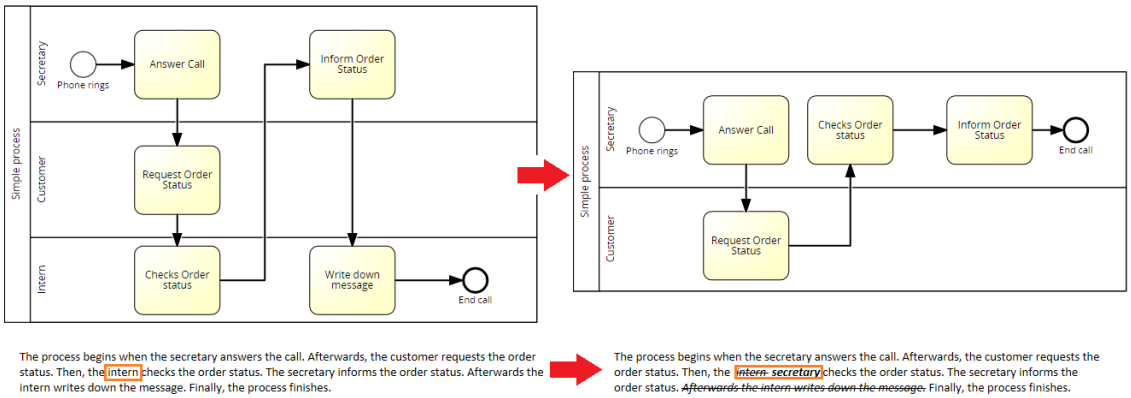


Figure 3.18: Original process model and its updated version, after having textual changes reflected to it.

Reflecting deleted sentences that represent gateways back to the original model is a little bit tricky. Through the process elements properties (obtained from the *SentenceText*), the algorithm searches for the respective process element in the original model. When the original element is found, it is removed from the model. Until this point, it is basically the same as removing an activity. But, the problem arises when the links must be updated. A gateway represents a control-flow, thus removing the gateway, the control-flow is removed. As a result, instead of having activities executed in parallel or executed under certain conditions, the related activities must appear sequentially after removing the gateway. To solve this problem, a simple criteria was adopted. The first sentence, related to the gateway, takes the gateway position in the original model, its link is updated to the next related sentence and the next sentence's link is updated to the next and so on. The example below illustrate this scenario.

1. **Original Text:** `The process begins when the Room-Service Manager takes down order. Then, the process is split into the 3 parallel branches:`

- `The Room-Service Manager gives order to the Sommelier.`
- `The Room-Service Manager submits the order ticket to the kitchen.`
- `The Room-Service Manager assigns order to the waiter.`

2. **Changed Text:** `The process begins when the Room-Service Manager takes down order. The Room-Service Manager gives order to the Sommelier. The Room-Service Manager submits the order ticket to the kitchen. The Room-Service Manager assigns order to the waiter.`

3. **Updated Model:** Figure 3.19 depicts the original and the updated version of the model after the text changes were automatically reflected to it.

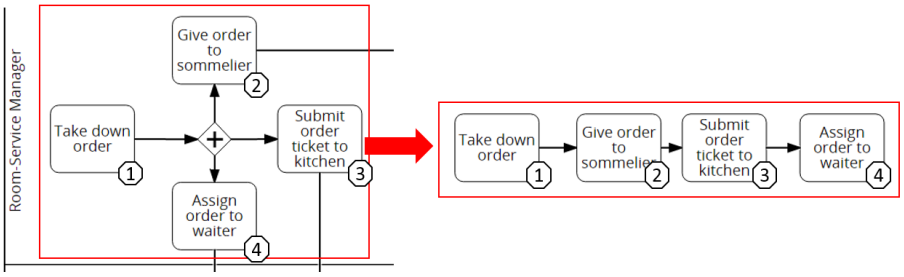


Figure 3.19: Original process model and its updated version, after removing a gateway sentence from the text.

Reflecting inserted elements to the original model Reflecting new sentences, back to the original model is not so straightforward as deleting. First, the new process model element is created from the sentence. Then, its link reference must be set to the next process model element. The question that arises is to find the exact position where the element should be inserted. This is done through the text analysis performed by the *DiffAnalyzer*. As can be seen in Section 3.3.2, the text structure makes possible to map the activity sequences and all the related elements in the correct order. The algorithm is capable of positioning the element into the correct lane (role) by querying the sentence's business actor. Figure 3.20 illustrates this procedure.

If the sentence does not represent an activity, a simple criteria is applied: the element is disposed into the previous element's lane. Figure 3.21 illustrates this procedure. If there is not any previous element, then it is randomly allocated into one of the available

lanes. The exception is for gateway elements. Based on process modelings guidelines, this kind of process model element should not be the first process element [7]. These scenarios are thus considered as modeling errors by the framework and automatically rejected. An error message is shown for the user suggesting that gateways should not be used as the first process element.

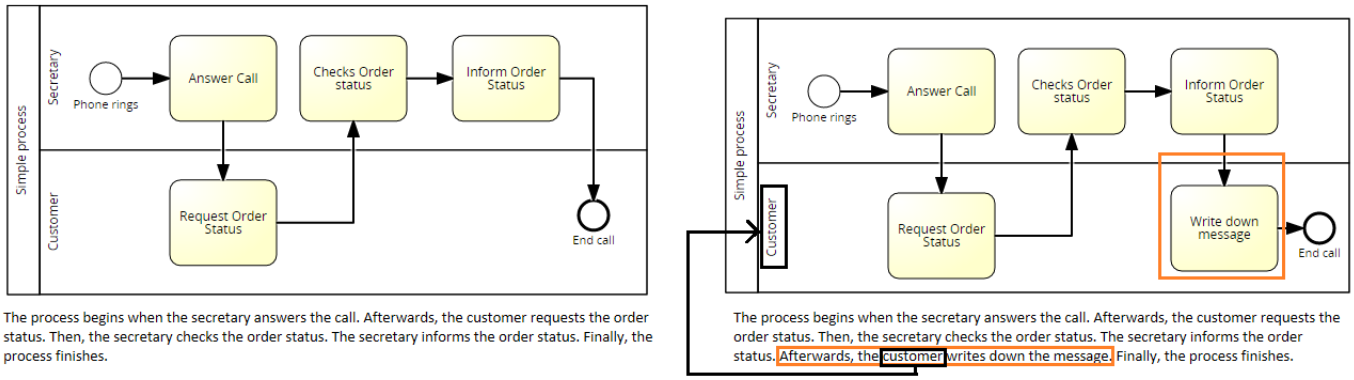


Figure 3.20: Activity Insert: Original process model and its updated version, after having textual changes reflected to it.

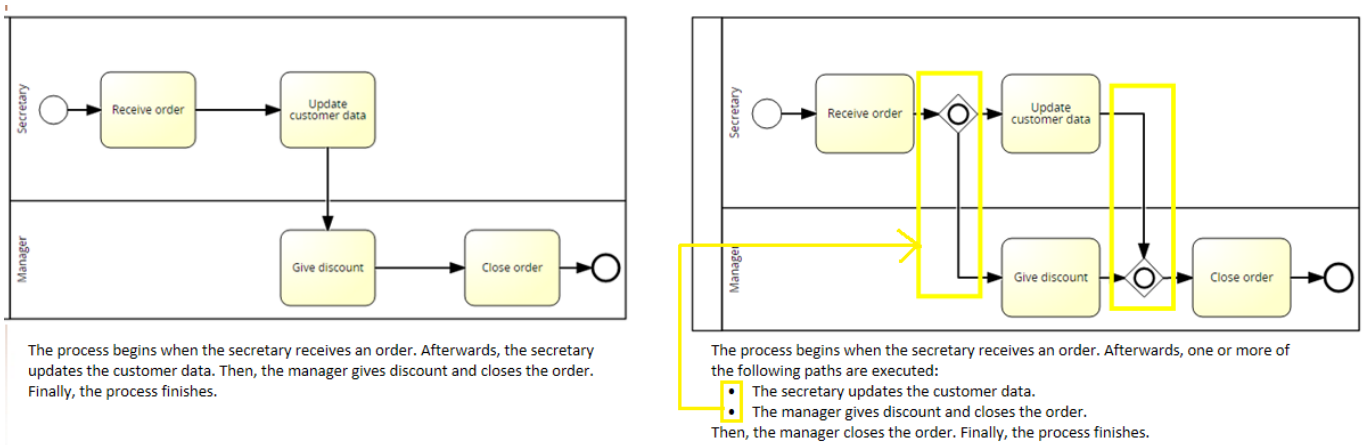


Figure 3.21: Gateway-And Insert: Original process model and its updated version, after having textual changes reflected to it.

3.5Framework's activities sequence

Apart from some intermediate auxiliary components, Figure 3.22 shows the sequence of activities executed by the system during the process of generating text from a business

[7]It is considered a good practice always to start the process with a begin event element. The *begin event* element should always be the first element within the process pool/lane.

process model. The diagram steps are detailed as follows:

- Step 1: The sequence is triggered when a user, with some file that represents a business process model, asks for the system to read it. In this work, the JSON extension was chosen to be the in-memory representation of the process model. In this moment, an object from the JSONReader class is created, which main responsibility is the transformation of the information within the JSON file in some compatible process model structure. The object returns a process model object.
- Step 2: With the model, the user asks for the transformation of this model in its respective natural language text form. This is done through a static class responsible for calling, sequentially, all the NLG components present in the pipeline architecture.
- Step 3 and 4: The language of the model is read from an attribute of the process model that identify the language used in its construction. As soon as the language is detected, the specific linguistic component is activated (Step 3), initializing all the objects with their respective implementations to treat the current language, as well as the necessary configuration for the localization component (Step 4). The localization component is a module responsible for the translation of the words and specific sentences, saved as generic keys of the dictionary. The keys are stored as a set of enumerations (enum data structure), where each item must have its respective translation for the current language. In order to do that, the user must store the translations of each given key in some external file. For example, regarding the Portuguese language, when the key ”PROCESS_BEGIN_WHEN” is detected, the translated message “O processo começa quando” is returned.
- Step 5: Afterwards, it is necessary to transform the current process model structure in some intermediate graph-based structure, where each arc is transformed to an edge and each node is some process element, such as event, gateway, task, etc.
- Step 6: The RPST tree is created from the graph with all the necessary linguistic information in order to generate the text.
- Step 7: After that, the root of the tree is used as starting point to the generation of all the DSynt trees that will represent, in a textual way, the several activities and components (gateways, events) from the process model. When this step is finished, we already have some textual representation of the process model. But, this representation is yet to be improved (in order to be readable), due to some grammatical issues and redundancy.

- Step 8: The first phase in the treatment of the text is the aggregation of the messages. This component returns the messages grouped by some criteria, for example, performed by the same actor (role) in some strict sequence.

- Step 9: The second phase corrects some of the problems resulted from the aggregation of the messages through a component that inserts referring expressions. As an example, it executes the replacement of the repeated role name (actor) in the same sentence for its respective pronoun, in order to obtain a better readability.

- Step 10: The third phase is responsible for the addition of sequence connectives to the several sentences to connect them semantically within the text. For example, the sentences “Garçom entregar pedido.” and “Cozinha preparar comida.” would be connected through the expression “Em seguida”. “Garçom entregar pedido Em seguida Cozinha preparar comida”. Note that the use of articles, punctuation and verb conjugation is incorrect. This will be solved in the next step.

- Step 11: The fourth and last phase has as main goal correct the grammatical errors present in the resulting sentences from the transformations, in order to generate a fluid and coherent text. In this component, occurs, for example, the transformation of the sentence “Garçom entregar pedido. Em seguida, cozinha preparar comida” into “O garçom entrega o pedido. Em seguida, a cozinha prepara a comida”. After the correction of all the messages within the Dsynt trees has been done, the component performs the concatenation of the messages and returns a text to the user, that represents the process model in natural language text.

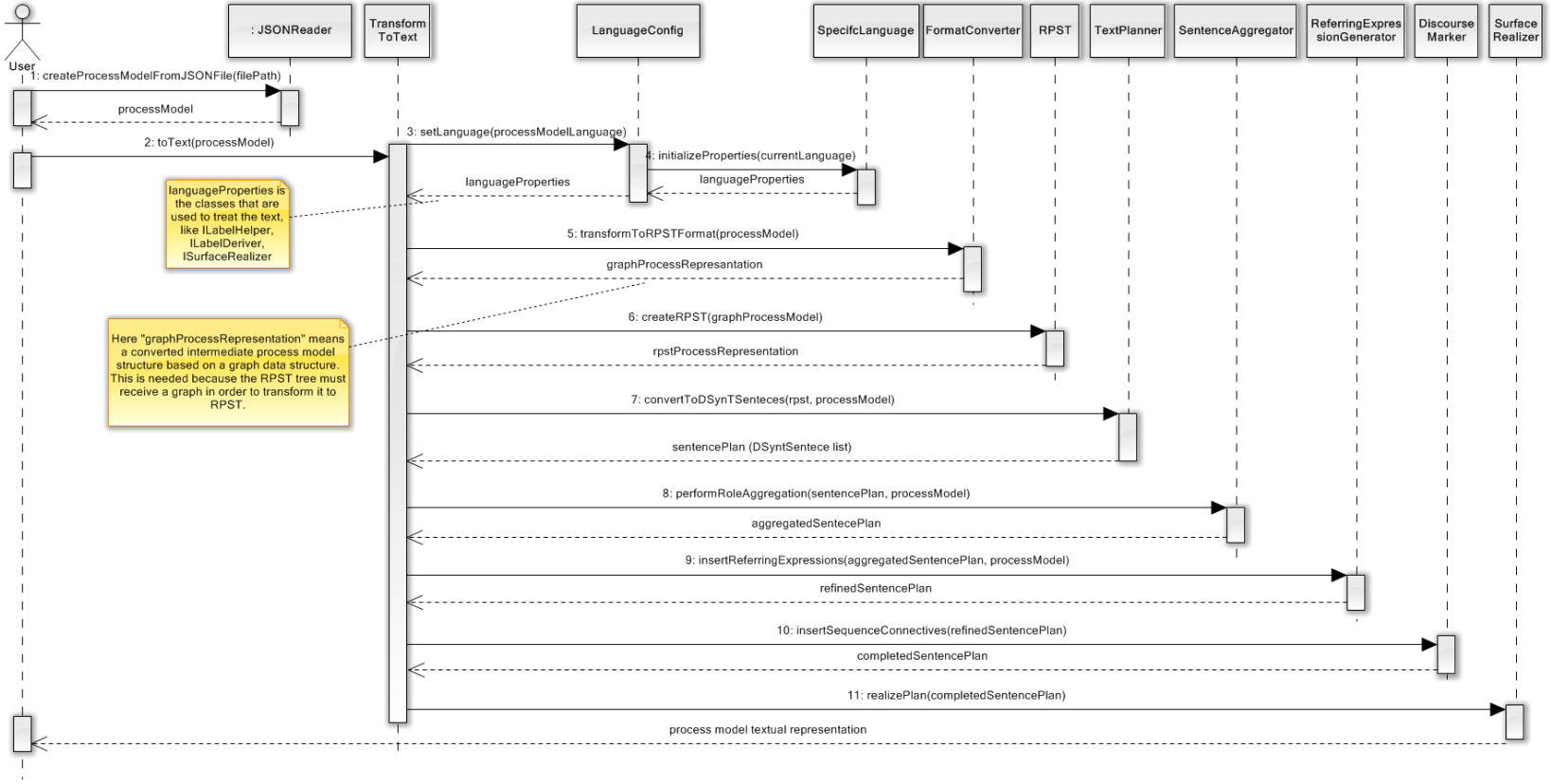


Figure 3.22: Sequence diagram representing the steps to generate text from model.

Apart from some intermediate auxiliary components, Figure 3.23 shows the sequence of activities executed by the system during the process of updating a business process model through a natural language text. The update process starts with the user searching for a business process through its respective name. If the process is found, then he must choose whether he wants the process's text version or the model version. After choosing the text version, the user update the text and submits the updated text as input for the system. Then, the system starts executing the following steps:

- Step 1: A new **RealizedText** is created. A query for the original text is submmited to the database and both, the original and the updated text, are stored within the new **RealizedText** object.
- Step 1.1: The method *cleanText* is called to begin with the pre-processing phase, which will remove any unnecessary data from the text.
- Step 1.2: The *removeFormatting* method from the **ITextFormatter** concrete implementation is called. This method recieves a plain text as input and returns the text without any formatting character.
- Step 1.3: The *removeStopWords* method from the **ITextFormatter** concrete implementation is called. This method recieves a plain text as input and returns the text without any unnecessary words (*i.e.*, stop words).
- Step 1.4: For each sentence stored within the **RealizedText** object, the system calls the *identifyChanges* method to assert whether or not that sentence was modified compared to the original sentence. During this step, the system also identifies the operation type (edit, add or delete). In particular, each sentence is stored in a **SentenceText** object.
- Step 1.5: If the sentence has been changes, the operation type is setted for that particular sentence.
- Step 1.6: With the operation type already configured, the system calls the update method.
- Step 2: After the update method from the **SentenceText** object is fired, a new instance of the **NaturalLanguageProcessor** is created for the current language (*e.g.*, English, Portuguese, German etc.).
- Step 3: Afterwards, the main NLP techniques are used to (i) identify which BPMN element(s) does the sentence represent(s), and (ii) extract and store the BPMN element's specific properties (as described in Section 3.1.1).

- Step 4 and 5: The BPMN properties are obtained through the **NaturalLanguageProcessor** and is then passed to the update method of the **DSnyT** data strcuture.
- Step 5: After the DSynT's *update* method is triggered, it fires another *update* method for the **ProcessModel** object. At this moment, the *update* method called is specific to each BPMN element type (*e.g.*, Activity, Gateway, Event etc).
- Step 6: Finally, after updating all specific elements, the whole process model is also updated. At this moment, both text and model, are synchronized regarding business process information.

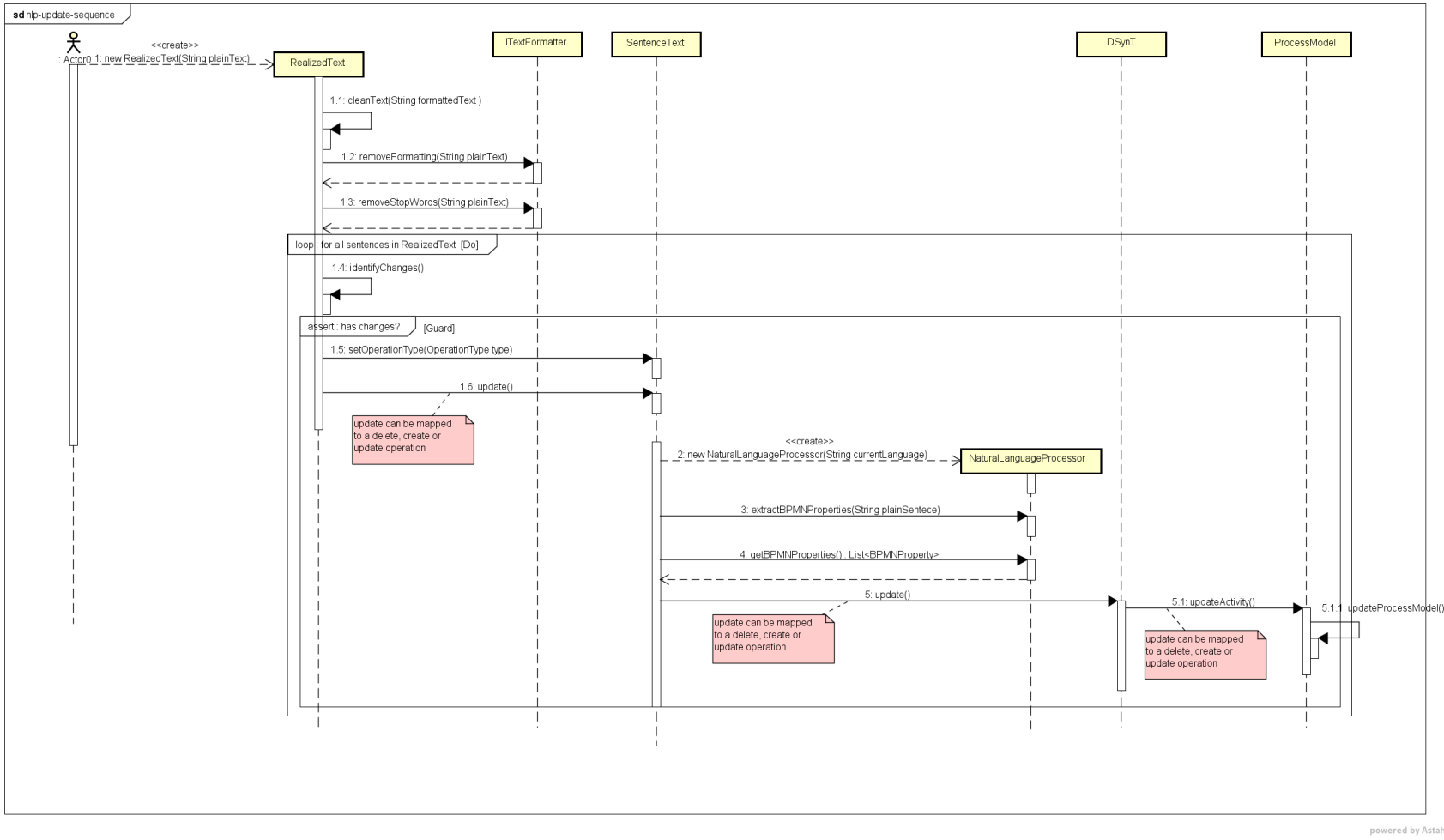


Figure 3.23: Sequence diagram representing the steps to update the model from changes in its text representation.

3.6Chapter Summary

This chapter presented the details about the proposed `round-trip` technique, explaining how it was implemented within a language-independent framework. The technique makes use of several NLP and NLG techniques, applied in a specific sequence through a pipeline structure. The NLG pipeline is used to generate textual representation from a process model, while the NLP pipeline is triggered to generate process model from natural language text. With the definition of a synchronization component, capable of identify and classifying textual changes within the text according to the set of basic change operations mapped by Weber *et al.* and Kolb *et al.*, it was possible to run the complete `round-trip`. The following chapter describes the `round-trip` technique evaluation through controlled experiments and a case study.

4. Evaluation

This chapter presents the evaluation of the methods developed and implemented within the language-independent framework. One evaluation was design and executed by the authors with a small set of business process models to evaluate the framework's behaviour. Afterwards, some research questions were defined to serve as a guide during the execution of an experiment, which objectives were to validate and evaluate the natural language text produced as the framework output when given a process model instance as input. Basically, it was an exploratory research to investigate weather the generated text is capable of transmitting the same knowledge as compared with a Business Process Model. Finally, a second experiment was run to evaluate the synchronization components through editions made to the original text and asking the business process expert to evaluate whether the changes were reflected correctly to the original process model.

4.1PoC: Proof-of-Concept

This evaluation aimed at validating the whole round-trip implemented within our framework. In other words, we were interested in automatically generating process textual descriptions from process models and asserting that that the original process model could be updated by submitting text-based changes to the synchronization component.

Basically we conducted a two-step evaluation. At first, artificial data were generated, which represented an experiment. This experiment was composed by ten (10) exclusively made-up models[1] (designed by the authors) to stress specific conditions (*i.e.*, scenarios) that should be covered by our solution. For example, the framework should be able to treat all the BPMN elements defined in our subset (Figure 2.3) and support all change operations presented in Table 3.5. The framework was designed to support multiple languages, thus the set of process models were written in both English and Portuguese.

[1] This set of process models (artificial data) are available in the Appendix B

With this strategy, it was also possible to test if the framework could deal with a dynamic language change in the process model received as input during execution time, initializing the necessary configurations to trigger the right implementations to extract the process semantic within the model. The language detection is identified through a meta-data in the process model, which informs the language used to design it. The tool's behavior was evaluated and the system was observed in different scenarios: process models composed by activities without any control flow logic and process models composed by activities with control flow logic (gateways AND, XOR and parallel). These models presented variations of important characteristics for the evaluation, which were:

- **Activities being executed in sequence by the same actor (role).** These activities should activate the module of "*Referring Expression*" and "*Message Aggregation*" defined in Section 3.2.2. For this reason these activities are essential to illustrate the framework usage.
- **Activities without roles (empty lane).** These activities are justified by the need to test if the algorithm responsible for treating activities without actors are correctly mapped to a textual description using passive voice.

After the initial evaluation with artificial data, real data was used through a second set of business process models[2] gathered from universities and from companies to stress real scenarios. This second set was composed by twenty (20) process models written in Portuguese. Thus, for each model an English translated copy was manually generated for testing the language-independence framework's feature.

In total, thirty (30) process models were used during the PoC evaluation, twenty (20) from real case scenarios and ten (10) using artificial data. The overall characteristics of these models are presented in Table 4.1. It is important to highlight that these characteristics are shared by both, models and textual descriptions.

Table 4.1: Overall characteristics of the complete test data set.

Metric	MIN	MAX	Average
Number of Actors	1	6	2,96
Number of Activities	2	40	11,68
Number of Control Flows	0	10	3,48

As can be observed, each process contained at least one (1) actor and no more than six

[2]Due to copyright reasons, the whole set is not available. Nevertheless, some process models are available in the Appendix B

(6) actors. In average, the whole data set contained 2,96 actors per process. Analogously, each process contained at least two (2) activities and no more than forty (40) activities. In average, the whole data set contained 11,68 activities per process. Finally, not all processes had control flow logic and had no more than ten (10) control flows within the same process. In average, each process contained 3,48 control flows.

Figure 4.1 depicts the methodology used for this evaluation (*i.e.*, test) which is detailed as follows:

1. The test begin by fetching a process model from our repository (step 1).
2. The process model is submitted as the framework's input which generates a textual description as its processing output (step 2). During this step, the output text was compared with the model used as input and a set of text metrics was used to investigate in how far the generated texts are comparable to manually created texts. The main component tested in this step is the NLG Core module.
3. If any errors were found then the test was interrupted and a new version was developed addressing these errors. After applying the correction (step 3), the process model was resubmitted (step 2) to the framework. This procedure was repeated until no critical error could be found. Before continuing to step 4, the generated process textual description was submitted as the framework input to assert whether the generated process model would match the original.
4. After asserting that no critical error could be found within the generated process textual description, text-based operations were made to the generated text (step 4) which was then submitted as the framework's input (step 5). These operations are better illustrated in the experiment described in Section 4.3.1.
5. The updated text triggers the synchronization component which generates an updated version of the original process model based on the detected text changes (step 6). The new process model was analyzed to assert if any critical error could be found. The main component tested in this step is the NLP Core module.
6. This step is very similar to the third (3) step. If any errors are found then the test is interrupted and a new version was developed addressing these errors. After applying the correction (step 6), the changed text was resubmitted (step 5) to the framework. This procedure was repeated until no critical error could be found. Before continuing to step 7, the updated process model was submitted as the framework input to assert whether the automatically generated process textual description would match the original.

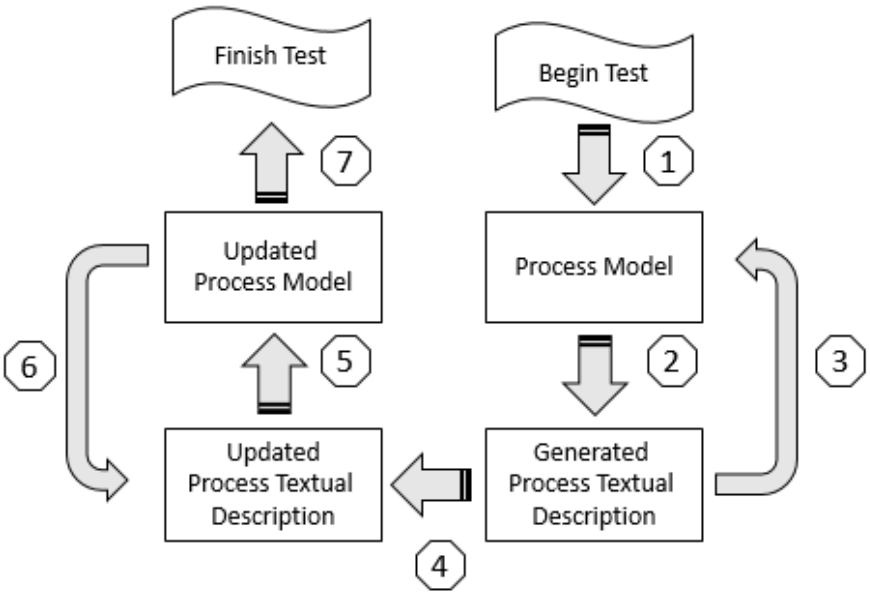


Figure 4.1: Evaluation methodology used for the Proof of Concept.

7. Finally, after asserting that no critical error could be found within the updated process model, the test was considered to be concluded successfully (step 7).

Another aspect to be taken into consideration is the evaluation of the framework regarding its capacity of extension to other languages. The necessary process to include a new language, to treat business process models should be easy, fast and clear. With the language-independent framework's architecture, a lot of complex and common aspects were abstracted, like the text planning phase and sentence planning phase. They do not need to be modified, decreasing the necessary time and maintaining the focus into the implementation of the operations defined in the interfaces, responsible to treat the languages. Appendix C details the necessary steps that must be followed to add support for new languages.

The quality of the generated text is directly dependent on the quality of the implemented operations for the current language. The Portuguese language does not yet has a free and powerful Java library work with its linguistics aspect, as the English language has WordNet. As a result, the coverage for Portuguese semantic and syntactical aspects is not the same as for English.

The framework had a positive general evaluation, being capable of both: correctly generating natural language texts from business process models used as input and generating process models from natural language texts. The framework was able to map and treat all the components in the models, as well as their respective notations and labels used to

their description. The NLG and NLP pipeline process (Figure 2.10 and 3.12, respectively) were followed without the detection of any error that could compromise the execution of the process. However, some problems were found:

- The algorithm responsible for the activation of the referring expression and message aggregation modules failed to identify correctly a sequence of activities performed by the same actor. As an example, in the process model (Figure 2.4) used as input, we can check in the generated text (Table 4.2) that the word "*waiter*" is frequently used in the last paragraph. In this case, it should be replaced with the pronoun "*he*".
- The coverage for business process in Portuguese is smaller than for English. This is due to the fact that the English is more used globally and, as a consequence, there are more resources and features for its linguistic manipulation.

Table 4.2: Natural Language Text generated by analyzing the process model data and extracting textual information.

The process begins when the Room-Service Manager takes down an order. Then, the process is split into 3 parallel branches: • In case alcoholic beverages are ordered, the Room-Service Manager gives order to the Sommelier. Afterwards, one or more of the following paths are executed: – The Sommelier fetches wine from the cellar. – The Sommelier prepares the alcoholic beverages. • The Room-Service Manager submits the order ticket to the Kitchen. Subsequently, the Kitchen prepares the meal. • The Room-Service Manager assigns order to the Waiter. Then, the Waiter readies the cart. As long as all the 3 branches were executed, the Waiter delivers to the guest's room. Afterwards, the Waiter returns to the room-service. Subsequently, the Waiter debits from the guest's account. Finally, the process finishes.

We can state that the first problem does not affect the correct understanding of the generated text. It affects only the quality of the generated text, because the reading of one big sentence made up of several activities is more tiresome but still, understandable. As we can see in the result of the process model used as input (Figure 2.4), we can understand the text perfectly, and which information the model transmits.

As a result, we got evidences that our approach is capable of dealing with the dynamic language change of the process model in execution time, automatically initializing the necessary configurations and correctly generating a natural language text from the model and also applying text-based changes back to the original model, through the synchronization component.

4.2Experiment: Business Process Automated Description Quality

This experiment evaluates the text generated by the framework. It was an exploratory research to investigate whether the generated text is capable of transmitting the same knowledge as compared with a Business Process Model.

4.2.1Experiment Design

This section presents the design of the proposed experiment, including our research questions, the instruments selected to address these questions, the participants, and the measurements taken to: (a) Analyze the equivalence between textual description and process models; (b) Analyze the textual description quality according to the subject's perspective; and, (c) Compare how does the experience influences the comparison between the textual description and the process model.

Research Questions The main objective of this experiment was *Assess whether the knowledge represented by the generated process description (i.e., textual work instructions) can be considered equivalent to the process model.* Two research questions were proposed to address this issue:

1. Is the knowledge represented by the natural language text, generated by the framework, equivalent to the process model?
2. Can the natural language text, generated by the framework, need to be enhanced to achieve better understanding?

Instrumentation An online questionnaire[3] was used to collect the data for this experiment analysis. The questionnaire was composed by: (i) a set of questions to characterize

[3]The questionnaire is available at https://docs.google.com/forms/d/1zvuuxojWUVWnyRpMsao-F1Yjygs-Dm2ebfrMjjjJqx4/viewform

participant experience in process modeling; and, (ii) a set of seven (7) Text-Model pairs (*i.e.*, seven BPMN models and textual work instructions presented in pairs) describing process fragments, followed by three questions. The first question's objective was to rate the equivalence between the textual work instruction and the BPMN model. The second question aimed at evaluating the text quality, varying from *very good* to *very bad*. The third question was optional and allowed participants to enter a free text regarding their impressions about the generated textual work instruction.

Process fragments were chosen instead of whole processes in order to minimize the time required for the participants to fill up the whole questionnaire. We believe a long time response questionnaire would lead to fewer participants and could compromise the study quality (*i.e.*, whole process could not be comprehensible in a short time span).

The experiment was executed in several sessions, each involving a subset of our participants. It is important to highlight that the experiment objective was not described for the participants nor did they know about our research, thus avoiding possible bias while answering the questions. The overall structure of a session was comprised by four (4) steps. First, the questionnaire was electronically sent through e-mail. Next, the questionnaire was filled by the participants. Afterwards, the answers were collected and stored in the database. Finally, the answers were analyzed and documented. No time restriction was imposed to participants, either for analyzing the process fragment or for filling up the questionnaire.

In any experiment, the results achieved after the execution phase are dependent on how the data collection instruments are created. For this experiment, the instrument was based on a digital questionnaire. This questionnaire was created in accordance to several guidelines in order to reduce the chances of introducing poorly phrased questions [7, 9, 99]. Special attention was paid when creating the set of questions because of the awareness that answers to poorly phrased questions may be worse than meaningless: they may be misleading.

The characteristics of the used instrument's elements are described below. They were originally written in Portuguese and all participants spoke Portuguese as their native language. For the purpose of consistency with the remaining text, we present English translated versions of the original elements.

Participants Characterization Questions: This set of questions aimed at identifying the profile and experience of the participants with process modeling and the BPMN notation. It was composed by four close-ended questions presented in Table 4.3. Questions

were answered according to a 5-point ordinal scale (*i.e.*, likert scale) list. These questions addressed whether the participant was familiar with process modeling and BPMN notation. These questions were elaborated to obtain detailed and specific information about the participant's experience related to process modeling, avoiding subjective answers. The ordinal scale was used to specify the participant's level of agreement or disagreement with a series of statements about business process modeling. This way we could assert the participant's confidence about a particular topic related to process modeling. Ordinal scales (usually with five or seven symmetrical points) are frequently used in survey research and have been shown to reliably measure mental effort [10] [91].

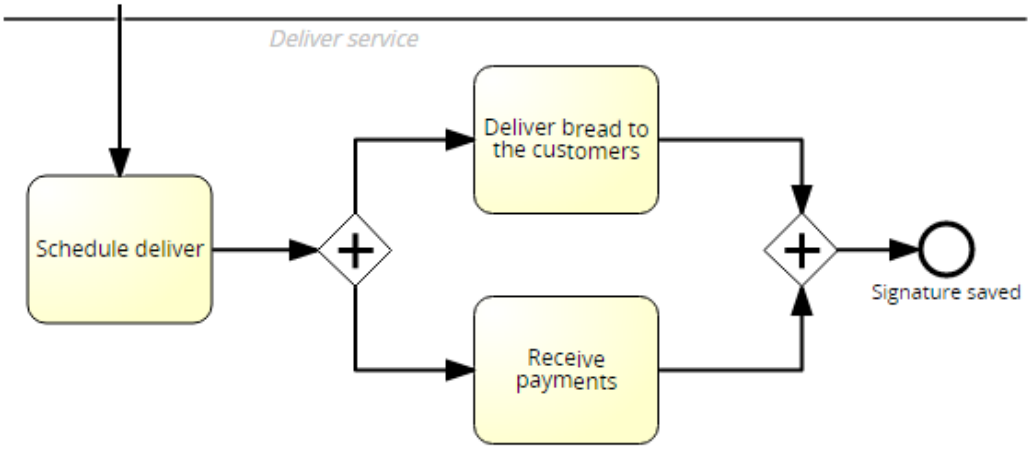
Process Fragment: Text-Model Pair. The text-model pair aimed at rating the equivalence in terms of the information transmitted by both. Each pair was composed by a BPMN model and textual work instructions generated by the framework. Figure 4.2 depicts one exemplary of the text-model pair used by this experiment. A process fragment was presented in order to minimize the time and to abstract the need for understanding the whole process semantic. Hence, isolating any domain knowledge that the participant might have about the process. Thus, the user can focus on short descriptions and fewer symbols within a BPMN model. The process fragment was accompanied by three questions. The first question was answered according to a 5-point ordinal scale. This question evaluated whether the participant consider both knowledge representations are able to transmit the same information about the process fragment. The ordinal scale was used to specify the participant's level of agreement or disagreement with the following statement "*Both, text and model, are considered equivalent in terms of the process which they describe.*". The second question was also answered according to a 5-point ordinal scale and aimed at evaluating the textual description quality. Finally, the third question was optional and answered as a free text. This question aimed at gathering qualitative data to enable an open exploratory research about the comparison between the text-model pair.

Participants Several instances of the same instrument (described in the previous section) were given to the participants. In total, 66 participants were selected to participate (9 students, 8 professors and 49 practitioners). Students and professors were from the following universities: Federal University of the State of Rio de Janeiro (UNIRIO), State University of Rio de Janeiro (UERJ) and Federal University of Rio de Janeiro (UFRJ)). Practitioners were from different IT companies located in Rio de Janeiro (Brazil). For the purpose of analysis, a more complete classification of participants into experienced or non-experienced, based on all answers given to the characterization questions (see table 4.3), was developed and used.

Table 4.3: Questions about the participant's experience with process models (characterization).

Questions and Options
1 - Overall, I am very familiar with process modeling.
a. Strongly disagree | b. Slightly disagree | c. Neutral | d. Slightly agree | e. Strongly agree
2 - Overall, I am very familiar with the BPMN notation.
a. Strongly disagree | b. Slightly disagree | c. Neutral | d. Slightly agree | e. Strongly agree
3 - I feel very confident in understanding BPMN process models.
a. Strongly disagree | b. Slightly disagree | c. Neutral | d. Slightly agree | e. Strongly agree
4 - I feel very competent in using BPMN for process modeling.
a. Strongly disagree | b. Slightly disagree | c. Neutral | d. Slightly agree | e. Strongly agree

Fragment B of a process model and its respective textual description



The Delivery Service schedules the delivery. Subsequently, the process is divided into two parallel branches:

- The Deliver Service delivers the bread to the customers
- The Deliver Service receives the payments

Once all the branches are executed, the process finishes with the signature saved.

Figure 4.2: A text-model pair describing a process fragment.

Measurements Results were gathered through the answers given by the participants in the questionnaire: for each rating (*i.e.*, 5, 4, 3, 2, 1) we counted the total number of answers for the same rating. For example, we counted how many times option 4 was chosen, how many times option 3 was chosen and so on. The instrument presented seven distinct process fragments, with the same question accompanied by the same number of options available to rate the equivalence between the process textual description and the process

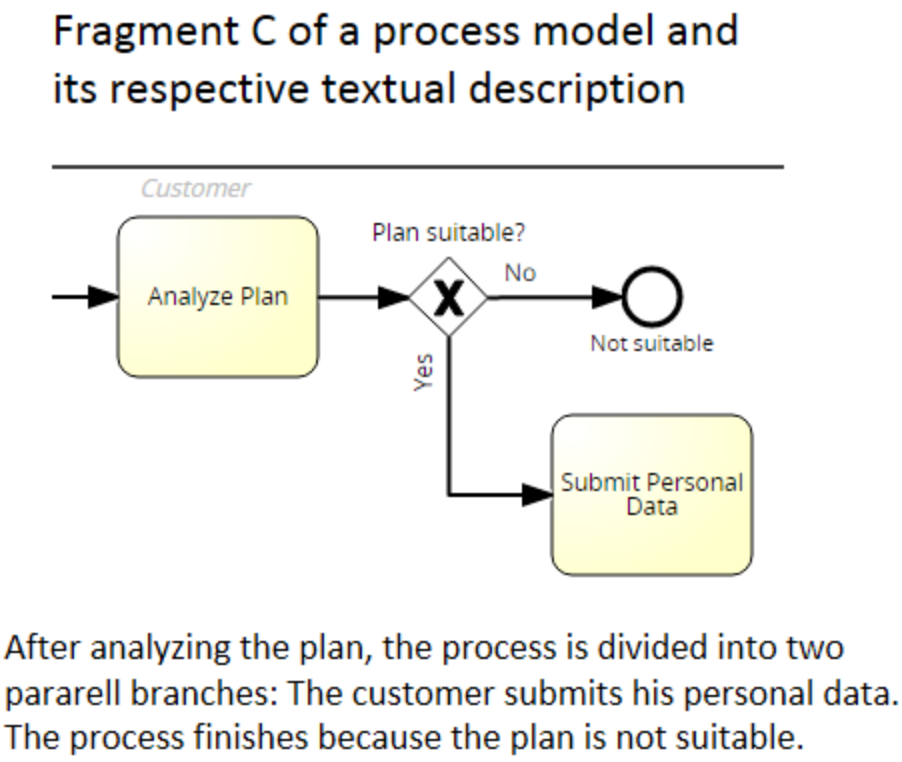


Figure 4.3: A text-model pair describing a process fragment, which is not equivalent.

graphic model (BPMN). Thus, this give us a total of seven (7) answers per participant. As we have 66 participants, this is equal to 462 (66 x 7) answers considering all process fragments.

Process Equivalence Measurement (RQ1): Our accuracy was measured by the number of correct answers. Due to the text being sensible to an open interpretation (it is not represented by a formal representation language, such as BPMN, thus may generate ambiguity), we did not expected both representations to be 100% equivalent. Instead, we claim a threshold varying from 100% to 70% can be considered as a good result for the sake of this analysis.

We sum the quantity of answers given to the same option, regardless of the process fragment being analyzed. This allows us to have a general overview of the evaluation. Nevertheless, a secondary analysis, which considered answers only to the same process fragment was made and is described in details in Section 4.2.2. In the instrument, "*Totally disagree*" and "*Totally agree*" options were equivalent to extreme points (0 and 100, respectively). We grouped the answers for options "*Totally disagree*" (0% equivalent) and "*Slightly disagree*" (Equivalence between 1 and 33%) into one, which give us an equivalence rating ranging from 0 to 33%. Analogously, we did the same for the options "*Slightly agree*" (Equivalence between 68 and 99%) and "*Totally agree*" (100% equivalent), which

Table 4.4: Question about the equivalence between the textual work instruction and the BPMN model, which describes a process fragment.

Question and Options
Both, text and model, are considered equivalent in terms of the process which they describe.
a. Totally disagree (0% equivalent)
b. Slightly disagree (Equivalence between 1 and 33%).
c. Neutral (Equivalence between 34 and 67%).
d. Slightly agree (Equivalence between 68 and 99%).
e. Totally agree (100% equivalent).

Table 4.5: Question about the textual work instruction quality, which describes a process fragment.

Question and Options
How would you rate the quality of the textual description, varying from 1 to 5 (where 1 stands for horrible and 5 excellent)?
1. (horrible) | 2. (very bad) | 3. (acceptable) | 4. (very good) | 5. (excellent)

give us an equivalence rating ranging from 68 to 100%. Doing so, we shorten our analysis into three groups. The optimal scenario would be all answers (from the total of 462 answers) were within the first group (which vary from 100% to 68%) and the worst case scenario would be all answers were within the third group (which vary from 33% to 0%). The counting process was performed by a specialized software from Google, known as Google Forms [4].

The third text-model pair used in the instrument was designed in order to be not equivalent. So, for this particular fragment, we expected more participants would answer that both representations were not equivalent regarding the knowledge which they represent.

Textual Description Quality Measurement (RQ2): Our expectation (*i.e.*, the optimal scenario) was that the number of answers for the textual quality description between 5, 4 and 3 (inclusive) be higher than the number of answers between 2 and 1. The worst case scenario would be the reverse, with more answers between 2 and 1 than answers between 5, 4 and 3. To be more more precise, we expected that at least 70% of the answers were within the first group (answers between 5, 4 and 3), which we defined as a threshold for a great result.

[4]For more information about Google Forms refer to https://www.google.com/forms/about/.

To measure the result for RQ2, we applied the same strategy used to measure RQ1 (*i.e.*, sum the quantity of answers given to the same option). Thus, to enable a better reading of the results, we grouped the answers for options "*Very Bad*" and "*Horrible*" into one group. All the remaining answers (options "*Good*", "*Very Good*" and "*Excellent*") were grouped into a second group. We considered the reverse strategy for the third fragment, which had errors inputted on purpose. So for this case, the sum of answers between 1 and 2 should be higher than the sum of answers between 3 and 5. Analogously, we did the same for answers between 3 and 5, which was grouped together with answers between 2 and 1 from the others fragments. In other words, for the third fragment we mapped "*Horrible*" and "*Very Bad*" to "*Excellent*" and "*Very Good*", respectively.

4.2.2Analysis and Discussion

This section presents the compiled results of the analysis of the data gathered. It is divided into four subsections, which address the research questions defined in the experiment design (Section 4.2.1) and discuss threats to the validity of the proposed experiment.

Overall Evaluation To address RQ1 (Section 4.2.1), we were interested in determining how many answers were within the equivalence range varying between 100 and 68%, how many were between 67 and 34% and finally how many were between 33 and 0%. Our expectation was that the number of answers between 100 and 68% were higher than the sum of the other two groups (participants who rate equivalence between 67 and 0%), except for the third process fragment, which the expectation was the reverse (number of participants that rate equivalence between 67 and 0% higher then the number of participants who hate equivalence between 100 and 68%). Figure 4.4 depicts the overall evaluation for this question.

As can be observed, 342 answers were within the equivalence group ranging from 100% to 68%, which can be read as "*74% of the participants claim that the equivalence between both knowledge representations vary from 100% to 68%* [5]". It is a great result since the textual representation is written without any formal structure; therefore, encompassing ambiguity and open interpretations. We argue the chosen text structure can achieve the expected results, which is to transmit the process knowledge through a natural language representation. Based on this, the answer for RQ1 is: "*The knowledge represented by the natural language text, generated by the framework, can be considered equivalent to the process model.*"

[5] *Each participant contributed with seven (7) answers, thus if we divide the total number of answers by seven (342/7) it give us the average number of participants that choose the same answer (48 participants)*

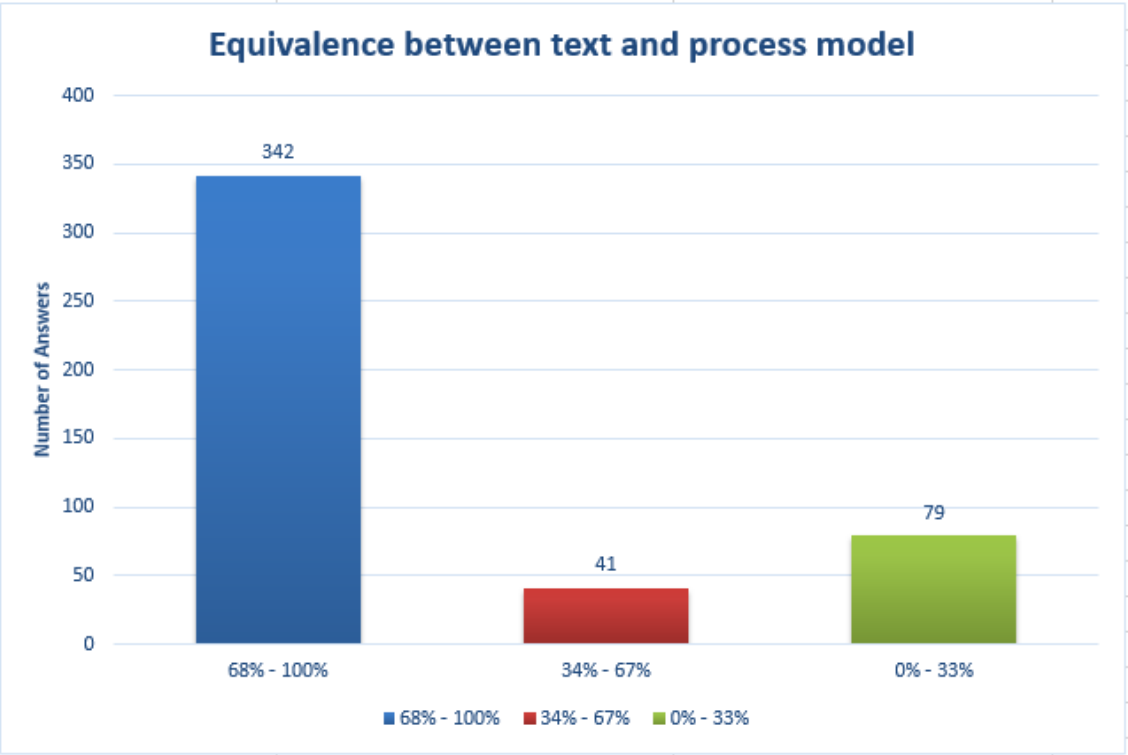


Figure 4.4: Subject's answers distribution among equivalence intervals.

For RQ2, we were interested in determining how many subjects claim the textual quality can be classified as "*Excellent*", "*Very Good*", "*Good*", "*Very Bad*" or "*Horrible*". Our expectation was the sum of answers varying from"*Excellent*", "*Very Good*" or "*Good*" were higher than the sum of all the others. Figure 4.5 depicts the overall evaluation without any grouping criteria, while Figure 4.6 depicts the overall evaluation for this question with the grouping criteria.

Analyzing the graphic depicted in Figure 4.6, 404 answers were within the first group (ranging from "*Good*", "*Very Good*" and "*Excellent*"). The result can be read as "*86% of the participants claim the textual description quality vary from Good, Very Good and Excellent* [6]". This can be considered a great result regarding our threshold, which was that 70% of the answers were within the first group.

Isolating the analysis for each process fragment This Section presents the evaluation regarding each process fragment. Figure 4.7 depicts the evaluation for RQ1, while Figure 4.8 depicts the evaluation for RQ2.

Figure 4.7 presents there is not much variation from one process fragment to another,

[6] *Each subject contributed with seven (7) answers, thus if we divide the total number of answers by seven (404/7), it give us the average number of subjects that choose the same answer (57 subjects).*

Figure 4.5: Participant's answers distribution among five groups (Ungrouped).

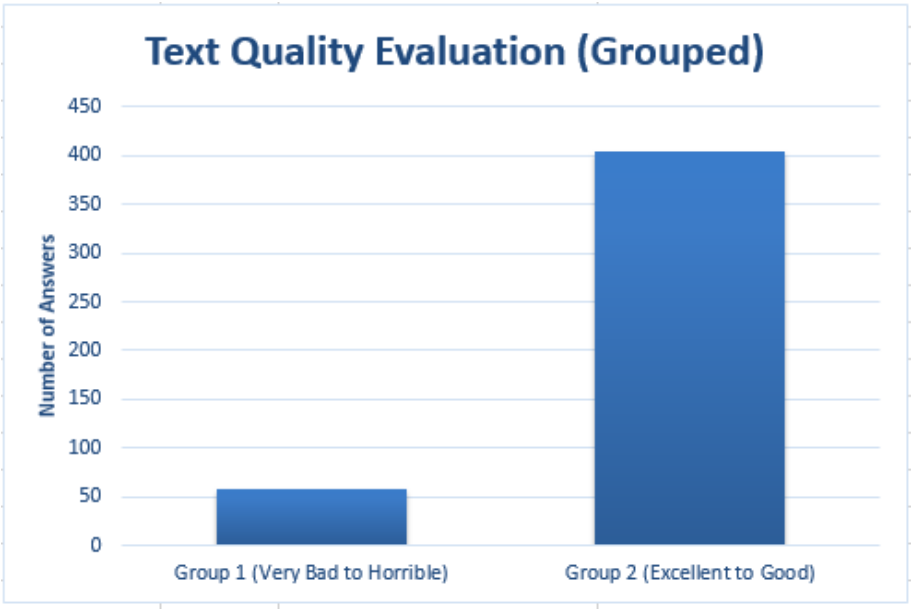


Figure 4.6: Participant's answers distribution among two groups (Grouped).

with the exception for the third process fragment. As mentioned earlier (Section 4.2.1), the expectation for the third process fragment was that the number of answers within the third and second group were higher that the number of answers within the first group. We got 19 answers for the first group, 3 answers for the second group and 44 answers for the third group, which give us: group 3 (44 answers) + group 2 (3 answers) > group 1 (19 answers) = 47 answers > 19 answers. Thus, the subjects were capable to identify the flaws in the third fragment with a good accuracy (68%). If we sum the answers percentage for group one, for each process fragment (excluding the third fragment) with

the percentage for group three for the third fragment and then divide the result by all the seven fragments, we should have the average for the knowledge representation between both representations. The final average is of 74%, which is 4% higher then our minimal threshold (70%). Therefore, textual work instructions are capable to represent the same knowledge that the model represents, within an acceptable difference due to the informal and lower abstraction given by the text format.

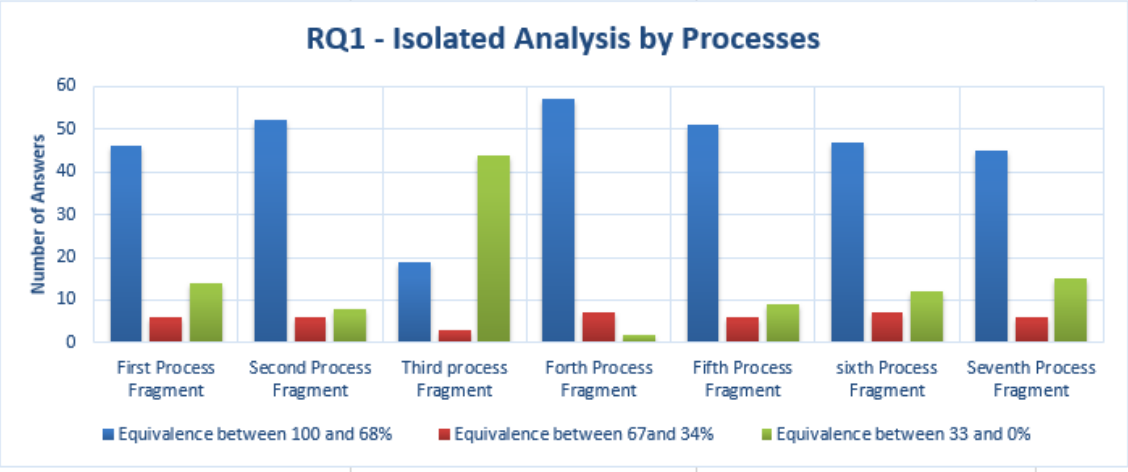


Figure 4.7: Chart that shows subject's answers distribution among the options available regarding the textual description quality.

Figure 4.8 presents there is not much variation from one process fragment to another, with the exception for the third process fragment also for RQ2. We got 41 answers within options four and five and 25 answers within options one, two and three. Thus, the subjects were capable to identify the flaws in the third fragment. Nevertheless, we expected that the difference among these two groups were higher because it means that 38% of the subjects claims that the textual quality vary from good to excellent. We argue this happen due to the abstraction undertaken by the subjects. It seems that most of the subjects were capable to abstract the model and analyze only the textual quality according to its structure. Unfortunately, we do not have enough data to assert this hypothesis, thus it can be answered by a future experiment. If we sum the answers percentage for group one (sum of answers for options 1 to 3) , for each process fragment (excluding the third fragment) with the percentage for group two for the third fragment and then divide the result by all the seven fragments, we should have the average quality for the textual representation. The result is an average of 86% of answers for group one, which can be read as: "*86% of the subjects claim the textual description quality vary from Good, Very Good and Excellent*". This indicates the chosen textual format is good. This result is aligned with what we expected due to the use of NLG techniques (*e.g.*, Discourse Marker insertion, Referring expression generation) which are capable of enhancing the text and improve its readability. We also

think that the use of bullets and indentation also contributed for the good evaluation.

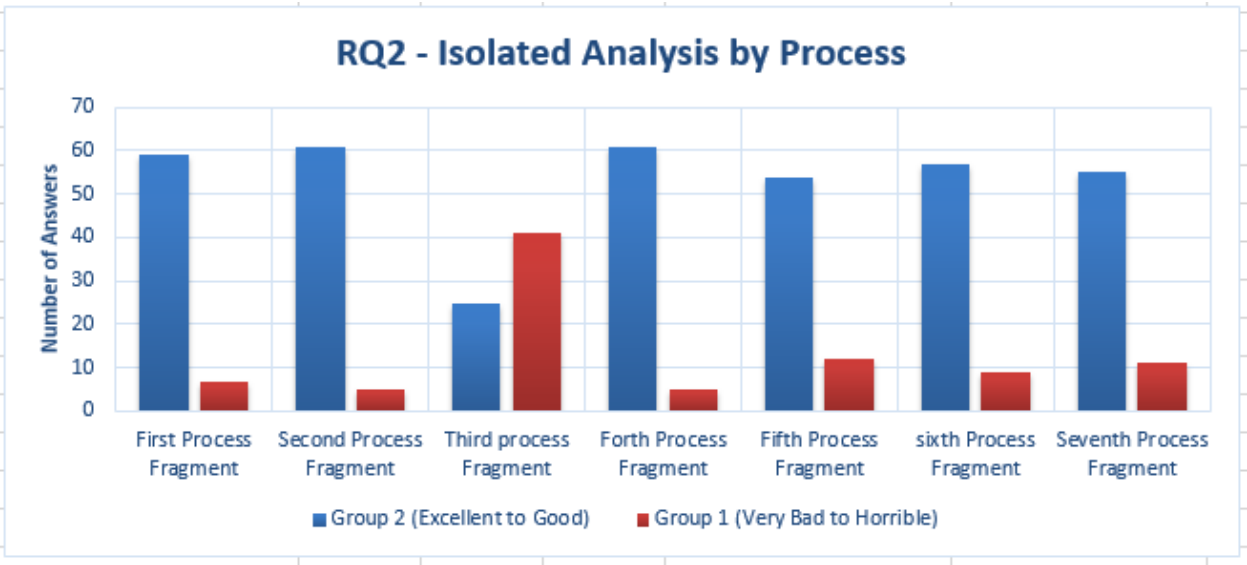


Figure 4.8: Chart that shows subject's answers distribution among the options available regarding the textual description quality.

Subjects Experience Evaluation We believed that users with limited experience in process modeling would better understand a process if it was described as textual work instructions instead of a BPMN process model. On the other hand, experienced users would better understand a process if it is depicted through a model instead of textual work instructions. Thus, experienced users would give a lower rate when asked about equivalence between both representations. We were also interested in investigating whether the subject's experience with process modeling could influence the overall evaluation described in Section 4.2.2.

To address this evaluation, subjects' experience must be analyzed and subjects must be classified into groups that represent different levels of experience with process modeling. To accomplish this, we prepared a dataset with one entry for each subject and five columns. The first column contained the subject identification number, a unique number for each subject. The four remaining columns contained the answers given by the related subject to the four questions comprising the characterization questionnaire (Section 4.2.1).

A clustering technique was applied upon the dataset to identify how many clusters (groups) can be drawn from the sample [123]. The hierarchical clustering technique produced a dendrogram that was used to visually identify the clusters and the subjects distribution among these clusters [121]. The R mathematical software was used to run the clustering technique. Figure 4.9 depicts two clusters identified after executing the tech-

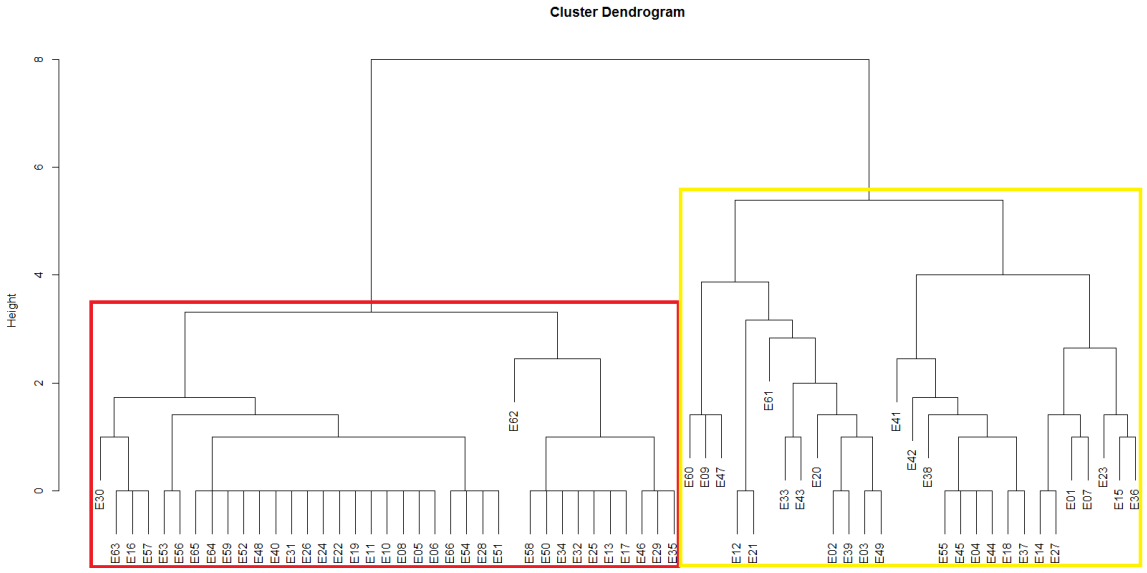


Figure 4.9: Two clusters identified by the hierarchical clustering technique.

nique upon the data we have collected. The first cluster (in the left side) represents experienced subjects regarding process modeling and BPMN. The second cluster (in the right side) represents inexperienced subjects. This technique can work with ordinal scale data, which was used in the answers to the characterization questionnaire.

After having visually identified two clusters, we can then draw two separate analysis. One for the experienced group and another for the inexperienced one, so that we are able to analyze if experience with process modeling and BPMN notation can affect subjects' answers significantly.

The first cluster represents experienced subjects and is composed by 37 subjects. The second cluster represents inexperienced subjects and is composed by 29 subjects. Thus, the number of experienced subjects suppress the number of inexperienced subjects by 8 which means that experienced subjects who participate in the experiment were 12% higher than inexperienced ones. Taking into account the seven process fragments, the experienced subjects contributed with 259 answers (37 subjects x 7 processes) while the inexperienced subjects contributed with 203 answers (29 subjects x 7 processes).

Figures 4.10 and 4.11 presents low variations between both clusters. For the experienced subjects we had 192 (74%) answers for group 3, 14 (6%) answers for group 2 and 53 answers for group 1 (20%). For the inexperienced subjects we had 140 (69%) answers for group 3, 36 (18%) answers for group 2 and 27 answers for group 1 (13%). The highest variation (14%) were within the second group. Nevertheless, the number of answers for this group (14 for experienced group and 36 for inexperienced group) cannot

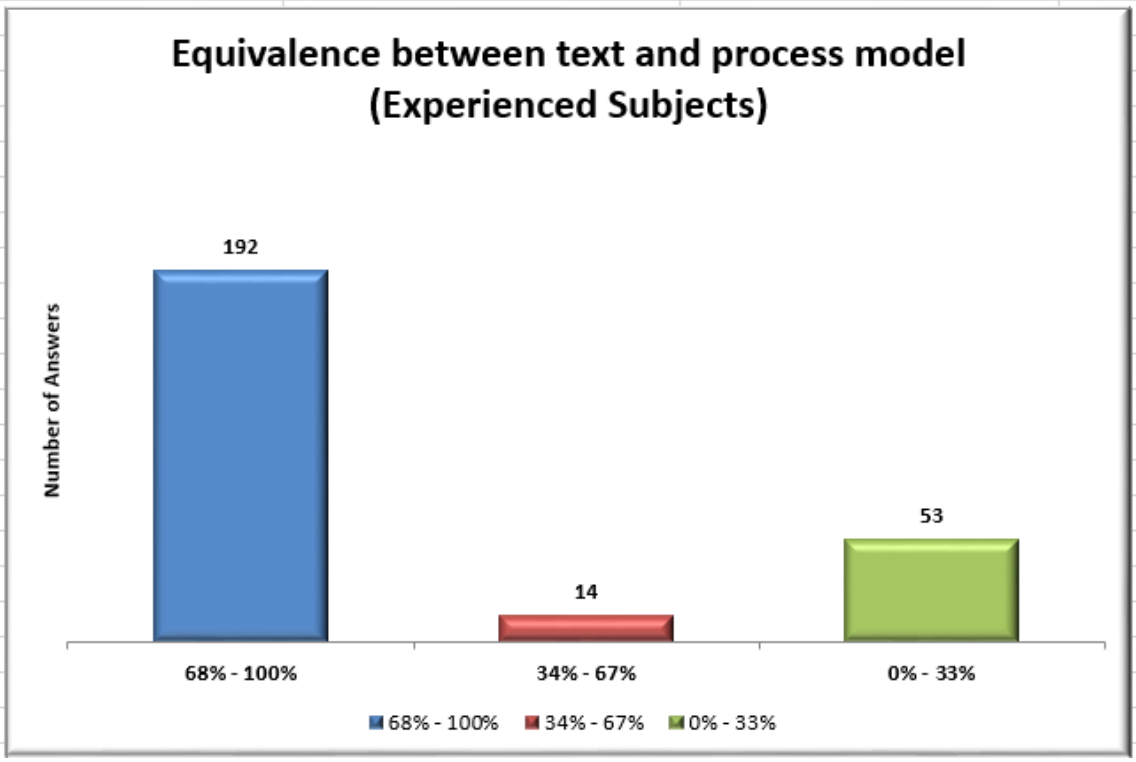


Figure 4.10: Chart that shows experienced subject's answers distribution among equivalence intervals.

be considered expressive enough to influence the overall analysis. We can then conclude that, regarding research question 1 (RQ1), the experience on process modeling does not influence the overall analysis.

Figures 4.12 and 4.13 presents low variations between both clusters. For the experienced subjects we had 220 (85%) answers for group 2 and 30 (15%) answers for group 1. For the inexperienced subjects we had 179 (88%) answers for group 2 and 24 (12%) answers for group 1. Thus, we had on average 3% variation between the answers for both clusters. This analysis indicates that, regarding research question 2 (RQ2), the experience on process modeling does not influence the overall analysis.

4.2.3Threats to Validity

One possible threat to the study was the strategy chosen to present the processes to the subjects. Due to time restrictions and to avoid an exhaustive questionnaire, we have presented only process fragments and not whole processes. This may lead to misunderstandings, because experienced subjects can detect lacks of important elements (*e.g.*, begin and end event). Nevertheless, we expected that this threat can be mitigated due to the subjects' abstraction ability, which help them focus on the analysis of the information

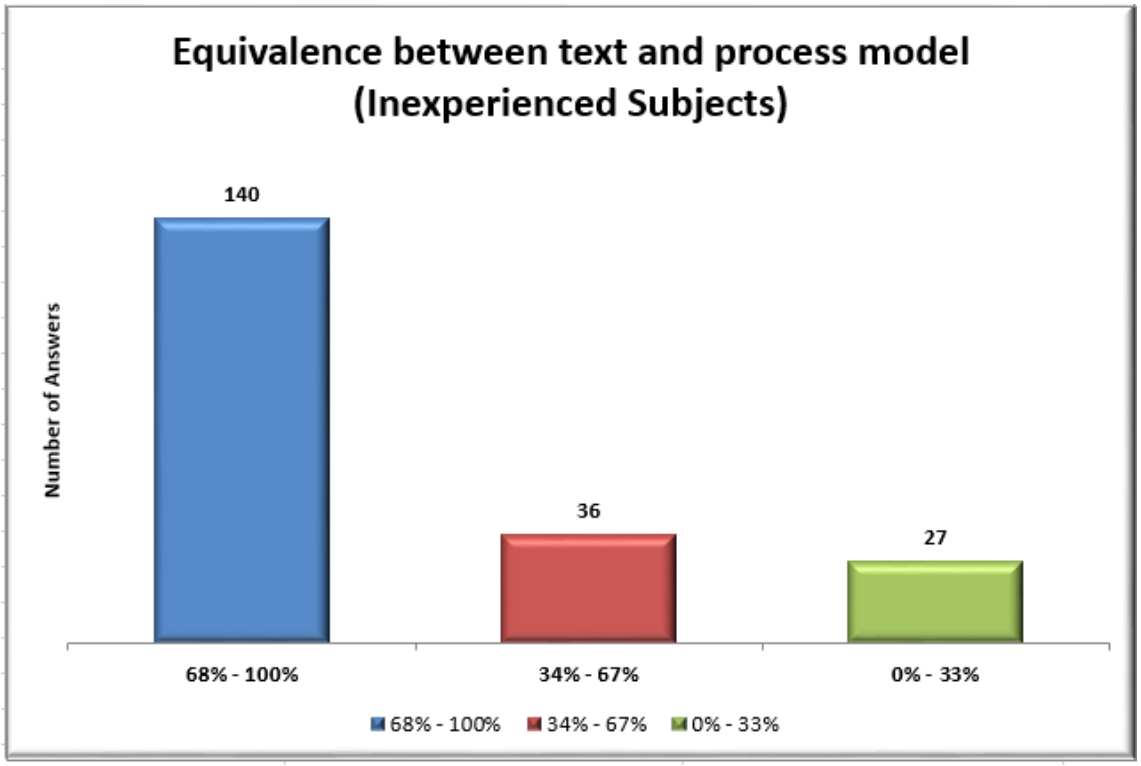


Figure 4.11: Chart that shows inexperienced subject's answers distribution among equivalence intervals.

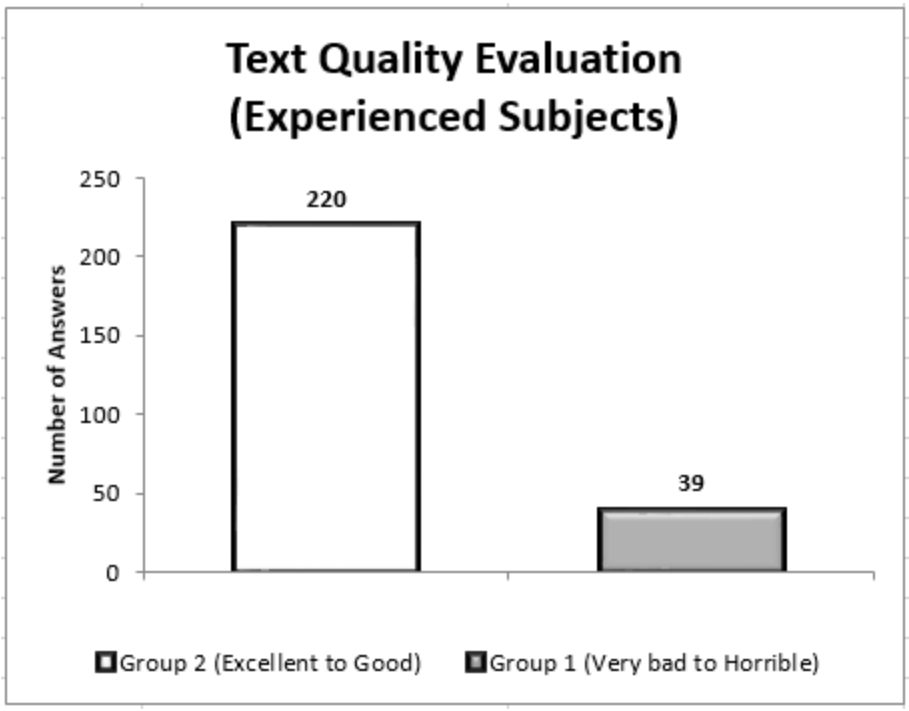


Figure 4.12: Chart that shows experienced subject's answers distribution among the options available regarding the textual description quality.

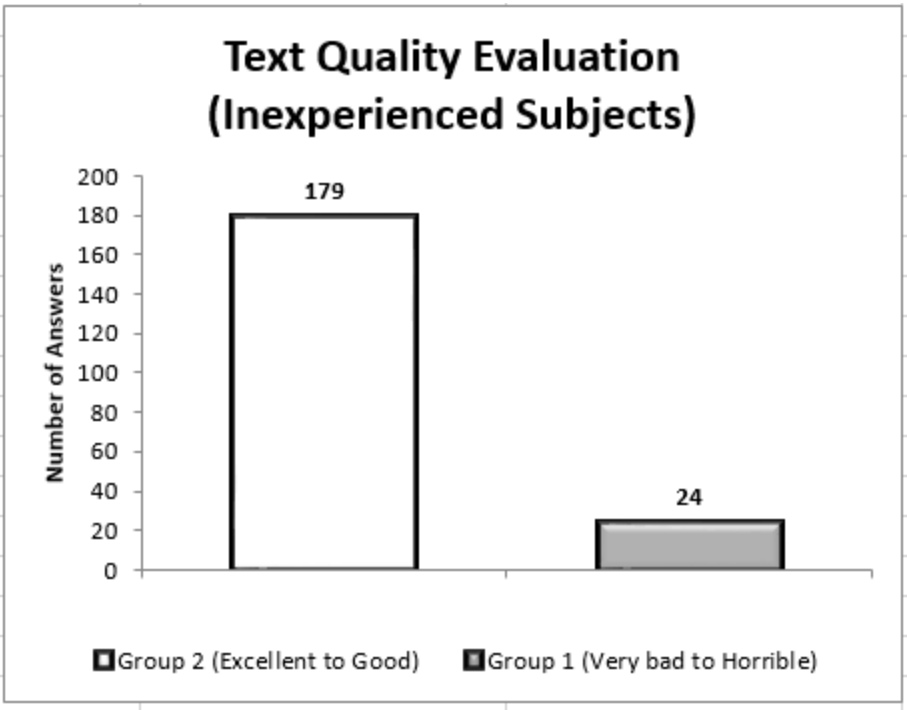


Figure 4.13: Chart that shows inexperienced subject's answers distribution among the options available regarding the textual description quality.

available instead of wondering about what is missing, and we have highlighted that the process was not a complete one but just a extracted fragment from a complete process.

Another possible threat is the presence of answers that does not follow a logic pattern. For instance, subjects that choose the same answers for all the process fragments, indicating that they were not paying attention to some details. Due to the number of subjects (67) we believe that this treat is mitigated, because this answers would not contribute significantly when considering the whole analysis.

The question that arises now is whether these results are generalizable. To what degree have we answered the proposed questions and closed the research gap? Unfortunately, we can hardly expect to find a small set of business process which can be representative of all the possibilities. There are also serious concerns about whether these results (already equivocal enough) would scale up to large or smaller processes. We cannot present new information to that question.

We are aware that besides understandability, there are other factors that are relevant when choosing a format for business process descriptions. For instance, the structure of both representations are taken into consideration through the premise that both must follow specific design guidelines, which makes easier for the user to understand the process [5] [80].

Regarding the process's expressiveness we believe that the selected process model set bears interesting characteristics for the proposed study. Additionally, the process cover many basic symbols proposed by the BPMN notation. Notwithstanding, these characteristics define a limited scope for which we have added a new piece of evidence by using a systematic study procedure.

4.2.4Conclusion

In general, graphic notations are easier for business users to understand and use [50]. The models can make explicit several process's patterns, flaws in process cycles and even bottlenecks or deadlocks. But, it is required that all the people involved with the process model have the necessary knowledge about the notation [62]. Besides, it is important to define a set of business process models samples that could be used as guides for the process model developers [62].

Two main conclusions can be drown from the study's analysis. First, the textual work instructions can be considered equivalent, in terms of knowledge representation, to process models within an acceptable threshold (74% of the subjects claims that the equivalence between both knowledge representations vary from 100% to 68%). Second, our evaluation indicates that the chosen textual format is good (86% of the subjects claims that the textual descriptions vary from excellent to good). This result is aligned with what we expected due to the use of NLG techniques like Discourse Marker insertion and Referring expression generation, which are capable of enhancing the text and improve its readability. Besides, we think that the use of bullets and indentation also contributed for the good evaluation.

Regarding the joint use of graphics and text to support understanding process models, both kinds of representation can fit together.

4.3Experiment: Text-Model Synchronization

This experiment evaluates the synchronization strategy (Section 3.4) implemented through the proposed framework. It was an exploratory research to investigate weather the automatically updated model is capable of transmitting the same knowledge as compared with the manually updated textual description. In other words, it aims to investigate if the text-based changes reflected to the process model were consistent.

4.3.1Experiment Design

This section presents the design of the proposed experiment, including the research questions, instruments selected to address these questions, subjects who participated in the experiment, and the measurements taken to: (a) Analyze the equivalence between the automatically updated process model and the manually updated textual description; and, (b) Analyze the process model synchronization strategy according to the subject's perspective.

Research Questions The main objective of this experiment is *Assess whether the knowledge represented by the automatically updated version of the process model, after been synchronized by text-based changes, represents the same knowledge as the manually updated textual description*.

The following research question was proposed to address this issue:

1. Is the knowledge represented by the manually updated text equivalent to the automatically updated process model?

Instrumentation An online questionnaire[7] was used to collect the data for this experiment. The questionnaire was composed by: (i) A set of questions to characterize subject experience in process modeling; and (ii) A set of fourteen (14) Text-Model pairs describing the operation (*i.e.*, change) made to the original text and the updated process model fragment after the framework's synchronization. Each pair was followed by two questions. The first question's objective was to rate the equivalence, varying from *Strongly Disagree* to *Strongly Agree*, between the text change and the model change based on it. The second question was optional and allowed subjects to enter a free text regarding their impressions about the synchronization strategy. The second question was designed to help us in answering the following question: "*How does the synchronization strategy, implemented through the framework, can be enhanced to achieve better results?* "

The strategy for designing the questionnaire was roughly the same as the one described in Section 4.2.1, including the same set of Characterization Questions (Table 4.3). Nevertheless, the set of Text-Model pairs were not the same and is detailed below:

Process Fragment: Text-Model Pair. The text-model pair aimed at rating the equivalence in terms of the information transmitted by both representations.

[7]Available at http://goo.gl/forms/JacmFbV3yTpPEfib2

First, a short description about the operation done to the text was presented to the participant. Afterwards, the text fragment (before and after applying the changes) alongside the model fragment (before and after having the textual change reflected to it) was presented. Figure 4.14 depicts one exemplary of the text-model pair used by this experiment. The process fragment was accompanied by two questions. The first question, presented in Table 4.6, was answered according to a 7-point ordinal scale. This question addressed whether the subject considered that the synchronization strategy, applied for updating the model through text-based changes, is consistent. The second question was optional and answered as a free text. This question aimed at gathering qualitative data to enable an open exploratory research (*i.e.*, feedback) about the synchronization strategy.

Change:

Type of Operation performed: Addition of Business Object to an Activity.
Addition of the business object "*price*" to the original sentence "*The client chooses the plan*"

Before	**After**
○ If the region is not covered, then he asks for signature subscription and chooses the plan.	○ If the region is covered, then he asks for signature subscription and chooses the plan and price.

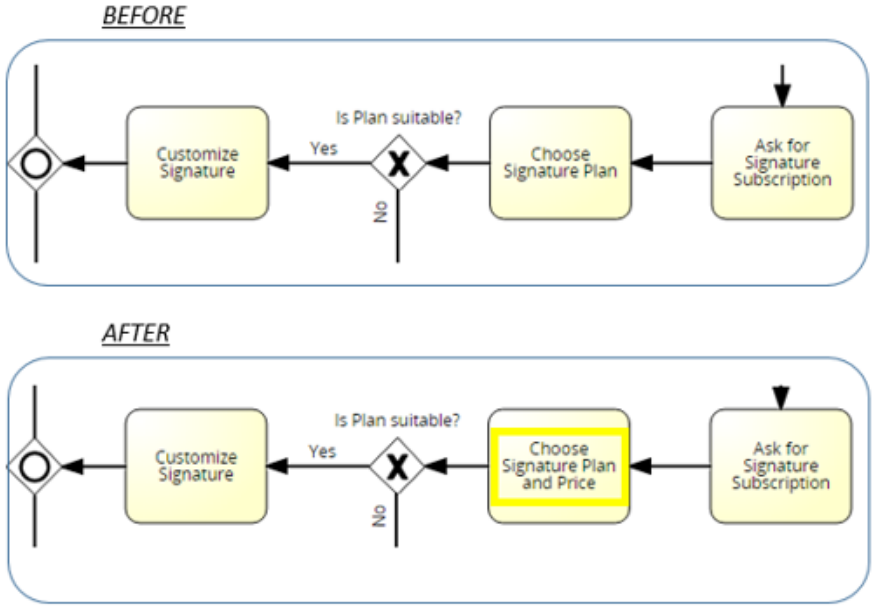


Figure 4 – Partial vision of the process model, before and after applying the change.

Figure 4.14: A text-model pair which was used during the experiment.

Subjects Several instances of the same instrument (described in the previous section) were given to the subjects. In total, 14 subjects were selected to participate. Subjects came

Table 4.6: Question about the process model synchronization after having changes made to the original text.

Question and Options
The updated process model after the synchronization is consistent with the text-based changes made, correctly reflecting the new process version.
a. Totally disagree
b. Disagree
c. Slightly disagree.
d. Neutral.
e. Slightly agree.
f. Agree
g. Totally agree.

from universities (*e.g.*, Federal University of the State of Rio de Janeiro (UNIRIO), State University of Rio de Janeiro (UERJ) and Federal University of Rio de Janeiro (UFRJ)) and from IT companies located in Rio de Janeiro (Brazil). As opposed to the experiment described in Section 4.2, this experiment had only subjects with modeling skills and experience with process modeling. This justifies the lower number of subjects, as compared with the first experiment evaluation.

Measurements Results were gathered through answers given by the subjects in the questionnaire: for each rating (varying from *Strongly Disagree* to *Strongly Agree*) we counted the total number of answers for the same rating. For example, we counted how many times option "*Strongly Agree*" was chosen, how many times option "*Agree*" was chosen and so on. The instrument presented fourteen (14) distinct process fragments, with the same question accompanied by the same number of options available to rate the synchronization between the process textual description and the process graphic model (BPMN). Thus, this give us a total of fourteen (14) answers per subject. As we have 14 subjects, this is equal to 196 (14 x 14) answers considering all process fragments.

Qualitative data was gathered through the answers given by the subjects in the questionnaire: for each optional question, we collected the feedback data and filtered out answers that could provide valuable insight to enhance the round-trip technique.

4.3.2Analysis and Discussion

This section presents the compiled result from the analysis of the data gathered. It address the research question defined in the experiment design (Section 4.3.1) and discuss some of the insights gathered from the analysis.

Overall Evaluation To address the proposed research question (Section 4.3.1), we were interested in determining how many answers were within the range varying from "*Strongly Disagree*" to "*Strongly Agree*". Our expectation was that the number of answers between "*Strongly Agree*" and "*Slightly Agree*" were higher than the sum of the other groups (subjects who rate accordance with the synchronization strategy between "*Neutral*" and "*Strongly Disagree*"). Thus, to enable a better reading of the results, we grouped the answers for options "*Strongly Agree*" and "*Slightly Agree*" into one group. All the remaining answers were grouped into a second group. Figure 4.15 depicts the overall evaluation for this question.

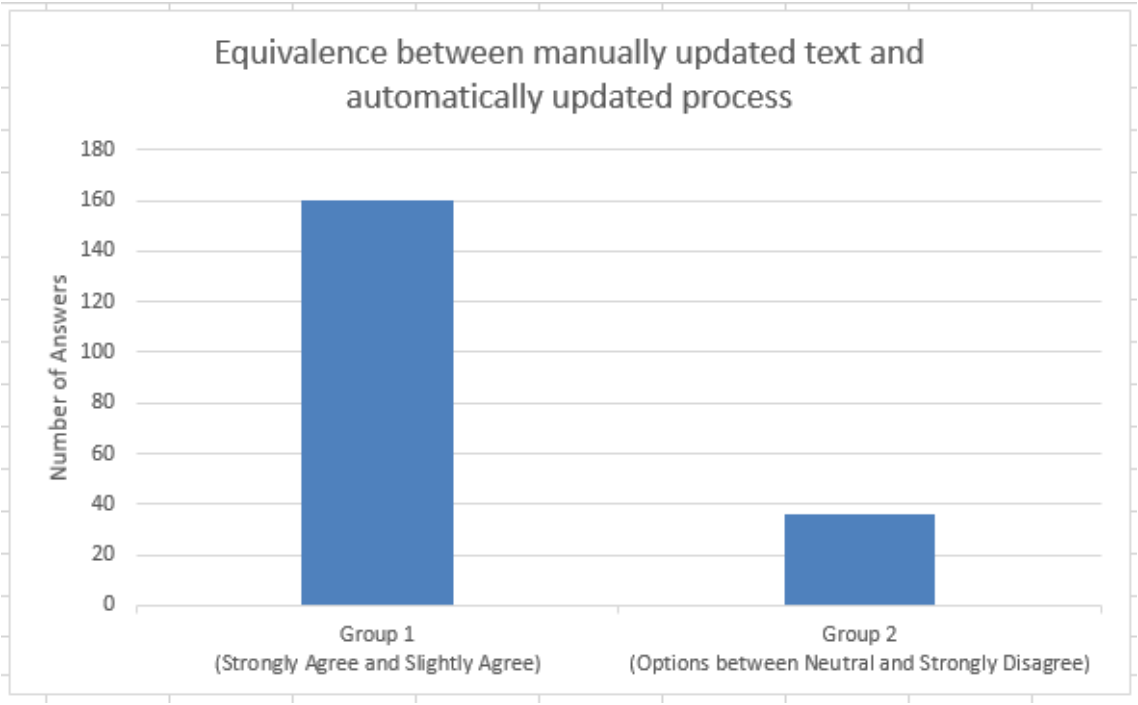


Figure 4.15: Subject's answers distribution among the available accordance options (grouped into two groups).

As can be observed, 160 answers were within the equivalence group ranging from "*Strongly Agree*" and "*Slightly Agree*", which can be read as "*78% of the subjects*[8] *claims*

[8]*Each subject contributed with seven (14) answers, thus if we divide the total number of answers by fourteen (160/14) it give us the average number of subjects that choose the same answer (11 subjects)*

the knowledge represented by the manually updated text is equivalent to the automatically updated process model". It is a great result since the textual representation is written without any formal structure; therefore, encompassing ambiguity and open interpretations.

Based on this, the answer for the proposed research question is: "*The knowledge represented by the manually updated text can be considered equivalent to the automatically updated process model.*"

Qualitative Evaluation As mentioned earlier, the second question was optional and answered as a free text. Thus, allowing us to perform a qualitative analysis using the feedback data provided by the subjects. The main goal of this qualitative evaluation was to investigate whether the implemented synchronization strategy could be enhanced to achieve better results.

From the whole set of possible optional answers (*i.e.*, 196 answers), we had a total of fifty (50) qualitative answers. From these fifty answers, we filtered out only answers that could provide feedback about the synchronization strategy, leading to the final value of fifteen (15) answers. The need of applying a filter is justified to narrow and prioritize the framework's enhancements that must be done. For example, there was feedback stating that the process could be more effective if the payment could be done by using money. This kind of feedback refers to how the process was designed and not to the synchronization strategy illustrated in the experiment, which is the focus of the evaluation. Feedback about process model elements disposition were considered as future work. There were also several feedback praising the proposed technique and claiming that it could aid during the modeling process. The most relevant feedback are grouped and discussed below:

- **Case of First word's letter does not follow a pattern:** there were several feedback about the usage of lower and upper case for the first word's letter in the process model activities. We think that the standardization of these scenarios can lead to better models and thus enhancing the synchronization strategy.
- **XOR Gateway "*yes*" and "*no*" labels are missing:** after submitting new tests, we confirmed that sometimes these labels were omitted from the updated process model. A new version was developed to address this issue and is already implemented within the current framework version.
- **Reflect gateway description change to events:** there were several feedback about updating an event as a consequence of a Gateway description update. For illustration purpose, consider the process fragment depicted by Figure 4.16. The XOR-Gateway

description is updated from "*Region Supported?*" to "*Region has Deliver?*" but the end event description continues referring to the old condition (Region not supported). The optimal result would be to update the event description to "*Region does not has deliver*" so that it matches the XOR-Gateway. Figure 4.17 illustrates this scenario.

- **Standardization of accentuation:** there were accentuation problems found within several words. This issue is under development and will be addressed in the next framework's version.

Figure 4.16: Process fragment which illustrates the scenario where an updated made to a Gateway description should also trigger an event description update.

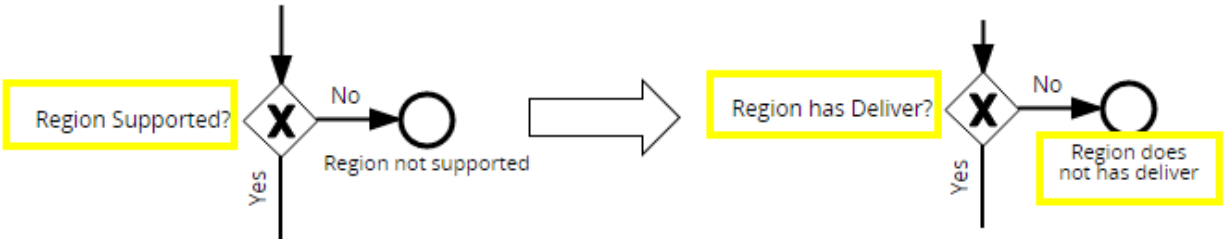


Figure 4.17: Process fragment which illustrates the optimal scenario where an updated made to a Gateway also update the event description.

4.3.3Threats to Validity

The same threats described for the experiment (Section 4.2.3) also applies for this experiment, due to the common characteristics shared by both.

The set of processes used was unique to each experiment and the second experiment took place approximately six months after the first one. Thus, the experience that a participant gained by participating in the first experiment is not likely to be representative enough to be considered as a threat to validity.

During this experiment, a process model was updated several times through changes made to the original text. Thus, it has been validated how the NLP pipeline (Section 3.3)

behaves in the scenarios described. This propose a threat to the round-trip validity, since the updated model is not submitted as input to assert weather the automatically generated text would match the original.

4.3.4Conclusion

Two main conclusions can be drown from the experiment's evaluation. First, the knowledge represented by the manually updated text can be considered equivalent to the automatically updated process model after the synchronization within an acceptable threshold (78% of the subjects claims that knowledge represented by the manually updated text is equivalent to the automatically updated process model). Second, our qualitative evaluation indicates several improvements that could help to achieve even better results. From these set of improvements, some were already implemented into the latest release while others were addressed as backlogs to be developed in future works. We have also received positive feedback regarding the framework's quality and usage potential. It is also important to highlight that no critical error, that could compromise the synchronization execution, has been found while running the experiment.

4.4Chapter Summary

This chapter presented details about the framework's evaluation, explaining the reasoning behind each evaluation. In total, three evaluations were made. The first evaluation was a PoC (Proof-of-Concept), which has as its main objective to validate the proposed solution in a in-house environment and also validate the framework language-independent features. Second, an experiment was conducted to assess whether the knowledge represented by the generated process description (*i.e.*, textual work instructions) can be considered equivalent to the process model. It also had as a secondary objective which was to evaluate the quality of the natural language text produced by the framework. Finally, a second experiment was conducted to assess whether the knowledge represented by the automatically updated version of the process model, after been synchronized by text-based changes, represents the same knowledge as the manually updated textual description. The following chapter describes works that are related to this master thesis scope.

5. Related Works

To the best of our knowledge, we are the first to propose a language-independent framework that implements a round-trip technique capable of generating natural language text from business process models and vice-versa. Due to its language independence, it can easily be adapted to a wide range of languages and it is extensible to support multiples text formats. Nevertheless, we were able to identify several related works that proposed similar approaches to the one described in this thesis. These approaches can be grouped into four (4) specific topics. These topics, and its related works, are better described in the subsequent sections, while also comparing their differences against the approach described by this thesis.

5.1Business Process Understandability

The field of process model understandability is discussed from different perspectives. For instance, the results from Mendling *et al.* show that the number of arcs has an important effect on the overall model understandability [78]. In fact, many studies on process model understandability have shown how complex the comprehension of process models can be, even for people who are familiar with process modeling [81]. Toward addressing this problem, Leopold *et al.* (2012) proposed a technique that can generate natural language text from BPMN process models, which can increase process understanding for non experienced users [62]. Mendling *et al.* demonstrates the impact of the natural language in the activity labels for model comprehension [79]. A more general perspective is taken by Zugal *et al.*, where the authors investigate in how far the cognitive inference process affects the model understanding [144]. The approach presented by Leopold *et al.* builds on these insights as it tries to lower the overall burden of process model comprehension [63].

There is a rich body of research on the pros and cons of visual diagrams in contrast

to natural language. The hypothesis by Larkin and Simon that, a diagram is sometimes worth ten thousand words, has inspired this stream of research [57]. Their observation is that text is limited to linear order while the spatial arrangement of different elements in a diagram allows for a more efficient information processing, through inducing cognitive processes such as visual chunking, mental imagery and parallel processing [139]. However, although the instructional and educational potential of graphical models are widely acknowledged, in some cases, they are not always more effective than other methods of representation. Usually, symbols of a graphical notations have to be learnt by readers in order to be understood [116]. This fact is exactly what sets system analysts and domain experts aside in terms of their model readership skills [62]. Therefore, training is required before the benefits of a graphical notation can materialize. This is supported by findings on considerable error rates in graphical process models [77, 42, 76]. The empirical findings on the strengths and weaknesses of text and diagrams are diverging. Moher *et al.* looked at several ways to express program structures in text and in Petri Nets, and state that "graphics were no better than text, and in several cases were considerably worse" [85]. The results obtained by Shneiderman *et al.* are also aligned with Moher *et al.* findings [115]. Shneiderman compared expressing program logic in flowchart and in programming language text. He also found that there were no statistically significant differences between the flowchart and non flowchart groups. By the contrary, in some cases the mean scores for the non flowchart groups even surpassed the means for the flowchart groups. Finally, also aligned with the aforementioned findings, Green *et al.* compares readability of textual and graphical programming notations, and refused the hypothesis, that graphics presentation would be superior. Actually, it was worse [41].

Prior work comparing process modeling notations can be roughly grouped into two categories: (i) Graphical notation comparison; (ii)Textual versus graphical notation comparison. The first is the most prominent one. Several of these studies suggest expertise is the most relevant factor in comprehension [19], but there is no absolute better or worse representation. However, while process understanding and comprehension has been intensively studied in recent research, there is a research gap on how the comprehension of business process can be affected when the information is presented in natural language text or a process model, according to the reader's experience with process modeling.

In previous work we tackled this gap through an experiment with several students and practitioners from the IT sector to address if there are significant differences in terms of process understandability depending on whether textual work instructions or process models are used to represent the process. Subject experience with process modeling is also taken into consideration in our analysis, allowing to identify the influence that the

experience have over the understanding of the process.

There are also other related works that focus on comparing business process understanding using different approaches for presenting the information. For example, compare declarative process models against a text based notation, using subjects that have some experience in modeling declarative process [43]. Differently to that work, our research used imperative process models, involved subjects whose experience with process modeling vary from none to expert, and presented a natural language text simulating a human description of the process [43].

While the experiment described by Ottensooser *et al.* compares model understanding with written use cases (using the Cockburn format), we have used a more fluent and natural representation of the text [89]. More specifically, we considered a natural language text which requires no background knowledge of layout or specific patterns. Also, our research focus on process understanding while Ottensooser *et al.* focus on domain understanding through different representations. Nevertheless, our findings corroborates to Ottensooser's results which indicate that there is no significant superiority between using graphical or textual notation for describing business process.

5.2Process Models to Textual Descriptions

It was also inspected related works where natural language techniques was used to achieve BPM relevant goals or areas where the creation of process models was the focus. The main challenge for generating text from process models is to adequately analyze the existing natural language fragments from the process model elements, and to organize the information from the process model in a sequential fashion. The below paragraphs present several approaches that provided initial insights for the construction of the NLG Core module architecture (Section 3.2).

The work presented by Leopold *et al.* describe an approach which automatically transforms BPMN process models into natural language texts combining different techniques from linguistics and graph decomposition [62]. It is based on the NLG pipeline defined by Reiter and Dale [27]. The evaluation of the technique is based on a prototypical implementation and involves a test set of 53 BPMN process models showing that natural language texts can be generated successfully. Following the same stream of research, Leopold *et al.* proposed a new approach which supports process model validation through natural language generation [63]. In this paper, the focus is using the generated texts for aiding the domain experts in the validation task, given the diverging skill set of domain experts

and system analysts. Nevertheless, although the base generation technique has been introduced, as opposed to our work both approaches do not support any other language besides English nor provide features for updating the original process model through text based changes made to the generated text. Another drawback is its architecture, which is not detailed, making it difficult to be used or extended to fit specific needs that were not defined beforehand.

5.3Textual Descriptions to Process Models

Goncalves *et al.* presents a method for deriving conceptual models from texts [24, 23]. The focus of this work was on the derivation of models from group stories, providing a prototype which handles Portuguese texts. Participants of the process to be analyzed are asked to write down their experiences. These texts are then interpreted using NLP techniques and BPMN process models are derived. The approach was further tested with a course enrollment process modeled by students. The examples of this paper show that process models can be created successfully, but only a limited set of BPMN elements is considered. Furthermore, a couple of their exhibits show that syntactical problems occur in some cases. As opposed to this thesis, it does not provide support for other languages rather than Portuguese nor is possible to customize the text pattern received as input. As a consequence, different text patterns are not supported by the method.

Ghose *et al.* developed a system called RBPD [40]. The toolkit uses a syntax parser to identify verb-object phrases in the given text and it also scans the text for textual patterns, like If <condition/event>, [then] <action>." [39]. The results are BPMN model snippets rather than a fully connected model. Nevertheless, this toolkit does not only derive BPMN snippets from unstructured text, but also takes existing model, *e.g.* an UML sequence diagram, into account. As some of the models used as a source might be a graphical representation of the texts which were also analyzed, a cross validation and check for duplicates is performed.

Kop and Mayr proposed a procedure called KCPM (Klagenfurt Conceptual Predesign Model) and developed a corresponding tool [52, 53]. It parses textual input in German and fills instances of a generic meta-model, the KCPM. Using the information stored in this meta-model an UML activity diagram and a UML class diagram can be created ([112] and [31], respectively) . The transformation from natural language input to the aforementioned meta-model is not a fully automated process, but rather semi-automated as a user has to be involved in the process. Using the tool, the user has to make decisions about the relevant

parts of a sentence or has to correct the automatic interpretations. Therefore, as opposed to our work, the proposed procedure does not leverage the full time- and cost-savings potential.

In contrast to that, the approach described by Tao *et al* is fully automated [143]. It uses use case descriptions in a format called RUCM [141] to generate UML activity and class diagrams [142]. But, the system is not able to parse free-text nor to generate process models. The RUCM input has to be in a very restricted format allowing only 26 types of sentence structures and relies on keywords like "*VALIDATES THAT*" or "*MEANWHILE*" to determine the semantics inherent in the text. Thus it cannot be used in an initial process definition phase as it would require rewriting of all documents present in a company to comply with the RUCM format. The main difference between Tao *et al* work and ours is that our approach is able to support free-text parsing through the interfaces implementations and is capable of generating process models from texts.

Wang *et al* describes a procedure which creates a BPMN diagram, given that data items, tasks, resources (actors), and constraints are identified in an input text document [135]. Although the approach does not require a process description to be sequential, as items are combined *e.g.* on their required inputs and outputs, it only supports a very limited set of BPMN elements. Pools, Data Objects, and Gateways other than an exclusive split are not considered. A exclusive Gateway is not allowed to have more than two outgoing arcs, reducing the space of available modeling constructs considerably. Furthermore, user-interaction is required at several stages throughout the process. Compared to Wang *et al* work, our approach does not require user-interaction during the parsing execution, it supports 15 different BPMN elements (including Pools and Gateways) and does not impose any restriction regarding the number of gateways' outgoing arcs.

Another approach, which is similar to ours was presented by Sinha *et al* (2008), Sinha *et al* (2010) and Kumanan *et al.* ([118, 117, 55], respectively). The authors of these papers employ a linguistic analysis engine based on the UIMA Framework. The UIMA Framework enables the constructions of linguistic analysis systems by combining different blocks into a pipeline. Specifically, the following steps are mentioned: Texts are preprocessed with a part-of-speech tagger in combination with a shallow parser. Afterward, the words are annotated with dictionary concepts, which classify verbs using a domain ontology created by the authors [118]. Then an anaphora resolution algorithms and a context annotator, which determines the likelihood of an identified noun phrase to be an actor in the system, is applied. The information is then transferred to a Use Case Description metamodel and later into a BPMN process model. The focus of this system is the analysis of very structured use-case descriptions. A single use case consisting of few sentences

is turned into a small process model. The authors then combine Subprocesses containing these models into a def-use graph, which is subsequently optimized [119]. Unfortunately, none of their works contains a full example text and model. Therefore, a verification of their results and a comparison to our approach is not possible. Furthermore, the approach can be regarded as text type specific as only use-case descriptions were used.

Leopold *et al.* also makes use of NLP techniques for integrating textual and model-based process descriptions for comprehensive process search [64]. The exiting techniques for automatically searching process repositories are limited to process models and many organizations complement these models with textual descriptions. The main idea behind this work is to enhance the process repository searches with the capability to search both, textual as well model-based process descriptions. To this end, they propose a unified data format that allows storing the information extracted from both process descriptions types in a unified way. The information is extracted from exclusives model and text-based parser developed by the authors. The format is implemented by building on the RDF[1] (Resource Description Framework) Based on such format, it becomes possible to perform search operations covering model-based and textual process descriptions. These operations are queries defined using SPARQL[2] (Simple Protocol And RDF Query Language). The evaluation of the technique is done with the process repository of an Austrian Bank. It shows that using the proposed technique, it was possible to retrieve additional relevant process models than using the available repository search techniques. The strategy of using intermediate data structures for storing process semantic data is also applied in this master thesis (Section 3). As opposed to Leopold's approach, our framework is capable of parsing process models and generating textual process descriptions from this models. It is also capable of parsing textual descriptions and reflecting changes made to it back to the process models. Besides, Leopold's approach is language specific, whereas our framework can deal with multiple languages. Despite the feature of searching process descriptions was not the focus of this thesis, we believe that our framework could be enhanced to support process model querying through the intermediate structures shared by both knowledge representation formats (*i.e.*, text and model-based descriptions).

Finally, the approach described by Friedrich *et al.*, proposes an automatic procedure for generating BPMN models from natural language text [34, 33]. The motivation behind this work is to address the problem of as-is models acquisition, which consumes up to 60% of the time spent on process management projects. The use of an automatic procedure for

[1]RDF is an XML-based specification developed by the World Wide Web Consortium (W3C). More information avaiable at http://www.w3.org/RDF

[2]In essence, SPARQL is similar to SQL (Structured Query Language), the most popular language to query data from relational databases), but is specifically designed to query RDF data.

generating business process models can take advantage of the extensive documentations often found within the companies, but are not in a ready-to-use format. The authors combine exiting tools from natural language processing and augment them with an anaphora resolution mechanism. The evaluation of the technique involves a test set of 47 text-model pairs from industry and textbooks, showing that they are able to generate on average 77% of the models correctly. The differences between their work and this thesis is that they generate a conceptual model (*i.e.*, process model) from natural language text and do not offer the possibility to navigate into the opposite direction. In other words, it is not possible to generate natural language texts from the models. It also has another drawback which is not begin language-independent and supporting only texts written in English. Furthermore, as far it is described, it is also not possible to extend or adapt the technique to fit specific needs. For example, to offer support to texts that follow specific patterns that were not defined beforehand.

While studying about these related works, it was reasoned about reusing some of the proposed approaches or even to combine them to help reach our objective. Nevertheless, we decided to focus on developing our own solution. The main reason for this was the lack of technical information (*e.g.*, API documentation, source code availability and architecture). Also, by developing our own code, we were capable of using our data structures for both: generate NLG texts from BPMN models and BPMN models from texts. By doing so, we manage to improve performance by updating only the model fragments which had changes. Finally, another reason that based this decision was our language core, which make our approach flexible enough to support multiple languages.

5.4Comparison

The previous sections described several related works that are within this dissertation research area. More specifically, works which are within the context of process models. Each paper has its own unique contribution along with several advantages and disadvantages. This section aims at identifying such characteristics followed by a comparison with this dissertation.

- **Leopold *et al.*, Supporting process model validation through natural language generation** [63]
  - **Objective:** Generate natural language texts from BPMN process models.
  - **Advantages:** Evaluations composed by several process models from both,

academy and industry.

  - **Disadvantages:** Is capable of parsing only BPMN models written in English, it cannot generate models from texts and the software architecture is void.

- **Goncalves *et al.*, A case study on designing business processes based on collaborative and mining approaches** [24].

  - **Objective:** Generate conceptual models from natural language texts.
  - **Advantages:** It is capable of parsing story like texts and can parse Portuguese texts.
  - **Advantages:** It does not offer support for any other language, it parses only story like texts and the evaluation is simple.

- **Ghose *et al.*, Process discovery from model and text artefacts, Rapid business process discovery** [39, 40].

  - **Objective:** Generate process models from natural language texts.
  - **Advantages:** It can generate UML compliant models.
  - **Disadvantages:** It parses only English texts, the models are partially generated (*i.e.*, requires manual competition) and the input text must be structured according to its guideline.

- **Tao *et al.*, Automatically deriving a UML analysis model from a use case model** [143].

  - **Objective:** Generate models from natural language texts.
  - **Advantages:** It is capable of generating UML sequence diagrams.
  - **Disadvantages:** It has a very strict output (only 26 output sentences may be used) and must follow the RUCM format.

- **Kop and Mayr, Tool supported extraction of behaviour models** [53].

  - **Objective:** Generate models from natural language texts.
  - **Advantages:** It is capable of generating both, UML sequence and activity diagrams. It defines an intermediate conceptual model, which can be easily used in other approaches.
  - **Disadvantages:** It supports only German written texts and is semi automatized (requires a modeler to finish the partial generated model).

- **Wang *et al.*, Policy-driven process mapping (PDPM): Discovering process models from business policies** [135].
  - **Objective:** Generates process model from natural language text.
  - **Advantages:** The process description does not need to be written in a sequential order.
  - **Disadvantages:** Only a small sub set of the core BPMN elements is supported and it is semi automatized (requires a modeler to finish the partial generated model).
- **Sinha *et al.*, Use cases to process specifications in business process modeling notation** [117].
  - **Objective:** Generate BPMN model from structured use cases.
  - **Advantages:** The paper describes the approach with details and the authors make use of several optimization techniques.
  - **Disadvantages:** The paper does not present any process model that was used by the authors.
- **Friedrich *et al.*, Process model generation from natural language text.** [34]
  - **Objective:** Generate model from natural language text.
  - **Advantages:** Development of an anaphora resolution component, which can be reused in other approaches. The evaluation count with more than 45 models from both, academy and industry, and is described in details.
  - **Disadvantages:** It is capable of parsing only English texts and does not support multiple text patterns that were not defined beforehand.

After identifying each paper advantage and disadvantage, we defined a set of criteria to enable a comparison between these papers and this thesis. The rationale behind each criteria is described below. Table 5.1 presents the comparison based on the defined criteria.

1. **Model Generation:** This criteria reflects whether the defined approach is capable of generating model from natural language texts.
2. **Text Generation:** This criteria reflects whether the defined approach is capable of generating text from process models. There are several companies which expend time and considerable resources in translating process models to process textual descriptions.

3. **Multiple Languages:** This criteria reflects whether the defined approach is capable of supporting multiple languages. Although English is the predominant business language, several companies typically model their processes in native language, partially driven by legal requirements [61]. Thus, language independence can be considered as an important feature to be found in language processing approaches.

4. **Architecture:** This criteria reflects whether the defined approach details its software architecture, making it possible to be easily extended or reused by future works. Source code availability is also considered within this criteria.

5. **Automatized:** This criteria reflects whether the defined approach is automatized or not. This can be considered as an important feature because it can significantly reduce modeling time and save employees from manually translating existent models to textual descriptions.

Table 5.1: Papers' Comparison

Paper	Model Generation	Text Generation	Multiple Languages	Architecture	Automatized
Leopold *et al.* [63]	-	YES	-	YES	YES
Goncalves *et al.* [24]	YES	YES	-	-	YES
Ghose *et al.* [39, 40]	-	-	-	-	-
Kop and Mayr [53]	YES	-	-	-	-
Tao *et al.* [143]	YES	-	-	-	YES
Wang *et al.* [135]	YES	-	-	-	-
Sinha *et al.* [117]	-	-	-	-	YES
Friedrich *et al.* [34]	YES	-	-	YES	YES
This thesis	YES	YES	YES	YES	YES

5.5Use of NLP techniques in other contexts

It was also inspected related work where NLP was used to achieve other BPM relevant goals or areas where the creation of process models was not the focus. The below paragraphs presents six streams of research that provided initial insights for the construction of the NLP pipeline architecture 3.3.

Research on the automatic matching of WebServices to user queries written in natural language [6, 18] Bosca *et al* and Cremene *el al* propose frameworks for the analysis

of user queries in natural language based on a domain ontology to determine an appropriate service composition. The services which were composed that way can then directly be parameterized with the information provided by the user to answer a query. The systems were used to answer question regarding entertainment, *e.g.* "*Which cinemas play the movie x*", or to control devices which support the UPnP protocol.

Identification of process model relevant sections in accompanying documentation [47] Ingvaldsen *et al* described a methodology which can be used to automatically determine links from an existing process model to the corresponding passages within a textual description [47]. Because both, the model and the text, are already assumed to be present, it becomes a matching problem, which is solved by applying a vector model to the text and the labels, respectively. As our approach uses the textual information to create the process model, such links can automatically be included. This enables the user to quickly gather additional information or verify the generated model.

Mining of process models from event logs [130] Process Mining is another approach for the automatic generation of process models [130]. But instead of text, it uses event logs (*e.g.*, from an ERP system) and then applies algorithms to construct a process model which is able to explain the logical and temporal relations between the events found in the log.

Generation of process models from text without linguistic analysis [70] A different approach to the generation of business processes in the EPC notation from use case templates is presented by Lubke [69]. A strict tabular form of use-cases where the action, trigger, pre and postconditions are clearly defined is required. By applying string matching to the pre and postconditions, a process model is created. The approach was tested with a single use-case describing a student enrolment process. Additionally, the approach was transferred to BPMN, but the requirement of a fully structured use-case template remain [70].

Usage of NLP techniques to provide machine-assistance during the modeling process [68] The Stanford Parser was used to analyze existing process models [68]. The results of this analysis is a Descriptor Space which contains information about entity life-cycles and activity hierarchies. This information can then be utilized to assist a modeler when creating a new model. Succeeding activities can be proposed based on the life-cycle of the used business objects in the analyzed model repository.

Works on the automated generation of data models, *e.g.* in UML, from text [67] The idea to automatically generate conceptual models from natural language input was pursued ten years ago already. Examples are the CM-Builder system, GOOAL and LIDA, which create UML class diagrams out of a software requirement text [44, 93, 90]. They analyze textual input using a POS tagger and create classes out of the identified noun phrases. Additionally, relations between the classes are created out of the S-V-O structure of a sentence. The mentioned approaches were compared and the limited set of supported UML modeling constructs was mentioned as the major limitation in a study conducted at the Heriot-Watt University, Edinburgh [67].

Although not exactly related to this research goal, we considered these six lines of research during the framework's construction as it provided valuable insights into the issues and considerations which are relevant for the automatic transformation of natural language input.

6. Conclusion

Many companies maintain both process models and textual work instructions to depict its business processes [129]. The use of both knowledge representations is needed to address specific audience. While domain experts are usually not qualified for reading process models, having to rely on textual descriptions, modeling experts prefer using the model representation. Hence companies face redundant effort for manually updating both process knowledge representation artifacts. This is prone to several inconsistencies problems, as usually only one of the artifacts is modified or updated.

In this thesis, we proposed a round-trip technique capable of automatically generating natural language text from process models and updating these models from editions on the text-based representation. By using this technique, it becomes possible to solve the mentioned inconsistency problem, leverage saving potentials, increase the efficiency of business analysts and to enable a quicker realization of BPM-projects and their benefits. The technique also enables domain experts to edit formal process models without the efforts of learning a modeling language.

The round-trip technique, combined existing tools from graph decomposition, natural language processing and generation in an innovative way. From a research perspective, the proposed technique provides the foundations for integrating textual and model-based information. The technique's capabilities were demonstrated through a prototype (*i.e.,* language-independent framework), implemented using the Java programming language. Throughout our analysis we highlighted the important issues for the construction of a framework capable of both, processing textual process descriptions and models. With this framework it is possible to maintain both business process representations (models and textual descriptions) automatically synchronized.

The technique was evaluated through three different perspectives. First, one proof of concept was design and executed by the authors with a small set of business process

models to evaluate the overall framework's behavior. Afterwards, some research questions were defined to serve as a guide during the execution of a case study, which aimed at validating and evaluating the natural language text produced as the framework output when given a process model instance as input. Finally, an experiment was run to evaluate the synchronization components through editions made to the original text and asking business process experts to evaluate whether the changes were reflected properly to the original process model. Several conclusions can be drown from the evaluation:

1. The textual work instructions can be considered equivalent to the process models in terms of knowledge representation within an acceptable threshold: *74% of the subjects claim the equivalence between both knowledge representations vary from 68% to 100%.*

2. Our evaluation indicates the chosen textual format is good: *86% of the subjects claim the textual descriptions vary from excellent to good*.

3. The knowledge represented by the manually updated text can be considered equivalent to the automatically updated process model after the synchronization within an acceptable threshold: *78% of the subjects claim the knowledge represented by the manually updated text is equivalent to the automatically updated process model*.

4. The NLG and NLP pipeline process (Figure 2.10 and 3.12, respectively) were followed without the detection of any error that could compromise the execution of the process.

5. The framework is capable of both: (i) Correctly generating natural language texts from business process models used as input; and, (ii) Generating process models from natural language texts.

6. The framework was able to map and treat all the BPMN elements defined in our subset (Figure 2.3), as well as their respective notations and labels used to their description. Therefore, we are confident that the required elements for the majority of BPM projects can be provided by our transformation procedure as it covers the most important and widely used elements [87].

From a practical perspective, our technique helps organization to simplify the process of model creation and simultaneously the effort required by a business analyst can be significantly reduced. To the best of our knowledge, we are the first to propose a language-independent framework that implements a round-trip technique capable of generating natural language text from business process models and vice-versa. Due to its

language independence, it can easily be adapted to a wide range of languages, belonging to the Romanian and Germanic sub-branches of the Indo-European language family, and is extensible to support multiple text formats. Hence, our evaluation results cannot be automatically transferred to other language families, as for instance Asian languages. However, our approach is designed as a language independent solution, which can be theoretically applied to any language.

The framework's architecture was designed to be extensible and to facilitate its usage in scenarios that were not defined beforehand. Thus, new researches could take advantage of the predefined modules and implement the interfaces to treat new scenarios or to help addressing other research questions.

6.1Limitations

Despite these encouraging results, our framework is able to read process descriptions consisting of full sentences. Furthermore, we assumed the description to be sequential and to contain no questions and little process irrelevant information. Another prerequisite is that the text is grammatically correct and constituent. Thus, the parsing of structured input, like tables or texts which are of low quality is not possible at the moment and presents opportunities for further research.

Word Sense Disambiguation and Named Entity Recognition was not within this thesis scope, since the main goal was to execute a round-trip using a structured text to generate process models [140]. By using structured text, we were able to define a text template. If the text does not matches the template, then it is rejected by the framework. This impose several limitations regarding the input text format and can be tackled by future researches.

Another issue is that our test data set comprising 30 text-model is relatively small. Therefore, our test results are not fully generalizable.

Regarding the process model notation coverage, our scope was the BPMN notation. Thus, no other notation is covered by our approach. Nevertheless, our architecture is flexible to allow the translation of a Process Model object to any chosen notation, by implementing specific interfaces. Another aspect to be taken into consideration is that the models must be compliant to the main process modeling and labeling style guidelines, otherwise the parsing component will not be able to handle the model correctly [80, 79, 61].

In a practical perspective, a problem that arises is the technique usage by companies

that have all processes documented only through textual descriptions and wish to automatically generate process models from these documents. Depending on the text pattern, these companies will have to implement its own parser, making use of the available interfaces or will have to extend the current template to support the desired structure, thus requiring to rewrite the text representation to fits the current framework implementation. Another drawback is the need to enhance the process model drawing algorithm. Currently, the framework is capable of updating a JSON process model but it is not possible to create the whole structure from zero. As mentioned in Section 2.2.1, a process model can be seen as a graph structure. Thus, drawing process models involves graph theory which was not within this thesis's scope.

On the other hand, companies that have processes documented through process models and wish to generate textual descriptions can make usage of the described technique without requiring any change. The descriptions can be automatically generated and the models will be updated according to text-based changes. Thus, the more direct focus is on companies who use process models as its primary process documentation format.

6.2Further Research

Different lines of research could be pursued in order to enhance the quality or scope of our round-trip technique. For example, it should be possible for the user to specify or override the text pattern for specific parts of the generated text without the need to develop new interface implementations. Due to the framework's architecture, this new feature could be implemented by making use of special labels (*i.e.*, markers) in the intermediate structures to specify which text format should be used for mapping specific process model elements.

A possible future work is to complement the framework with a domain expert. Hence, missing information could then be requested from the user in an interactive fashion and further relive the business analyst from interviewing tasks. Nevertheless, this would need the usage of semantic analysis and reasoning capabilities. The possibility to analyze and process larger amounts of textual information and extending the generation capabilities to include organization charts and data models, is also a important stream of future research in a practical context. Methods from the area of text mining and information retrieval would then be necessary to identify important fragments and to perform a thematic clustering of the acquired information [56].

We also suggest adding new languages to the tool. For this, it is necessary to imple-

ment the operations present in the specific interfaces for the treatment of a new language, defined in the package `GeneralLanguageCommon` (3.2). A language suitable for this test would be the German or Spanish.

Another improvement to be developed is the expansion of elements covered from the BPMN notation. Currently the subset we defined, presented in Figure 2.3, covers the core BPMN elements (15 elements). Even though zur Muehlen states that only few BPMN diagrams use more than 15 different elements, we believe that this set should be extended to support all BPMN symbols, thus improving the framework process model coverage [87].

Finally, the process model drawing algorithms need to be enhanced to support creating whole process models from zero. This could be achieved by applying graph theory techniques to draw the model elements (nodes) and the connecting objects (*e.g.*, sequence flow) [138].

Bibliography

[1] Camille Ben Achour. Guiding scenario authoring1. *Information Modelling and Knowledge Bases X*, 51:152, 1999.

[2] Susana Afonso, Eckhard Bick, Renato Haber, and Diana Santos. Floresta sintá (c) tica: A treebank for portuguese. In *LREC*, 2002.

[3] Thomas Baier and Jan Mendling. Bridging abstraction layers in process mining by automated matching of events and activities. In *Business Process Management*, pages 17–32. Springer, 2013.

[4] Wasana Bandara, Guy G Gable, and Michael Rosemann. Factors and measures of business process modelling: model building through a multiple case study. *European Journal of Information Systems*, 14(4):347–360, 2005.

[5] Jörg Becker, Michael Rosemann, and Christoph von Uthmann. Guidelines of business process modeling. In *Business Process Management*, pages 30–49. Springer, 2000.

[6] Alessio Bosca, Fulvio Corno, Giuseppe Valetto, and Roberta Maglione. On-the-fly construction of web services compositions from natural language requests. *Journal of Software*, 1(1):40–50, 2006.

[7] Ian Brace. *Questionnaire design*. Kogan Page London, 2004.

[8] Ronald Brachman and Hector Levesque. *Knowledge representation and reasoning*. Elsevier, 2004.

[9] Norman M Bradburn, Seymour Sudman, and Brian Wansink. *Asking questions: the definitive guide to questionnaire design–for market research, political polls, and social and health questionnaires*. John Wiley & Sons, 2004.

[10] Michael Busch. Using likert scales in l2 research a researcher comments…. *TESOL Quarterly*, 27(4):733–736, 1993.

[11] Stephan Busemann. *Best First Surface Realization*. DFKI, 1996.

[12] Stuart K Card, Jock D Mackinlay, and Ben Shneiderman. *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[13] Evellin Cristine Souza Cardoso, João Paulo A Almeida, and Giancarlo Guizzardi. Requirements engineering based on business process models: A case study. In *Enterprise Distributed Object Computing Conference Workshops, 2009. EDOCW 2009. 13th*, pages 320–327. IEEE, 2009.

[14] Russell Carney and Joel Levin. Pictorial illustrations still improve students' learning from text. *Educational psychology review*, 14(1):5–26, 2002.

[15] Lucia Castro, Fernanda Baião, and Giancarlo Guizzardi. A semantic oriented method for conceptual data modeling in ontouml based on linguistic concepts. In *Conceptual Modeling–ER 2011*, pages 486–494. Springer, 2011.

[16] Suranjan Chakraborty, Saonee Sarker, and Suprateek Sarker. An exploration into the process of requirements elicitation: A grounded approach. *Journal of the Association for Information Systems*, 11(4):1, 2010.

[17] Michele Chinosi and Alberto Trombetta. Bpmn: An introduction to the standard. *Computer Standards & Interfaces*, 34(1):124–134, 2012.

[18] Marcel Cremene, Jean-Yves Tigli, Stéphane Lavirotte, Florin-Claudiu Pop, Michel Riveill, and Gaëtan Rey. Service composition based on natural language requests. In *Services Computing, 2009. SCC'09. IEEE International Conference on*, pages 486–489. IEEE, 2009.

[19] Bill Curtis, Sylvia B Sheppard, Elizabeth Kruesi-Bailey, John Bailey, and Deborah A Boehm-Davis. Experimental evaluation of software documentation formats. *Journal of Systems and Software*, 9(2):167–207, 1989.

[20] Robert Dale and Ehud Reiter. *Building natural language generation systems*. Cambridge University Press, 2000.

[21] Hercules Dalianis. Aggregation in natural language generation. *Computational Intelligence*, 15(4):384–414, 1999.

[22] Daniela Damian and James Chisan. An empirical study of the complex relationships between requirements engineering processes and other processes that lead to payoffs in productivity, quality, and risk management. *Software Engineering, IEEE Transactions on*, 32(7):433–453, 2006.

[23] Goncalves de AR, Joao Carlos, Flávia Maria Santoro, and Fernanda Araujo Baião. A case study on designing business processes based on collaborative and mining approaches. In *Computer Supported Cooperative Work in Design (CSCWD), 2010 14th International Conference on*, pages 611–616. IEEE, 2010.

[24] Joao Carlos de AR Goncalves, Flavia Maria Santoro, and Fernanda Araujo Baiao. Business process mining from group stories. In *Computer Supported Cooperative Work in Design, 2009. CSCWD 2009. 13th International Conference on*, pages 161–166. IEEE, 2009.

[25] Paul F Dietz. Maintaining order in a linked list. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 122–127. ACM, 1982.

[26] Robert MW Dixon. Deriving verbs in english. *Language Sciences*, 30(1):31–52, 2008.

[27] Robert Dale Ehud Reiter. Building applied natural language generation systems. *Natural Language Engineering 1*, 1997.

[28] Shimon Even. *Graph algorithms*. Cambridge University Press, 2011.

[29] Kathrin Figl and Ralf Laue. Cognitive complexity in business process modeling. In *Advanced Information Systems Engineering*, pages 452–466. Springer, 2011.

[30] Günther Fliedl, Christian Kop, and Heinrich C Mayr. From textual scenarios to a conceptual schema. *Data & Knowledge Engineering*, 55(1):20–37, 2005.

[31] Günther Fliedl, Christian Kop, Heinrich C Mayr, Alexander Salbrechter, Jürgen Vöhringer, Georg Weber, and Christian Winkler. Deriving static and dynamic concepts from software requirements using sophisticated tagging. *Data & Knowledge Engineering*, 61(3):433–448, 2007.

[32] Paul JM Frederiks and Th P Van der Weide. Information modeling: The process and the required competencies of its participants. *Data & Knowledge Engineering*, 58(1):4–20, 2006.

[33] Fabian Friedrich. *Automated generation of business process models from natural language input*. PhD thesis, Citeseer, 2010.

[34] Fabian Friedrich, Jan Mendling, and Frank Puhlmann. Process model generation from natural language text. In *Advanced Information Systems Engineering*, pages 482–496. Springer, 2011.

[35] Michel Galley, Eric Fosler-Lussier, and Alexandros Potamianos. Hybrid natural language generation for spoken dialogue systems. In *INTERSPEECH*, pages 1735–1738, 2001.

[36] Albert Gatt, Francois Portet, Ehud Reiter, Jim Hunter, Saad Mahamood, Wendy Moncur, and Somayajulu Sripada. From data to text in the neonatal intensive care unit: Using nlg technology for decision support and information management. *Ai Communications*, 22(3):153–186, 2009.

[37] Niyu Ge, John Hale, and Eugene Charniak. A statistical approach to anaphora resolution. In *Proceedings of the sixth workshop on very large corpora*, volume 71, page 76, 1998.

[38] Andrew Gemino. Empirical comparisons of animation and narration in requirements validation. *Requirements Engineering*, 9(3):153–168, 2004.

[39] Aditya Ghose, George Koliadis, and Arthur Chueng. Process discovery from model and text artefacts. In *Services, 2007 IEEE Congress on*, pages 167–174. IEEE, 2007.

[40] Aditya Ghose, George Koliadis, and Arthur Chueng. Rapid business process discovery (r-bpd). In *Conceptual Modeling-ER 2007*, pages 391–406. Springer, 2007.

[41] Thomas RG Green, Marian Petre, and RKE Bellamy. Comprehensibility of visual and textual programs: A test of superlativism against the'match-mismatch'conjecture. *ESP*, 91(743):121–146, 1991.

[42] Volker Gruhn and Ralf Laue. What business process modelers can learn from programmers. *Science of Computer Programming*, 65(1):4–13, 2007.

[43] Cornelia Haisjackl and Stefan Zugal. Investigating differences between graphical and textual declarative process models. In *Advanced Information Systems Engineering Workshops*, pages 194–206. Springer, 2014.

[44] Harmain M Harmain and R Gaizauskas. Cm-builder: An automated nl-based case tool. In *Automated Software Engineering, 2000. Proceedings ASE 2000. The Fifteenth IEEE International Conference on*, pages 45–53. IEEE, 2000.

[45] AR Henver, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[46] Jerry R Hobbs. Resolving pronoun references. *Lingua*, 44(4):311–338, 1978.

[47] Jon Espen Ingvaldsen, Jon Atle Gulla, Xiaomeng Su, and Harald Rønneberg. A text mining approach to integrating business process models and governing documents. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 473–484. Springer, 2005.

[48] Dan Jurafsky and James H Martin. *Speech and language processing*. Pearson, 2014.

[49] Rodger Kibble and Richard Power. An integrated framework for text planning and pronominalisation. In *Proceedings of the first international conference on Natural language generation-Volume 14*, pages 77–84. Association for Computational Linguistics, 2000.

[50] Ryan KL Ko, Stephen SG Lee, and Eng Wah Lee. Business process management (bpm) standards: a survey. *Business Process Management Journal*, 15(5):744–791, 2009.

[51] Jens Kolb, Henrik Leopold, Jan Mendling, and Manfred Reichert. Creating and updating personalized and verbalized business process descriptions. In *The Practice of Enterprise Modeling*, pages 191–205. Springer, 2013.

[52] Christian Kop and Heinrich C Mayr. Conceptual predesign bridging the gap between requirements and conceptual design. In *Requirements Engineering, 1998. Proceedings. 1998 Third International Conference on*, pages 90–98. IEEE, 1998.

[53] Christian Kop, Jürgen Vöhringer, Martin Hölbling, Thomas Horn, Heinrich C Mayr, and Christian Irrasch. Tool supported extraction of behavior models. In *ISTA*, volume 63, pages 114–123, 2005.

[54] Joseph B Kruskal. overview of sequence comparison. *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison/edited by David Sankoff and Joseph B. Krustal*, 1983.

[55] Palani Kumanan, Amit Paradkar, Avik Sinha, and Stanley M Sutton Jr. Automated inspection of industrial use case models inferred from textual descriptions. 2010.

[56] Sylvain Lamprier, Tassadit Amghar, Bernard Levrat, and Frederic Saubion. Using an evolving thematic clustering in a text segmentation process. *J. UCS*, 14(2):178–192, 2008.

[57] Jill H Larkin and Herbert A Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive science*, 11(1):65–100, 1987.

[58] Craig Larman. Applying uml and patterns: An introduction to object-oriented analysis and design and iterative development. 2005.

[59] Benoit Lavoie and Owen Rambow. A fast and portable realizer for text generation systems. In *Proceedings of the fifth conference on Applied natural language processing*, pages 265–268. Association for Computational Linguistics, 1997.

[60] Benoit Lavoie, Owen Rambow, and Ehud Reiter. The modelexplainer. In *Demonstration presented at the Eighth International Workshop on Natural Language Generation, Herstmonceux, Sussex*, 1996.

[61] Henrik Leopold, Rami-Habib Eid-Sabbagh, Jan Mendling, Leonardo Guerreiro Azevedo, and Fernanda Araujo Baião. Detection of naming convention violations in process models for different languages. *Decision Support Systems*, 56:310–325, 2013.

[62] Henrik Leopold, Jan Mendling, and Artem Polyvyanyy. Generating natural language texts from business process models. In *Advanced Information Systems Engineering*, pages 64–79. Springer, 2012.

[63] Henrik Leopold, Jan Mendling, and Artem Polyvyanyy. Supporting process model validation through natural language generation. 2014.

[64] Henrik Leopold, Han van der Aa, Fabian Pittke, Manuel Raffel, Jan Mendling, and Hajo A Reijers. Integrating textual and model-based process descriptions for comprehensive process search. In *International Workshop on Business Process Modeling, Development and Support*, pages 51–65. Springer, 2016.

[65] David D Lewis and Karen Spärck Jones. Natural language processing for information retrieval. *Communications of the ACM*, 39(1):92–101, 1996.

[66] Jiexun Li, Harry Jiannan Wang, Zhu Zhang, and J Leon Zhao. A policy-based process mining framework: mining business policy texts for discovering process models. *Information Systems and E-Business Management*, 8(2):169–188, 2010.

[67] Ke Li, RG Dewar, and RJ Pooley. Object-oriented analysis using natural language processing. *Linguistic Analysis*, pages 75–76, 2005.

[68] Maya Lincoln, Mati Golani, and Avigdor Gal. Machine-assisted design of business process models using descriptor space analysis. In *International Conference on Business Process Management*, pages 128–144. Springer, 2010.

[69] Daniel Lübke. Transformation of use cases to epc models. In *EPK*, pages 137–156. Citeseer, 2006.

[70] Daniel Lübke and Kurt Schneider. Visualizing use case sets as bpmn processes. In *Requirements Engineering Visualization, 2008. REV'08.*, pages 21–25. IEEE, 2008.

[71] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[72] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[73] James H Martin and D Jurafsky. Speech and language processing, 2000.

[74] Igor A Mel'čuk and Alain Polguere. A formal lexicon in the meaning-text theory:(or how to do lexica with words). *Computational linguistics*, 13(3-4):261–275, 1987.

[75] Jan Mendling. *Metrics for process models: empirical foundations of verification, error prediction, and guidelines for correctness*, volume 6. Springer Science & Business Media, 2008.

[76] Jan Mendling. Empirical studies in process model verification. In *Transactions on petri nets and other models of concurrency II*, pages 208–224. Springer, 2009.

[77] Jan Mendling, Michael Moser, Gustaf Neumann, HMW Verbeek, Boudewijn F van Dongen, and Wil MP van der Aalst. *Faulty EPCs in the SAP reference model*. Springer, 2006.

[78] Jan Mendling, Hajo A Reijers, and Jorge Cardoso. What makes process models understandable? In *Business Process Management*, pages 48–63. Springer, 2007.

[79] Jan Mendling, Hajo A Reijers, and Jan Recker. Activity labeling in process modeling: Empirical insights and recommendations. *Information Systems*, 35(4):467–482, 2010.

[80] Jan Mendling, Hajo A Reijers, and Wil MP van der Aalst. Seven process modeling guidelines (7pmg). *Information and Software Technology*, 52(2):127–136, 2010.

[81] Jan Mendling, Mark Strembeck, and Jan Recker. Factors of process model comprehension findings from a series of experiments. *Decision Support Systems*, 53(1):195–206, 2012.

[82] Jan Mendling, Boudewijn F van Dongen, and Wil MP van der Aalst. Getting rid of the or-join in business process models. In *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*, pages 3–3. IEEE, 2007.

[83] Marie Meteer. *Expressibility and the problem of efficient text planning*. Bloomsbury Publishing, 2015.

[84] Farid Meziane, Nikos Athanasakis, and Sophia Ananiadou. Generating natural language specifications from uml class diagrams. *Requirements Engineering*, 13(1):1–18, 2008.

[85] Thomas G Moher, DC Mak, B Blumenthal, and LM Levanthal. Comparing the comprehensibility of textual and graphical programs. In *Empirical Studies of Programmers: Fifth Workshop*, pages 137–161. Ablex, Norwood, NJ, 1993.

[86] Gerardus Maria Nijssen and Terence Aidan Halpin. *Conceptual Schema and Relational Database Design: a fact oriented approach*. Prentice-Hall, Inc., 1989.

[87] Anna Gunhild Nysetvold and John Krogstie. Assessing business process modeling languages using a generic quality framework. *Advanced topics in database research*, 5:79–93, 2006.

[88] OMG. Business process model and notation (bpmn) version 2.0. http://www.bpmn.org/, 2011.

[89] Avner Ottensooser, Alan Fekete, Hajo A Reijers, Jan Mendling, and Con Menictas. Making sense of business process descriptions: An experimental comparison of graphical and textual notations. *Journal of Systems and Software*, 85(3):596–606, 2012.

[90] Scott P Overmyer, Benoit Lavoie, and Owen Rambow. Conceptual modeling through linguistic analysis using lida. In *Proceedings of the 23rd international conference on Software engineering*, pages 401–410. IEEE Computer Society, 2001.

[91] Fred Paas, Alexander Renkl, and John Sweller. Cognitive load theory and instructional design: Recent developments. *Educational psychologist*, 38(1):1–4, 2003.

[92] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135, 2008.

[93] Hector G Perez-Gonzalez and Jugal K Kalita. Gooal: a graphic object oriented analysis laboratory. In *Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 38–39. ACM, 2002.

[94] Massimo Poesio and Mijail Alexandrov Kabadjov. A general-purpose, off-the-shelf anaphora resolution module: Implementation and preliminary evaluation. In *LREC*, 2004.

[95] Artem Polyvyanyy, Jussi Vanhatalo, and Hagen Völzer. Simplified computation and generalization of the refined process structure tree. In *Web Services and Formal Methods*, pages 25–41. Springer, 2011.

[96] Wolfgang Pree. Meta patterns a means for capturing the essentials of reusable object-oriented design. In *Object-oriented programming*, pages 150–162. Springer, 1994.

[97] Anand Rajaraman, Jeffrey D Ullman, Jeffrey David Ullman, and Jeffrey David Ullman. *Mining of massive datasets*, volume 77. Cambridge University Press Cambridge, 2012.

[98] Leonardo Azevedo Raphael Rodrigues and Kate Revoredo. Sincronização automática dos artefatos de processos de negócio: Métodos e aplicações. In *SBSI 2015 - WTDSI 2015*, may 2015.

[99] Janice Rattray and Martyn C Jones. Essential elements of questionnaire design and development. *Journal of clinical nursing*, 16(2):234–243, 2007.

[100] Raj Ratwani, Gregory Trafton, and Deborah Boehm-Davis. Thinking graphically: Connecting vision and cognition during graph comprehension. *Journal of Experimental Psychology: Applied*, 14(1):36, 2008.

[101] Jan Recker. *Scientific research in information systems: a beginner's guide*. Springer, 2012.

[102] Hajo A Reijers, Selma Limam, and Wil MP Van Der Aalst. Product-based workflow design. *Journal of Management Information Systems*, 20(1):229–262, 2003.

[103] Hajo A Reijers and Jan Mendling. A study into the factors that influence the understandability of business process models. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 41(3):449–462, 2011.

[104] Ehud Reiter. Nlg vs. templates. *arXiv preprint cmp-lg/9504013*, 1995.

[105] Ehud Reiter. An architecture for data-to-text systems. In *Proceedings of the Eleventh European Workshop on Natural Language Generation*, pages 97–104. Association for Computational Linguistics, 2007.

[106] Ehud Reiter and Chris Mellish. Optimizing the costs and benefits of natural language generation. In *IJCAI*, pages 1164–1171, 1993.

[107] Ehud Reiter, Chris Mellish, and John Levine. Automatic generation of on-line documentation in the idas project. In *Proceedings of the third conference on Applied natural language processing*, pages 64–71. Association for Computational Linguistics, 1992.

[108] R. D. A. Rodrigues, M. D. O. Barros, K. Revoredo, L. G. Azevedo, and H. Leopold. An experiment on process model understandability using textual work instructions and bpmn models. In *Software Engineering (SBES), 2015 29th Brazilian Symposium on*, pages 41–50, Sept 2015.

[109] Raphael Rodrigues, Leonardo Azevedo, Kate Revoredo, and Henrik Leopold. Text generation from business process models. In *CBSoft 2014 - Industry Track*, sep 2014.

[110] Raphael Rodrigues, Leonardo Azevedo, Kate Revoredo, and Henrik Leopold. A tool to generate natural language text from business process models. In *CBSoft 2014 - Tool Session*, sep 2014.

[111] Michael Rosemann. Potential pitfalls of process modeling: part a. *Business Process Management Journal*, 12(2):249–254, 2006.

[112] Alexander Salbrechter, Heinrich C Mayr, and Christian Kop. Mapping predesigned business process models to uml. In *Software Engineering and Applications: Proceedings of the Eighth IASTED International Conference*, 2004.

[113] August-Wilhelm Scheer and Markus Nüttgens. Aris architecture and reference models for business process management. In *Business Process Management*, pages 376–389. Springer, 2000.

[114] August-Wilhelm Scheer, Oliver Thomas, and Otmar Adam. Process modeling using event-driven process chains. *Process-Aware Information Systems*, pages 119–146, 2005.

[115] Ben Shneiderman, Richard Mayer, Don McKay, and Peter Heller. Experimental investigations of the utility of detailed flowcharts in programming. *Communications of the ACM*, 20(6):373–381, 1977.

[116] Keng Siau. Informational and computational equivalence in comparing information modeling methods. *Journal of Database Management (JDM)*, 15(1):73–86, 2004.

[117] Avik Sinha and Amit Paradkar. Use cases to process specifications in business process modeling notation. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 473–480. IEEE, 2010.

[118] Avik Sinha, Amit Paradkar, Palani Kumanan, and Branimir Boguraev. An analysis engine for dependable elicitation on natural language use case description and its application to industrial use cases. *IBM Report*, 2008.

[119] Amie L Souter, Lori L Pollock, and Dixie Hisley. Inter-class def-use analysis with partial class representations. *ACM SIGSOFT Software Engineering Notes*, 24(5):47–56, 1999.

[120] John F Sowa. Knowledge representation: logical, philosophical, and computational foundations. 1999.

[121] Michael Steinbach, George Karypis, Vipin Kumar, et al. A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 400, pages 525–526. Boston, 2000.

[122] Veselin Stoyanov, Claire Cardie, Nathan Gilbert, Ellen Riloff, David Buttler, and David Hysom. Reconcile: A coreference resolution research platform. 2010.

[123] Alexander Sturn, John Quackenbush, and Zlatko Trajanoski. Genesis: cluster analysis of microarray data. *Bioinformatics*, 18(1):207–208, 2002.

[124] Melanie Tory and Torsten Moller. Human factors in visualization research. *Visualization and Computer Graphics*, 10(1):72–84, 2004.

[125] UML. Uml ad - activity diagrams. http://www.uml-diagrams.org/activity-diagrams.html, 2014.

[126] Kees Van Deemter, Emiel Krahmer, and Mariët Theune. Real versus template-based natural language generation: A false opposition? *Computational Linguistics*, 31(1):15–24, 2005.

[127] Wil MP Van Der Aalst and Arthur HM Ter Hofstede. Yawl: yet another workflow language. *Information systems*, 30(4):245–275, 2005.

[128] Wil MP van der Aalst, Kees M van Hee, Arthur HM ter Hofstede, Natalia Sidorova, HMW Verbeek, Marc Voorhoeve, and Moe Thandar Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.

[129] TTGP van der Molen. Maintaining consistency between business process diagrams and textual documentation using the epsilon model management platform. 2011.

[130] Boudewijn F Van Dongen, Ana Karla A de Medeiros, HMW Verbeek, AJMM Weijters, and Wil MP Van Der Aalst. The prom framework: A new era in process mining tool support. In *International Conference on Application and Theory of Petri Nets*, pages 444–454. Springer, 2005.

[131] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. The refined process structure tree. *Data & Knowledge Engineering*, 68(9):793–818, 2009.

[132] GMA Verheijen and J Van Bekkum. Niam: an information analysis method. *Information Systems Design Methodologies: A Comparative Review, TW Olle, et al.(eds.)*, pages 537–590, 1982.

[133] Yannick Versley, Simone Paolo Ponzetto, Massimo Poesio, Vladimir Eidelman, Alan Jern, Jason Smith, Xiaofeng Yang, and Alessandro Moschitti. Bart: A modular toolkit for coreference resolution. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Demo Session*, pages 9–12. Association for Computational Linguistics, 2008.

[134] Jacques Wainer. Métodos de pesquisa quantitativa e qualitativa para a ciência da computação. *Atualização em informática*, pages 221–262, 2007.

[135] Harry Jiannan Wang, J Leon Zhao, and Liang-Jie Zhang. Policy-driven process mapping (pdpm): Discovering process models from business policies. *Decision Support Systems*, 48(1):267–281, 2009.

[136] Barbara Weber, Manfred Reichert, and Stefanie Rinderle-Ma. Change patterns and change support features–enhancing flexibility in process-aware information systems. *Data & knowledge engineering*, 66(3):438–466, 2008.

[137] Mathias Weske. *Business process management: concepts, languages, architectures*. Springer Publishing Company, Incorporated, 2010.

[138] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[139] William Winn. Contributions of perceptual and cognitive processes to the comprehension of graphics. *Advances in psychology*, 108:3–27, 1994.

[140] David Yarowsky, H Somers, R Dale, and H Moisl. Word-sense disambiguation. *Handbook of natural language processing*, pages 629–654, 2000.

[141] Tao Yue, Lionel C Briand, and Yvan Labiche. A use case modeling approach to facilitate the transition towards analysis models: Concepts and empirical evaluation. In *International Conference on Model Driven Engineering Languages and Systems*, pages 484–498. Springer, 2009.

[142] Tao Yue, Lionel C Briand, and Yvan Labiche. An automated approach to transform use cases into activity diagrams. In *Modelling Foundations and Applications*, pages 337–353. Springer, 2010.

[143] Tao Yue, Lionel C Briand, and Yvan Labiche. *Automatically deriving a UML analysis model from a use case model*. Citeseer, 2011.

[144] Stefan Zugal, Jakob Pinggera, and Barbara Weber. Assessing process models with cognitive psychology. In *EMISA*, volume 190, pages 177–182, 2011.

A. Appendix - Algorithms

Algorithm 1 identifySentenceType algorithm, from PortugueseLanguageProcessor implementation

```
1:  procedure identifySentenceType(plainSentence)
2:      elementType ← ProcessElementType.UNKNOW
3:      if (plainSentence ≠ ∅ and plainSentence.trim().isNotEmpty()) then ▷ Input validation
4:          if extractActivityFrom(plainSentence) ≠ ∅ then
5:              elementType ← ProcessElementType.ACTIVITY
6:          else
7:              eventProperties ← extractEventFrom(plainSentence)
8:              if eventProperties ≠ ∅ then
9:                  elementType ← eventProperties.getElementType()
10:             else
11:                 gatewayProperties ← extractGatewayFrom(plainSentence)
12:                 if gatewayProperties ≠ ∅ then
13:                     elementType ← gatewayProperties.getElementType()
14:                 else
15:                     splittedSentence ← plainSentence.split("\s+")
16:                     for word splittedSentence do
17:                         if ignoredWords.contains(word) then
18:                             throw new Exception("The sentence 'plainSentence' could not be mapped because the 'word' syntactic function was not present in the current language corpus.")
19:                         else
20:                             isSentenceRelevant ← sentenceRelevanceAnalysis(plainSentence)
21:                             if isSentenceRelevant = false then
22:                                 elementType ← ProcessElementType.NOT_RELEVANT
23:                             else
24:                                 elementType ← ProcessElementType.UNKNOW
25:     return elementType
```

Algorithm 2 extractActivityProperties algorithm, from PortugueseLanguageProcessor implementation

```
1:  procedure extractActivityProperties(plainSentence)
2:      activityProperties ← ∅
3:      i ← 0
4:      splittedSentence ← plainSentence.split("\s+")
5:      isFirstWordNoun ← labelHelper.isNoun(splittedSentence[i++])
6:      compositeNoun ← splittedSentence[i-1]
7:      if isFirstWordNoun then
8:          for  ; i < splittedSentence.length ; i+=2 do
9:              isPreposition ← labelHelper.isPreposition(splittedSentence[i])
10:             isNoun ← labelHelper.isNoun(splittedSentence[i+1])
11:             if isPreposition and isNoun then
12:                 compositeNoun ← compositeNoun + " " + splittedSentence[i] + " " + splittedSentence[i+1]
13:             else
14:                 break for
15:
16:     action ← splittedSentence[i++]
17:     businessObject ← splittedSentence[i]
18:     isSecondWordVerb ← labelHelper.isVerb(action)
19:     isThirdWordNoun ← labelHelper.isNoun(businessObject)
20:     if isFirstWordNoun and isSecondWordVerb and isThirdWordNoun then
21:         if labelHelper.isInfinitive(action) = false then
22:             action ← labelHelper.getInfinitiveOfAction(action)
23:         indexBeginAddition ← plainSentence.indexOf(businessObject) + businessObject.length()
24:         if indexBeginAddition < plainSentence.length() then
25:             addition ← plainSentence.substring(indexBeginAddition + 1)
26:         else
27:             addition ← ""
28:         activityProperties ← new ProcessActivityProperty(businessObject, compositeNoun, action, addition)
29:     return activityProperties
```

Algorithm 3 extractEventProperties algorithm, from PortugueseLanguageProcessor implementation

```
1: procedure extractEventProperties(plainSentence)
2:     eventProperties ← ∅
3:     cleanedPlainSentence ← textFormatter.removeStopWords(plainSentence)
4:     lowerCaseSentence ← cleanedPlainSentence.toLowerCase()
5:     eventProperties ← extractBeginEvent(lowerCaseSentence)
6:     if eventProperties = ∅ then
7:         eventProperties ← extractEndEvent(lowerCaseSentence)
8:     return eventProperties
```

Algorithm 4 extractGatewayProperties algorithm

```
1: procedure extractGatewayProperties(plainSentence)
2:     gatewayProperties ← ∅
3:     return gatewayProperties
```

Algorithm 5 processTextDiff algorithm

```
1: procedure processTextDiff(originalText, currentText)
2:     differences ← new List<SentenceDiff>()
3:     originalTextWithoutWhiteSpaces ← originalText.removeAll("\s+")
4:     currentTextTextWithoutWhiteSpaces ← currentText.removeAll("\s+")
5:     if originalTextWithoutWhiteSpaces.notEqualsIgnoreCase(currentTextWithoutWhiteSp
       then
6:         originalSentences ← originalTextWithoutWhiteSpaces.split("\.")
7:         currentSentences ← currentTextWithoutWhiteSpaces.split("\.")
8:         removedSentences ← processRemovals(originalSentences,
       currentSentences)
9:         insertedSentences ← processInserts(originalSentences,
       currentSentences)
10:        differences.addAll(removedSentences)
11:        differences.addAll(insertedSentences)
12:    return differences
```

Algorithm 6 processRemovals algorithm

```
1: procedure processRemovals(original, current)
2:     diffs ← new List<SentenceDiff>()
3:     for index ← 0; index < original.size(); index++ do
4:         originalSentence ← original.get(index)
5:         if current.notContains(originalSentence) then ▷ Verify if the old sentence exists within the new text
6:             diff ← new SentenceDiff(originalSentence, index, SentenceOperationType.DELETE)
7:             diffs.add(diff) ▷ If the sentence does not exists, then it was removed
8:     return diffs
```

Algorithm 7 processSentencesRecursivly algorithm, from SurfaceRealizer class

```
1: procedure processSentencesRecursivly(List<DSynTSentence> sentencePlan, RealizedText currentMainBranch, int mainBranchSenLevel)
2:     surfaceText ← ""
3:     while currentDsyntSentence < sentencePlan.size() do
4:         dsyntSentence ← sentencePlan.get(currentDsyntSentence)
5:         eFrag ← dsyntSentence.getExecutableFragment()
6:         mainBranchSenLevel ← eFrag.sen_level
7:         if lastLevel > mainBranchSenLevel then        ▷ checks if the secondary branch has finished
8:             currentMainBranch ← currentMainBranch.getFatherBranch() ▷ it means that all secondary paths (branches) are finished, then we must go back to the main branch
9:         if (mainBranchSenLevel > lastLevel or lastBulletNumber < eFrag.sen_bulletNumber) then
10:            if eFrag.sen_bulletNumber > 1 then        ▷ if bullet number > 1, then we must return to the previous mainBranch
11:                currentMainBranch ← currentMainBranch.getFatherBranch()
12:            lastBulletNumber ← eFrag.sen_bulletNumber
13:            newCurrentMainBranch ← new RealizedText(lHelper, languageProcessor)
14:            currentMainBranch.addBranchSentences(newCurrentMainBranch)
15:            newCurrentMainBranch.setFatherBranch(currentMainBranch)
16:            if currentDsyntSentence < sentencePlan.size()-1 then
17:                lastLevel ← mainBranchSenLevel
18:                surfaceText ← surfaceText + processSentencesRecursivly(sentencePlan, newCurrentMainBranch, mainBranchSenLevel)
19:        else
20:            textSentence ← realizeSentence(dsyntSentence, mainBranchSenLevel, lastLevel)
21:            surfaceText ← surfaceText + " " + textSentence
22:            currentMainBranch.addSentence(textSentence, dsyntSentence)
23:            currentDsyntSentence++
24:            lastLevel ← mainBranchSenLevel
25:    return surfaceText
```

Algorithm 8 processInserts algorithm

```
1: procedure processInserts(original, current)
2:     diffs ← new List<SentenceDiff>()
3:     for index ← 0; index < current.size(); index++ do
4:         currentSentence ← current.get(index)
5:         if original.notContains(currentSentence) then        ▷ Verify if the new sentence exists within the old text
6:             diff ← new SentenceDiff(currentSentence, index, SentenceOperationType.INSERT)
7:             diffs.add(diff)        ▷ If the sentence does not exists, then it was inserted
8:     return diffs
```

Algorithm 9 removeStopWords algorithm, from ILabelHelper interface specification

```
1:  procedure removeStopWords(String cleanText)
2:      if (cleanText ≠ ∅ and cleanText.trim().isNotEmpty()) then        ▷ Input validation
3:          discourseMarkers ← Localization.getInstance().getDiscourseMarkers()
4:          for marker ϵ discourseMarkers do
5:              if (cleanText.toLowerCase().contains(marker) then
6:                  cleanText ← cleanText.removeAll("(?i)" + marker)      ▷ Use of Regular expression for removing the marker from the string
7:          words ← cleanText.split("\s+")                               ▷ Tokenization regex
8:          for word ϵ words do
9:              word ← word.removeAll(".")   ▷ remove attached periods (e.g., .entao. -> entao)
10:             word ← word.removeAll(",")  ▷ remove attached commas (e.g., ,entao, -> entao)
11:             if discourseMarkers.contains(word.toLowerCase()) then
12:                 regex ← "(?i) + \s* + word + \s*"      ▷ where \s* = Regex which represents 0 or more white space characters (e.g., " ")
13:                 cleanText ← cleanText.removeAll(regex)        ▷ removes all the strings which match with the regex
14:             else if StopWordManager.getInstance().isStopWord(word) then
15:                 regex ← "(?i) + \s+ + word + \s+"      ▷ where \s+ = Regex which represents 1 or more white space characters (e.g., " ")
16:                 cleanText ← cleanText.removeAll(regex)
17:         cleanText ← cleanText.trim().replaceAll("\s+", " ")     ▷ remove trailing and excessive white spaces
18:         cleanText ← cleanTextFromEmptyComas(cleanText)     ▷ remove empty comas
19:     return cleanText
```

Algorithm 10 checkForConjunction algorithm (for Portuguese)

```
1:  procedure checkForConjunction(Label label, ILabelProperties props)
2:      splitLabel ← label.split(" ")
3:      if label.contains("e") then
4:          props.setIndexConjunction(label.indexOf("e")
5:          props.setHasConjunction(true)
6:          for word ϵ splitLabel.words do
7:              if word = "e" then
8:                  props.setIndexConjunction(word)
9:                  break for
10:     if (label.contains("/") and label.indexOf("/") < props.getIndexPrep)  then
11:         props.setIndexConjunction(label.indexOf("/"))
12:         props.setHasConjunction(true)
13:         for word ϵ splitlabel.words do
14:             if word = "/" then
15:                 props.setIndexConjunction(word)
16:                 break for;
17:     if label.contains(",") then
18:         props.setIndexConjunction(label.indexOf(","))
19:         props.setHasConjunction(true)
20:         for word ϵ splitlabel.words do
21:             if word.contains(",") then
22:                 props.setIndexConjunction(word);
23:                 break for
```

Algorithm 11 removeArticleFromBo algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1:  procedure removeArticleFromBo(String bo)
2:      splittedBo ← bo.split(" ")
3:      if (splittedBo.length > 1 and isArticle(splittedBo.words[0])) then
4:          bo ← bo.substring(splittedbo.words[0].length)
5:      return bo
```

Algorithm 12 getPrepositions algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1: procedure getPrepositions
2:     if prepositions = null then
3:         prepositions ← (LanguageData.noArticlePrepositions U
   LanguageData.articlePrepositions)
4:     return prepositions
```

Algorithm 13 getPrepositions algorithm (for English)

```
1: procedure getPrepositions
2:     if prepositions = null then
3:         prepositions ← "within", "into", "upon", "until", "on", "of", "for", "by",
   "from", "to", "at", "with", "w/o", "without", "in", "that", "using", "via", "during"
4:     return prepositions
```

Algorithm 14 isAdverb algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1: procedure isAdverb(String word)
2:     isAdverb ← false
3:     if getPos(word) = PortugueseLabelHelper.POS_ADV then
4:         isAdverb ← true
5:     return isAdverb
```

Algorithm 15 isVerb algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1: procedure isVerb(String word)
2:     isVerb ← false
3:     if getPos(word) = PortugueseLabelHelper.POS_VERB then
4:         isVerb ← true
5:     return isVerb
```

Algorithm 16 isNoun algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1: procedure isNoun(String word)
2:     isNoun ← false
3:     if getPos(word) = PortugueseLabelHelper.POS_NOUN then
4:         isNoun ← true
5:     return isNoun
```

Algorithm 17 isAdjective algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1: procedure isNoun(String word)
2:     isAdjective ← false
3:     if getPos(word) = PortugueseLabelHelper.POS_ADJ then
4:         isAdjective ← true
5:     return isAdjective
```

Algorithm 18 getInfinitive algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1: procedure getInfinitive(String conjugatedVerb)
2:     infinitiveForm ← conjugatedVerb
3:     if present3PP.containsValue(conjugatedVerb) then
4:         infinitiveForm ← lookForKey(present3PP, conjugatedVerb)
5:     else if present3PS.containsValue(conjugatedVerb) then
6:         infinitiveForm ← lookForKey(participle, conjugatedVerb)
7:     return infinitiveForm
```

Algorithm 19 getParticiple algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1: procedure getParticipleForm(String infinitive)
2:     participleForm ← participle.get(infinitive.toLowerCase())
3:     if participleForm = ∅ then
4:         stemmedVerb ← steamPrefix(infinitive)
5:         if stemmedVerb ≠ ∅ then
6:             participleForm ← prefix + getParticipleForm(stemmedVerb)
7:     if participleForm = ∅ then
8:         participleForm ← infinitive
9:     return participleForm
```

Algorithm 20 getPresentForm algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1: procedure getPresentForm(String infinitive, Boolean boisSubject, Boolean
   boisPlural)
2:     presentForm ← ∅
3:     if boisSubject and boisPlural then
4:         presentForm ← present3PP.get(infinitive.toLowerCase())
5:     else
6:         presentForm ← present3PS.get(infinitive.toLowerCase())
7:     if presentFormm = ∅ then
8:         stemmedVerb ← steamPrefix(infinitive)
9:         if stemmedVerb ≠ then
10:            presentForm ← prefix + getPresentForm(stemmedVerb)
11:    if presentForm = ∅ then
12:        presentForm ← infinitive
13:    return presentForm
```

Algorithm 21 is3PS algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1: procedure is3PS(String word)
2:     is3PS ← false
3:     if present3PS.contains(word) then
4:         is3PS ← true
5:     return is3PS
```

Algorithm 22 transformToSingularForm algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1: procedure transformToSingularForm(String pluralNoun)
2:     for pluralSuffix ϵ pluralSinguarMap.keySet do
3:         if pluralNoun.endsWith(pluralSuffix) then
4:             singularSuffix ← pluralSinguarMap.get(pluralSuffix)
5:             singularNoun ← pluralNoun.replace(pluralSuffix,
   singularSuffix)
6:     return singularNoun
```

Algorithm 23 getGender algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1:  procedure getGender(String noun, Boolean isPlural)
2:      gender ← LanguageData.Gender_FEM
3:      tempNoun ← noun.toLowerCase()
4:      if tempNoun.contains(" ") then                ▷ checks if it is composite noun
5:          tempNoun ← tempNoun.split(" ")[0]
6:      if isPlural then
7:          tempNoun ← transformToSingularForm(tempNoun)
8:      if genderMap.containsKey(tempNoun) then
9:          if genderMap.get(tempNoun).equals("f") then
10:             gender ← LanguageData.Gender_FEM
11:         else
12:             gender ← LanguageData.GENDER_MAS
13:     else                                          ▷ gender is not mapped, try to infer it
14:         for femSuffix ϵ LanguageData.feminineSuffixes do
15:             if tempNoun.endsWith(femSuffix) then
16:                 gender ← LanguageData.GENDER_FEM
17:         for mascSuffix ϵ LanguageData.masculineSuffixes do
18:             if tempNoun.endsWith(mascSuffix) then
19:                 gender ← LanguageData.GENDER_MAS
20:     return gender
```

Algorithm 24 getArticle algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1:  procedure getArticle(String noun, Boolean isObject, Boolean isIndef, Boolean
    isPlural)
2:      article ← ∅
3:      articleIndex ← getGender(noun, isPlural)
4:      if isPlural then
5:          articleIndex++
6:      if isObject and isIndef then
7:          article ← LanguageData.indefArticles[articleIndex]
8:      else
9:          article ← LanguageData.defArticles[articleIndex]
10:     return article
```

Algorithm 25 isDefArticle algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1: procedure isDefArticle(String word)
2:     isDefArticle ← false
3:     if LanguageData.defArticles.contains(word) then
4:         isDefArticle ← true
5:     return isDefArticle
```

Algorithm 26 isPronoun algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1: procedure isPronoun(String word)
2:     isPronoun ← false
3:     if LanguageData.pronouns.contains(word) then
4:         isPronoun ← true
5:     return isPronoun
```

Algorithm 27 getPronouns algorithm (for Portuguese) from PortugueseLabelHelper implementation

```
1: procedure getPronouns
2:     if pronouns = ∅ then
3:         pronouns ← LanguageData.demPronouns U
   LancuageData.pronouns
4:     return pronouns
```

Algorithm 28 posTagging algorithm

```
1:  procedure posTagging(words)
2:      if sytaticRoleMap = ∅ then
3:          syntacticRoleMap ← new Map<String, Integer>()
4:      for word ϵ words do
5:          if syntacticRoleMap.notContains(word) then
6:              if isVerb(word) then
7:                  syntacticRoleMap.put(word, PortugueseLabelHelper.POS_VERB)
8:              else if isNoun(word) then
9:                  syntacticRoleMap.put(word, PortugueseLabelHelper.POS_NOUN)
10:             else if isAdjective(word) then
11:                 syntacticRoleMap.put(word, PortugueseLabelHelper.POS_ADJ)
12:             else if isAdverb(word) then
13:                 syntacticRoleMap.put(word, PortugueseLabelHelper.POS_ADVERB)
14:             else
15:                 ignoredWords.add(word)
16:                 words.remove(word)
17:     return syntacticRoleMap
```

B. Appendix - Process Models and Textual Descriptions

This appendix presents some simple process, written in both Portuguese and English, that were used as input for the tool and the natural language text, was generated as the output. After automatically generating the process textual description from the model, it was then submitted to the framework for asserting if the model produced as output matched the original process model. These models were used to test if the tool was capable of generating texts for both languages correctly.

The first process, depicted by Figure B.3 and Figure B.4, presents a simple exam application process. The main actors of this process are: Teacher (Professor), Secretary (Secretaria) and Student (Aluno). First, the teacher prepare the exam. Then, the student do the exam. After that, the teacher receives the exam and corrects it. After the correction, the teacher deliver the grade to the secretary, which input the grade in the student's grades history. Tables B.1 and B.2 provides the textual description generated by the framework for the models used as input.

Table B.1: English Natural Language Text generated by analyzing the process model data and extracting textual information.

The process begins when the Teacher creates the test. Then, the Student does the test. Afterwards, the Teacher corrects the test. Subsequently, the Teacher delivers grade to the Secretary. Then, the Secretary saves grade to the Student's profile. Finally, the process finishes.

The second process, depicted by Figure B.3 and Figure B.4, represents a simple request order received by the Secretary. The main actors are: Secretary and Manager. The process starts when the secretary receives a request order from a client. If the client is already in the system, the secretary update the client's data. After that, the manager receives the request order and conclude the request. Finally, the process finishes. Tables B.9 and B.4 provides the textual description generated by the framework for the models used as input.

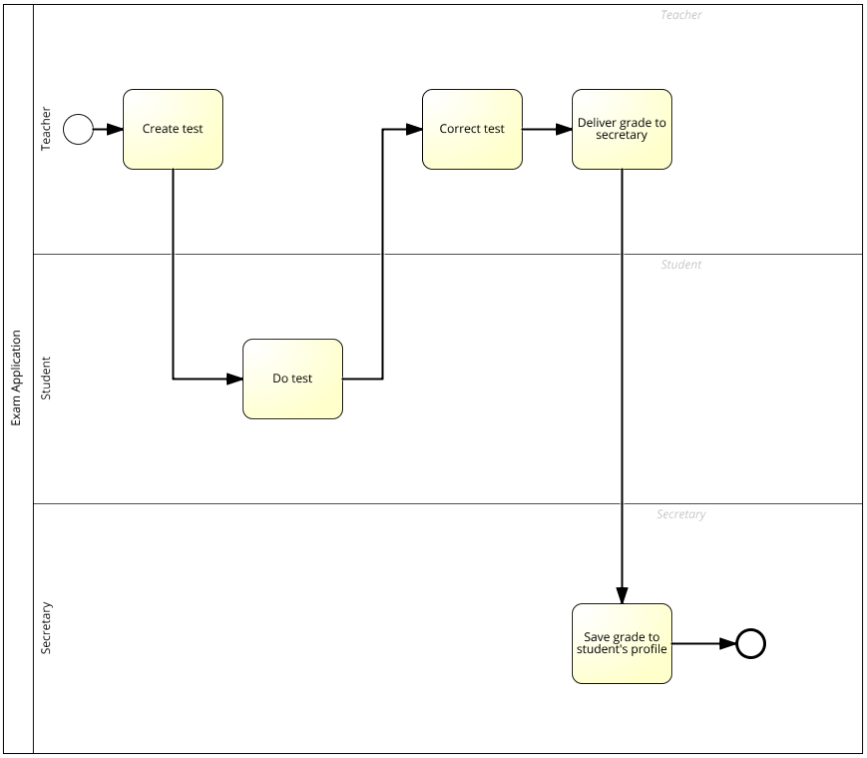


Figure B.1: Simple Exam application process, written in English.

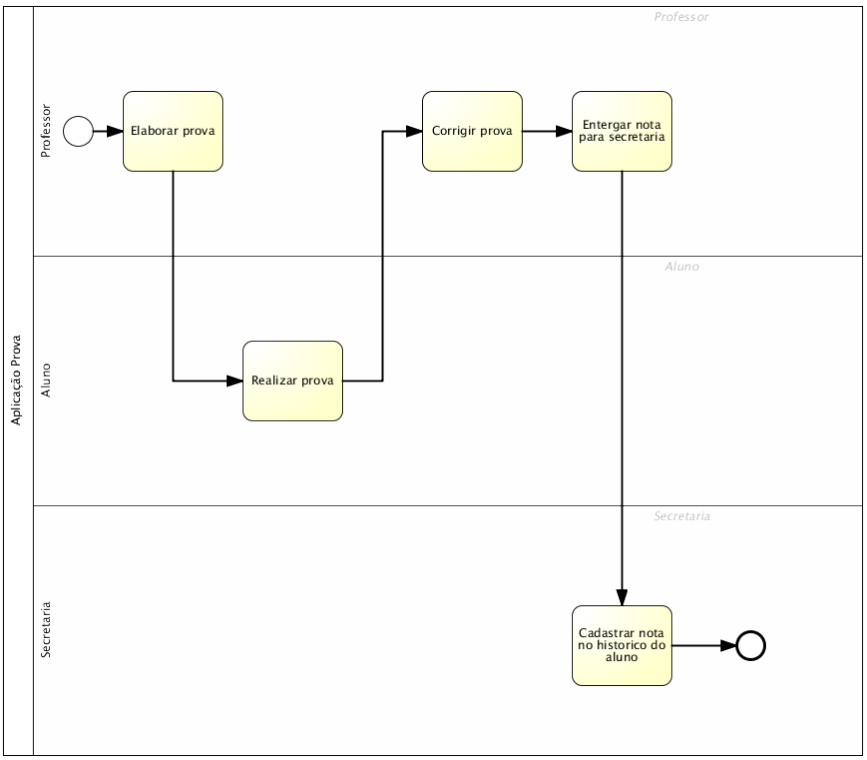


Figure B.2: Simple Exam application process, written in Portuguese.

Table B.2: Portuguese Natural Language Text generated by analyzing the process model data and extracting textual information.

O processo começa quando o Professor elabora a prova. Então, o aluno realiza a prova. Em seguida, o professor corrige a prova. Subsequentemente, o professor conduz a atividade de entregar nota para secretaria. Então, a secretaria cadastra a nota no histórico do aluno. Finalmente, o processo é terminado.

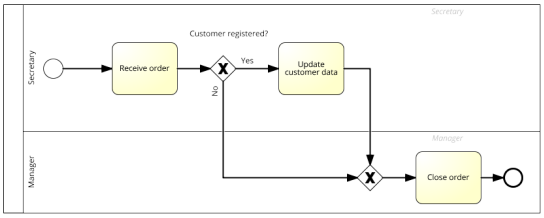


Figure B.3: Simple Secretary process, written in English.

Table B.3: English Natural Language Text generated by analyzing the process model data and extracting textual information.

The process begins when the Secretary receives an order. In case the Customer is registered, the Secretary updates the customer data. Then, the Manager closes the order. Finally, the process finishes.

Table B.4: Portuguese Natural Language Text generated by analyzing the process model data and extracting textual information.

O processo começa quando o Secretário recebe o pedido de compra. Caso o cliente é cadastrado, o Secretário atualiza o cadastro. Então, o gerente finaliza o pedido. Finalmente, o processo termina.

The third process, depicted by Figure B.5 and Figure B.6, represents a hotel room service process. The main actors are: Waiter, Barman, Kitchen and Room Service Manager.

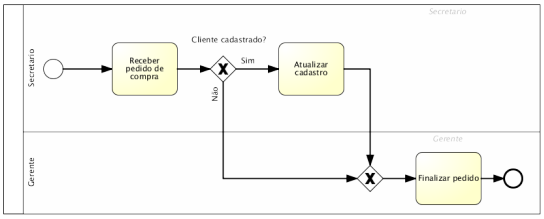


Figure B.4: Simple Secretary process, written in Portuguese.

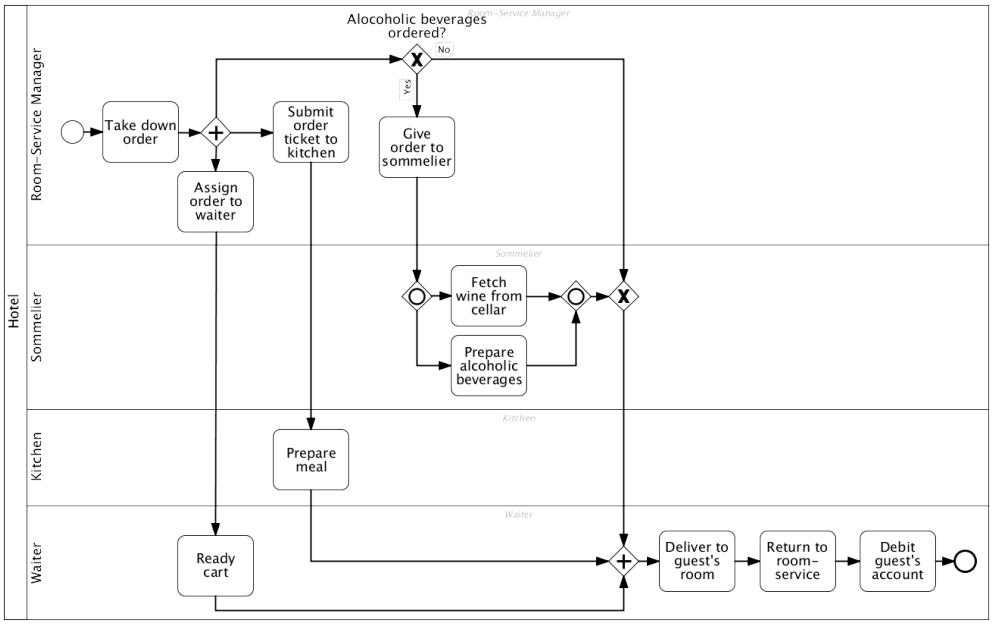


Figure B.5: Hotel Service process, written in English.

Tables B.5 and B.6.

Table B.5: English Natural Language Text generated by analyzing the process model data and extracting textual information.

The process begins when the Room-Service Manager takes down an order. Then, the process is split into 3 parallel branches:

- In case alcoholic beverages are ordered, the Room-Service Manager gives order to the Sommelier. Afterwards, one or more of the following paths are executed:
  - The Sommelier fetches wine from the cellar.
  - The Sommelier prepares the alcoholic beverages.
- The Room-Service Manager submits the order ticket to the Kitchen. Subsequently, the Kitchen prepares the meal.
- The Room-Service Manager assigns order to the Waiter. Then, the Waiter readies the cart.

As long as all the 3 branches were executed, the Waiter delivers to the guest's room. Afterwards, the Waiter returns to the room-service. Subsequently, the Waiter debits from the guest's account. Finally, the process finishes.

The forth process, depicted by Figure B.7 and Figure B.8, represents a bread delivery

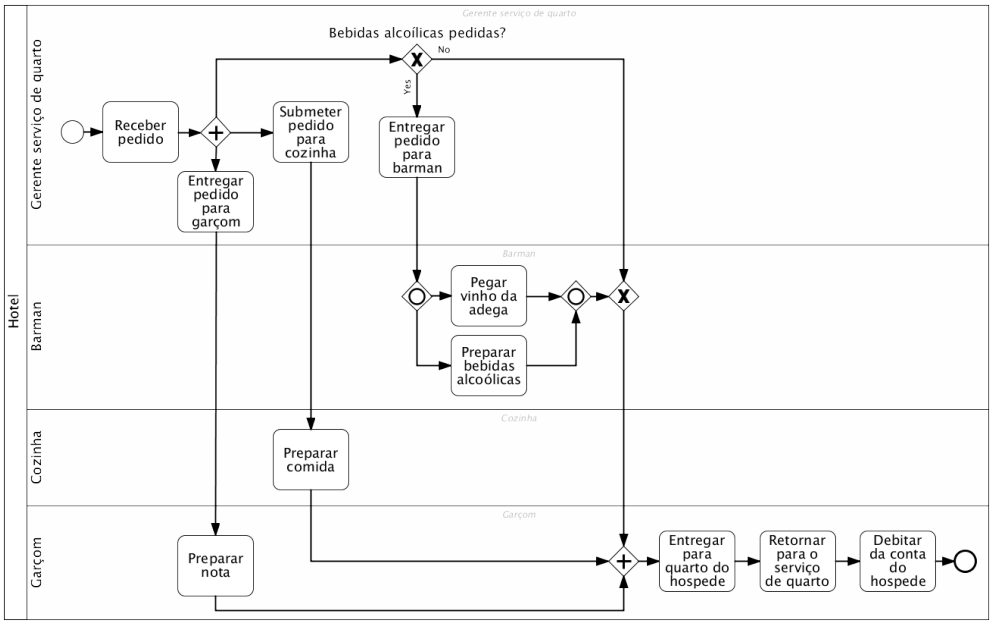


Figure B.6: Hotel Service process, written in Portuguese.

Table B.6: Portuguese Natural Language Text generated by analyzing the process model data and extracting textual information.

O processo começa quando o Gerente serviço de quarto recebe o pedido. Então, o processo é dividido em 3 ramificações paralelas:

- Caso bebidas alcoólicas são pedidas, o Gerente serviço de quarto entrega o pedido para o barman. Em seguida, os seguintes caminhos são executados:
  - O Barman pega o vinho da adega.
  - O Barman prepara as bebidas alcoólicas.
- O Gerente serviço de quarto entrega o pedido para o garçom. Subsequentemente, o garçom prepara a nota.
- O Gerente serviço de quarto submete o pedido para cozinha. Então, a cozinha prepara a comida.

O Garçom entrega para o quarto do hospede. Em seguida, o garçom retorna para serviço de quarto. Subsequentemente, o garçom debita da conta do hospede. Finalmente, o processo é terminado.

subscription process. The main actors are: Customer, Financial Department, Deliver Service and Marketing. In a nutshell, the process describes a subscription service that allows

Table B.7: English Natural Language Representation for the Bread delivery service subscription .

The process begins when there is advertising available for the customer. Then, the customer understands the proposal and understands the signature service. Afterwards, the customer verifies if he is interested:
- If he is not interested, then the process finishes.
- If he is interested, then he submits his personal data. Afterwards, he checks whether or not the region is covered:

- If the region is not covered, then the process finishes.
- If the region is covered, then he submits the signature and chooses the plan. Afterwards, he checks if the plan is suitable:
  - If the plan is not suitable, then the Marketing department registers disinterest. Finally, the process finishes with the signature canceled.
  - If the plan is suitable, the customer configures the signature. Afterwards, he executes one or more of the following branches:
    * The customer pays with money.
    * The customer uses a voucher for discount.
    * The customer pays with credit card.
  - Afterwards, the Financial Department processes the payment and verifies if the payment is ok:
    * If the payment is successful, then the Financial Department notifies the payment. Afterwards, the Delivery Service schedules the delivery. Subsequently, the process is divided into two parallel branches:
      · The Deliver Service delivers the bread to the customers
      · The Deliver Service receives the payments.
    * If the payment is not ok, then the Marketing department registers disinterest. Finally, the process finishes with the signature canceled.
    * Once all the branches are executed, the process finishes with the signature saved.

customers to sign to a bread delivery service for receiving bread at home according to the customer preferences. Tables B.7 and B.8 provides the textual description generated by the framework for the models used as input.

The fifth process depicted by Figure B.9 and Figure B.9, represents a claims handling process. This process was provided by Queensland University of Technology[1] (QUT). The main actors are: handling department and Notification department. It can be described as follows: The process starts when a customer submits a claim by sending in relevant documentation. The Notification department at the car insurer checks the documents upon completeness and registers the claim. Then, the Handling department picks up the claim and checks the insurance. Then, an assessment is performed. If the assessment is positive, a garage is phoned to authorize the repairs and the payment is scheduled (in this order). Otherwise, the claim is rejected. In any case (whether the outcome is positive or negative), a letter is sent to the customer and the process is considered to be complete.

[1]University located in Brisbane, Australia, home page is accessible under https://www.qut.edu.au/

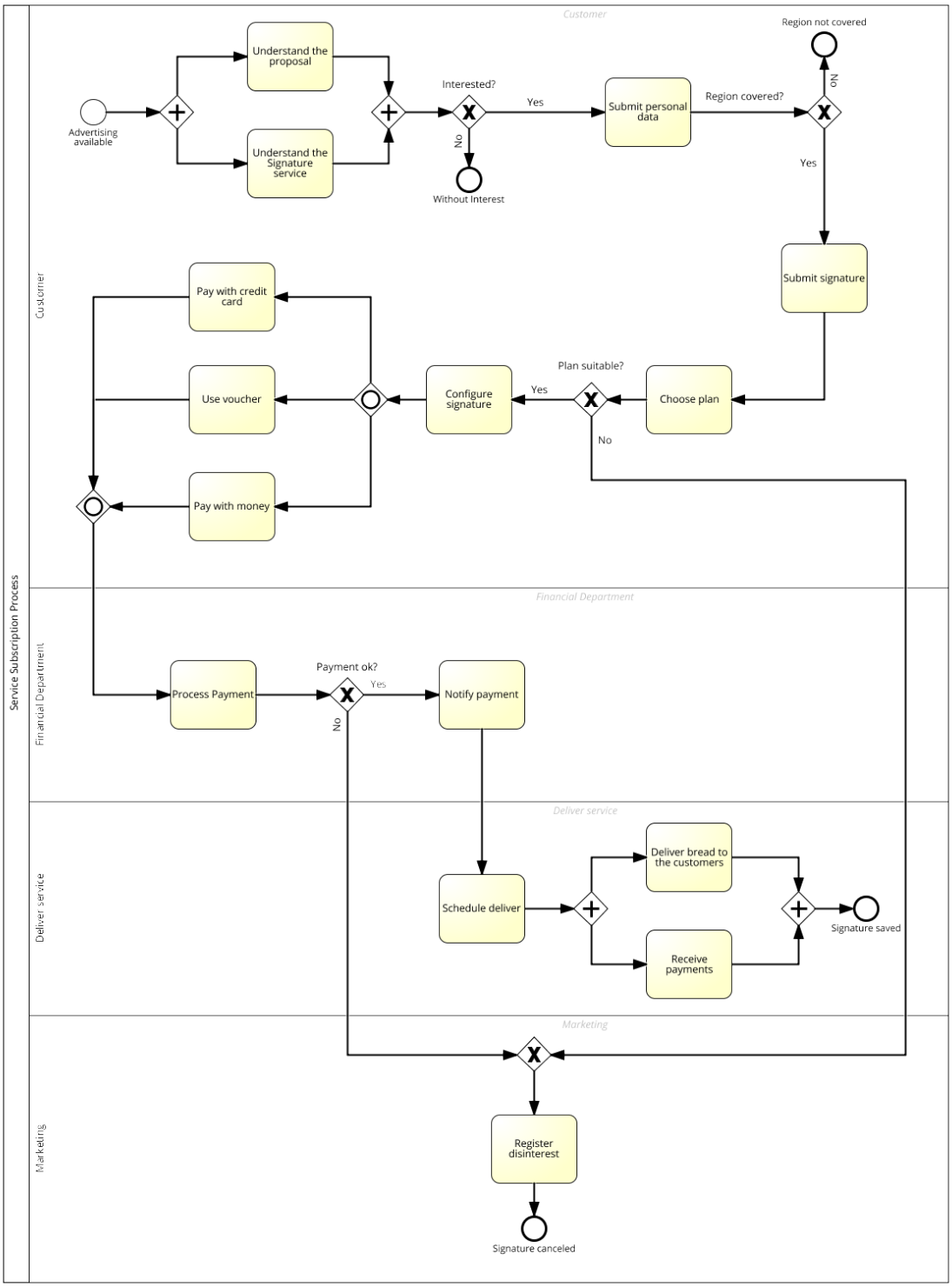


Figure B.7: English Bread delivery service subscription in BPMN.

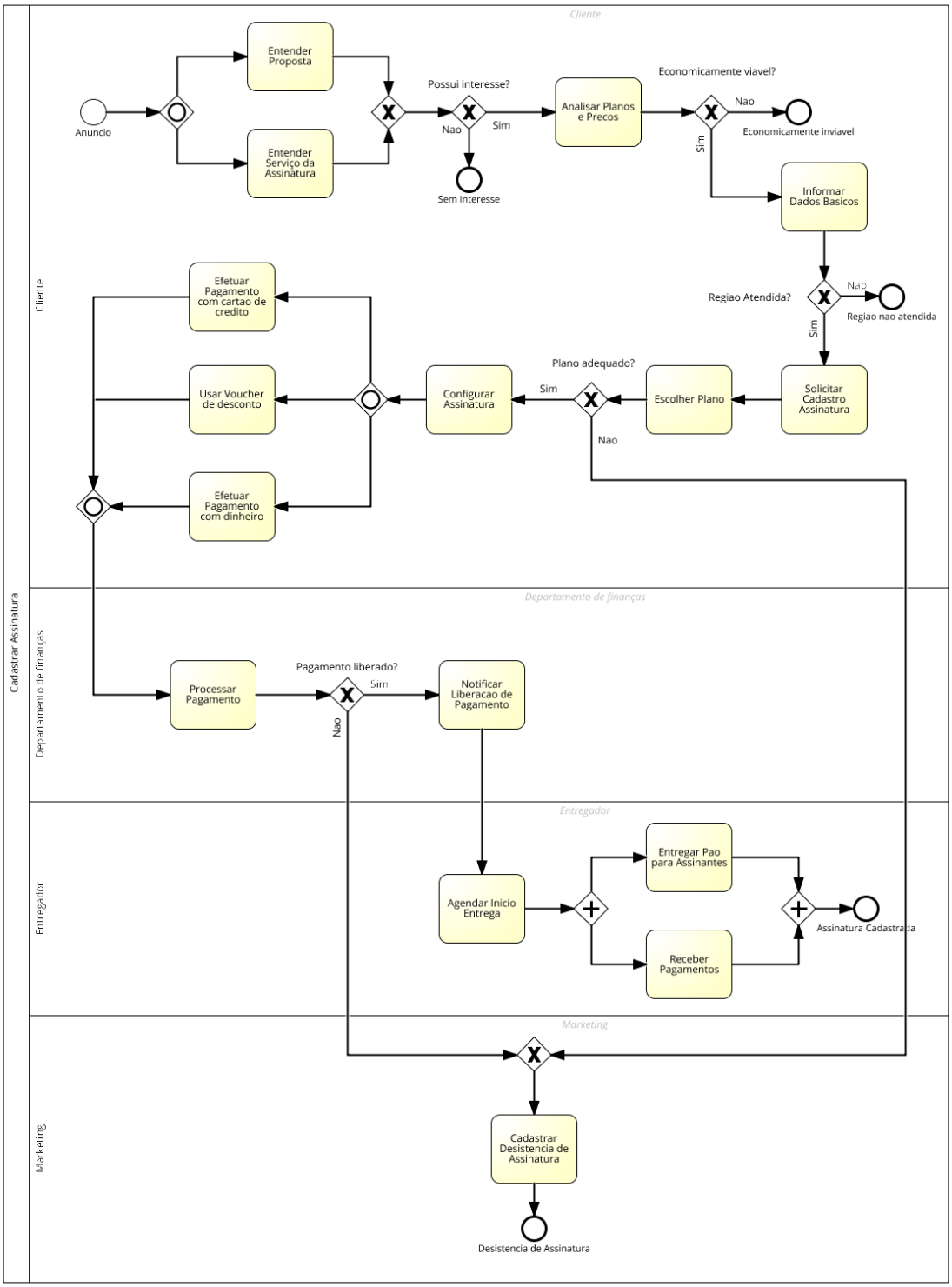


Figure B.8: Portuguese Bread delivery service subscription in BPMN.

Table B.8: Portuguese Natural Language Representation for the Bread delivery service subscription .

O processo começa quando há um anúncio disponível para o cliente. Então, o cliente entende a proposta e entende o serviço da assinatura. Subsequentemente, o cliente verifica se possui interesse:
- Se não possuir interesse, então o processo termina.
- Se possuir interesse, então ele informa os dados básicos. Após informar os dados básicos, ele verifica se a região é atendida:

- Se a região não for atendida, então o processo termina.
- Se a região for atendida, então ele solicita o cadastro na assinatura e escolhe o plano. Depois disso, ele verifica se o plano é adequado:
  - Se o plano não for adequado, então o Marketing cadastra a desistência de assinatura. Em seguida, o processo termina com a desistência de assinatura.
  - Se o plano for adequado, o cliente configura a assinatura. Após configurar a assinatura, ele executa uma ou mais das seguintes atividades:
    * O cliente efetua o pagamento com dinheiro.
    * O cliente efetua o pagamento com cartão de credito.
    * O cliente usa um voucher de desconto.
  - Em seguida, o Departamento de finanças processa o pagamento. Após processar o pagamento, o departamento verifica se o pagamento foi liberado:
    * Se o pagamento foi liberado, então o Departamento de finanças notifica a liberação de pagamento. Então, o Entregador agenda o início da entrega. Subsequentemente, o processo é dividido em duas ramificações paralelas:
      · O Entregador entrega o pão para os assinantes.
      · O Entregador recebe os pagamentos.
    * Se o pagamento não foi liberado, então o Marketing cadastra a desistência de assinatura. Em seguida, o processo termina com a desistência de assinatura.
    * Quando ambas as ramificações tiverem sido completadas, o processo termina com a assinatura cadastrada.

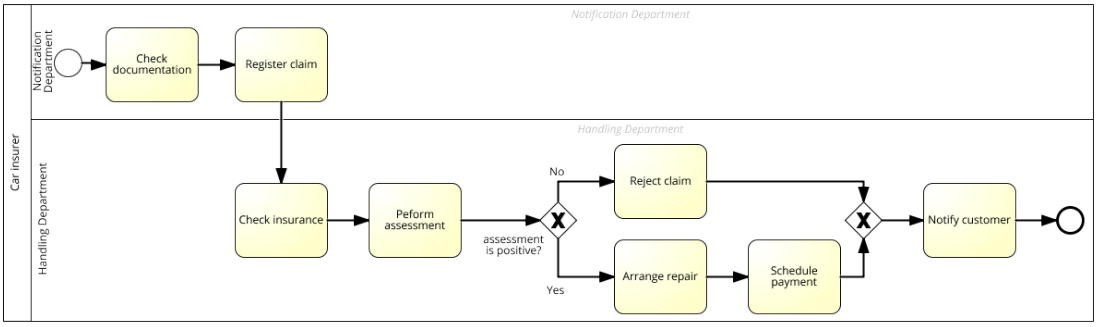


Figure B.9: claims handling process .

Table B.9: English Natural Language Text generated by analyzing the Claims Handling process model and extracting textual information.

The process begins when the Notification Department checks the documentation. Next, the Notification Department registers the claim. Afterwards, the Handling Department checks the insurance and performs the assessment. Following, the Handling Department verifies if the assessment is positive:

- If the assessment is positive, then the Handling Department arranges the repair. Subsequently, it schedules the payment.
- If the assessment is negative, then the Handling Department rejects the claim.

The Handling Department notifies the customer. Finally, the process finishes.

C. Appendix - Extending the Framework to new languages

To add support to a new language, the following steps must be executed:

- Creation of a Project responsible for the implementation of the several interfaces. None of the classes in the main project needs to be created or modified (with the exception of the configuration class, explained in the second step). The interfaces to be implemented in the new project are:
  - ILabelDeriver
  - ILabelHelper
  - ILabelProperties
  - ISurfaceRealizer
  - INaturalLanguageProcessor
  - ITextFormatter

  In this thesis, the Project "*RealizerPort*" corresponding to the `PortugueseRealizer` package was developed, to the manipulation of business process models written in Portuguese.

- Extension of the localization modules through the addition of a dictionary that contains the respective translations for the given Keys, defined in the localization module. In order to do that, its enough to create a text file with some structure similar to the one presented in Figure C.1 with the keys and their translations for the given language. The file should be saved into the localization module directory. The file name must be named according to the pattern "*Dictionary_NameOfLanguage*", for example, "*Dictionary_Portuguese*". The module searches, during execution time, for the respective dictionary for the current language and loads the translations to be used during the process of natural text generation from process models.

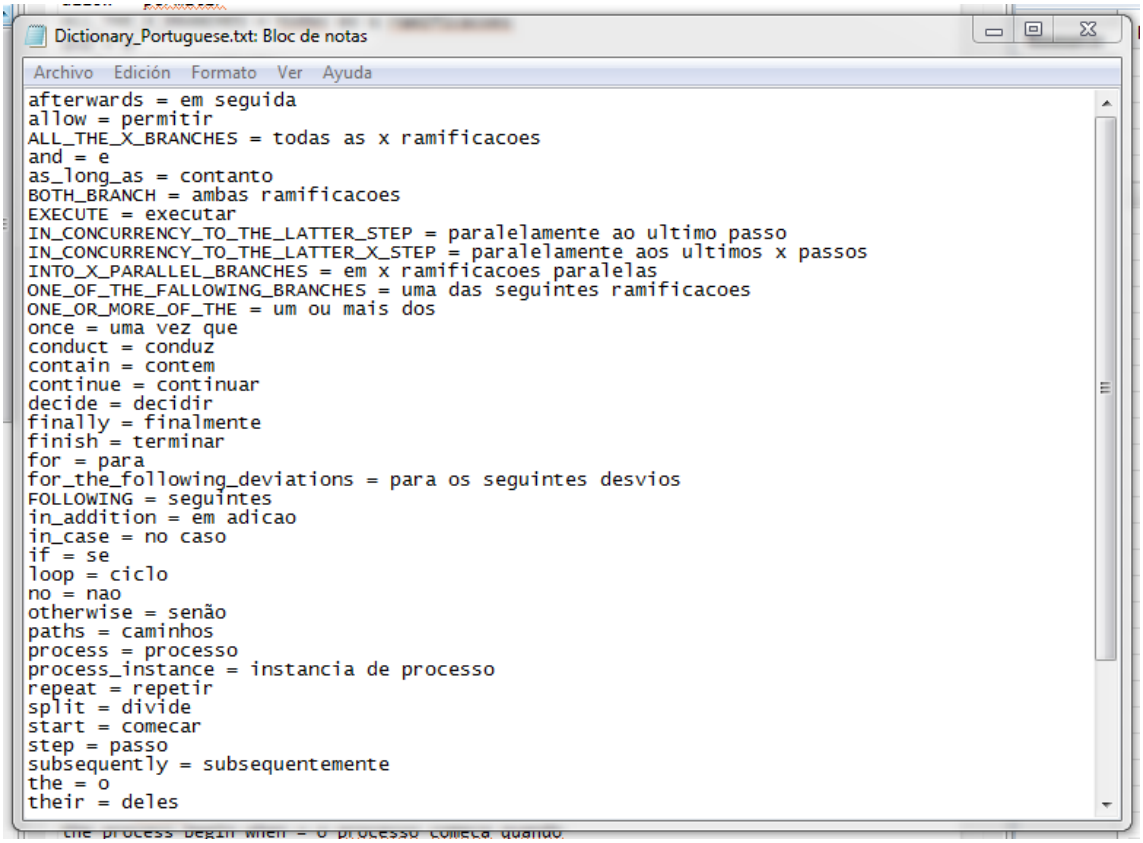


Figure C.1: Dictionary file structure. In this example, it is shown a Portuguese dictionary file.

- Add reference to the new Project or library responsible for the implementation of the operations defined in the Project `GeneralLanguageCommon` in the method "*SetCurrentLanguage*" of the `LanguageConfig` class in the main Project. An improvement to be done is reading the necessary configurations through a *.xml* or *.txt* file, by some sort of dependency injection mechanism. With this modification, it will not be necessary any change in the main Project, responsible for the framework's execution.

- All the other components do not have to be modified. In other words, regarding the NLG pipeline, the only step that needs to be changed for the new language is the last one (Message Realization). All the necessary algorithms to the macro steps of Text Planning and Sentence Planning are used through the interfaces defined in the `LabelAnalysis` package of the generic project `GeneralLanguageCommon`. Regarding the NLP pipeline, there is no need to apply any change besides implementing both NLP interfaces which are *INaturalLanguageProcessor* and *ITextFormatter*.