FACULTAT D'INFORMÀTICA DE BARCELONA

Model-Agnostic Process Modeling

Master Thesis

Master in Innovation and Research in Informatics

Data Mining and Business Intelligence

June 18th, 2018

Student: Josep Sànchez Ferreres
Advisors: Josep Carmona Vargas
Lluís Padró Cirera

Abstract

Modeling techniques in Business Process Management (BPM) often suffer from low adoption due to the variety of profiles found in organizations. This project aims to provide a novel alternative to BPM documentation with *Annotated Textual Descriptions* (ATD). ATD are based on text annotations, which can be understood by non-technical users, but are still a robust alternative for business process automation.

In order to deploy ATD-based solutions in BPM systems, the ATD reference implementation, *ATDlib*, has been developed as part of this project. The current implementation is able to parse and manage ATD, automatically generate annotations, and convert ATD into other business process representations.

Finally, a practical application of ATD, the *Model Judge*, is presented. Model Judge is a web platform that can students learn to model business processes and guide them with accurate diagnostics. The tool has been tested in two pilot university courses and validated using the analytics collected during the sessions.

Acknowledgements

This Master Thesis would have not been possible without the invaluable collaboration of many people. I want to thank everyone for their support and all the good ideas that have arisen as part of this project.

First of all, I would like to acknowledge my advisors, Josep Carmona and Lluís Padró, for their feedback and support, and also for the opportunity they have given me to work in their department as a research assistant, and as a future Ph.D. student to continue the work we started in this thesis. Special thanks go as well to all my colleagues from the office, the *NLP4BPM* project, and especially to Luís Delicado, for his work on *Model Judge*.

Part of the work presented is the fruit of collaboration between our research group and two other universities, Denmark's Technical Univeristy (DTU) and the Universidad Católica de Santa María. I want to thank Barbara Weber, Andrea Burattin, Amine Abbad and Guillermo Calderón for their feedback, ideas and for running the two Model Judge pilot courses in their universities.

And last, but not least, I want to give special thanks to my parents for their invaluable support. Also my family, friends and of course Montse, who stood next to me during this long journey.

Contents

1 Introduction **4**
1.1 Motivation . . . 4
1.2 Project Scope . . . 5
1.3 Structure of this Document . . . 6

2 Background **7**
2.1 Business Process Management . . . 7
2.1.1 Business Process Notations . . . 8
2.2 Natural Language Processing . . . 9
2.2.1 Steps of Language Analysis . . . 10
2.2.2 Text Annotations . . . 11
2.3 The *NLP4BPM* Project . . . 12

3 Describing Processes with Natural Language **15**
3.1 Running Example . . . 15
3.2 Annotated Textual Descriptions . . . 15
3.3 ATD Semantics . . . 18
3.3.1 Annotation Types . . . 18
3.3.2 Relation Types . . . 19
3.3.3 Automatic Reasoning . . . 20
3.4 Possible Applications of ATD . . . 20

4 *ATDlib*: A Library to Manage and Generate ATD **22**
4.1 Features and Architecture . . . 22
4.1.1 Converting ATD to Freeling Semantic Graphs . . . 23
4.2 Creating an ATD with *ATDlib* . . . 24

5 *Model Judge*: A Practical Use Case of ATD **27**

5.1 The *Model Judge* Platform . . . 27

5.1.1 Diagnostics . . . 28

5.2 A Tour through *Model Judge* . . . 29

5.3 Approach . . . 30

5.3.1 Process Model Analysis . . . 31

5.3.2 ATD Exensions for Model Judge . . . 32

5.3.3 Diagnostics Generation . . . 33

5.4 Aligning ATD and BPMN Process Models . . . 34

5.4.1 Feature Extraction . . . 35

5.4.2 Similarity Computation . . . 36

5.4.3 Alignment Creation . . . 36

5.4.4 Predictor-based Refinement . . . 37

5.5 Experiments . . . 38

5.5.1 Analysis of the Modeling Session Data . . . 39

5.5.2 Student Survey . . . 43

6 Conclusions and Future Work **44**

A Source Code **49**

List of Figures and Tables

2.1 The three abstraction levels of business process management. Adapted from the figure in [10]. 8

2.2 Example process in different graph-based execution languages: BPMN (top-left) YAWL (bottom-left) BPEL (right). 9

2.3 One of the example annotations provided with the Brat annotation tool. 12

2.4 One of the example annotations provided with the TAG annotation tool. 13

2.5 Example execution of the BPMNvsText module at the NLP4BPM web interface. 14

3.1 Textual description of a patient examination process. 16

3.2 Process model for the patient examination process. 17

4.1 ATDlib feature overview 24

4.2 UML diagram showing the public interface used for ATD representation 25

4.3 The brat user interface: (left) An automatically generated ATD. (right) After the user performs a manual validation. 26

4.4 Workflow of the creation of an ATD using ATDlib 26

5.1 Model Judge modeling view 30

5.2 *Model Judge* process overview 31

5.3 Overview of the alignment technique presented in [33] 35

5.4 Three characteristic behaviors observed in the modeling sessions. The blue circles represent simple validations, while green squares denote complete validations. 40

5.5 Evolution of the diagnostic types for the modeling session. 40

5.6 Density plot of the diagnostic lifetimes variable. 41

5.7 Results of the student survey for the modeling courses. 42

3.1 Example fragment of Annotated Textual Description with annotations (left) and relations (right). 18

5.1 Pearson's correlation test results for the relevant pairs of studied variables (shown per course and total). 43

Chapter 1

Introduction

1.1 Motivation

Business Process Management (BPM) is an area in operations research with the goal of modeling, analyzing and optimizing *business processes*. One of the main pillars of BPM are *process models*. A process model represents how a certain task is performed in an organization, and as such, can be used to efficiently communicate the internal operations of a company.

In the era of information systems and artificial intelligence, the focus of process models has progressively shifted from *human readability* to *machine readability* [1]. We have moved from the traditionally graphical models for business process documentation, such as the flow chart, to more formal standards built on top of machine-readable languages, like the Business Process Model and Notation (BPMN) [2] standard. This has enabled automation for the analysis and improvement of business processes, which in turn translates in real value for companies using these techniques.

However, this formalization of the semantics of process models results in an increase in modeling languages' complexity, which in turn, results in non-experts –stakeholders and employees alike– having difficulties in understanding the processes [3]. These misunderstandings may have serious consequences for any organization [4]

For the above reasons, no standard for process modeling has ultimately stood above the others. This has lead to a scenario where BPM-focused organizations often have to maintain multiple representations of the same process model. Informal versions, which heavily rely on semi-structured or unstructured natural language, are kept next to more formal and structured alternatives. Furthermore, in this formal spectrum, we encounter several modeling languages, each one aiming to solve a different subset of problems [5, 6].

In this work, we propose an alternative to conventional process modeling languages that aims to be intuitive and ergonomic, but at the same time acts as a formal representation enabling the automation of analysis, monitoring, or simulation of business processes. Our technique combines the flexibility of unstructured human langauge with Natural Language Processing (NLP) techniques in order to create an Annotated Textual Description (ATD) that represents the business process. Additionally, we present a practical application of ATD with *Model Judge*, a web platform designed to aid novices in the creation of business process models which uses ATD as one of the core ideas behind its algorithm.

1.2 Project Scope

The work presented in this Master Thesis project has been divided into the following activities:

Formalization of ATD and its semantics

The goal of this task is the creation of a robust language with well defined semantics that is able to represent business processes based on annotated natural language descriptions.

As part of this task, we have explored the common characteristics in business process descriptions, in order to identify the relevant types of information for the documents in our problem domain.

Next, the identified concepts have been translated in terms of an annotation language: Annotations, and the set of relations between them.

Development of *ATDlib* and Integration with Annotation Tools

This task consisted in the development of a hybrid Clojure/Java library, *ATDlib*. This library is intended to be a set of software tools to manage and generate ATD process descriptions.

The two main functionalities implemented in *ATDlib* are the automatic generation of partial ATD from text and the translation between ATD and FreeLing [7] semantic graphs.

In order to operationalize the workflow presented in this work for the generation of ATD, the functionalities presented in *ATDlib* must be integrated into a graphical annotation tool. As part of this task, we explored two alternatives in the NLP field: Brat [8] and TAG [9].

Design and Development of the *Model Judge*'s Algorithm

This activity consisted in the design of an algorithm behind *Model Judge*, a web platform designed to aid novice students in the process of process modeling. This algorithm is based on the notion of Annotated Textual Descriptions that was formalized and developed in the two previous tasks.

Analysis of the *Model Judge* Analytics Data

In order to gain insights and evaluate the *Model Judge* platform, we collected analytics information from two courses that took place in Denmark's Technical University (DTU) and the Universidad Catolica de Santa María (UCSM), in Peru, which used *Model Judge* as the main modeling tool.

This task consisted in the preprocessing and analysis of the data collected from the platform.

1.3 Structure of this Document

The remainder of this document is structured as follows: Chapter 2, discusses the state of the art in the several fields involved in this project and introduces all the necessary background concepts. Chapter 3 presents Annotated Textual Descriptions and discusses some of its theoretical applications in the field of Business Process Management. Chapter 4 presents the technical details behind the implementation of *ATDlib*. Chapter 5 describes the *Model Judge* platform, a practical use case of ATD and offers an empirical evaluation of the platform with two modeling courses. Finally, Chapter 6 concludes this thesis and discusses some possible future research lines.

Chapter 2

Background

In this chapter, we analyze the state of the art of all the relevant fields involved in this work. Business Process Management, covered in Section 2.1 is where the main contribution of this work lies in, since we aim to define a new business process modeling technique. Natural Language Processing, covered in Section 2.2 is used extensively in our work in order to automatically extract the relevant information out of unstructured business process description. Finally, Section 2.3 presents *NLP4BPM*, a project developed in our research group which is very closely related to the work presented.

2.1 Business Process Management

In large companies and organizations coordination between multiple actors involved in *business processes* is very important for correct operations. Business process management is a field of research focused in how to *communicate*, *document*, *analyze* and *enhance* business processes.

Business Process Management approaches processes at three different levels [10], illustrated in Figure 2.1:

Multi Process Management focuses on the identification of major processes of an organization and their prioritization. This task involves the inspection of the data repositories of a company, such as *Data Warehouses*, in order to discover and extract what are the relevant business processes.

Process Model Management is concerned with the management of a single Business Process Model. This involves *discovering* the process and creating a *process model* using some kind of process model notation, as well as implementing the necessary *monitoring* tools in order to *control* the process.

Process Instance Management deals with the actual execution of process: *Planning* how the tasks are going to be executed, *monitoring* the process during its execution and *adapting* the process if problems are detected.

BPM is a very wide area of research, with several sub-fields focusing on different aspects of business processes. From very theoretical research based on Petri net theory [11], to empirical psychological research about the *process of process modeling* [12].

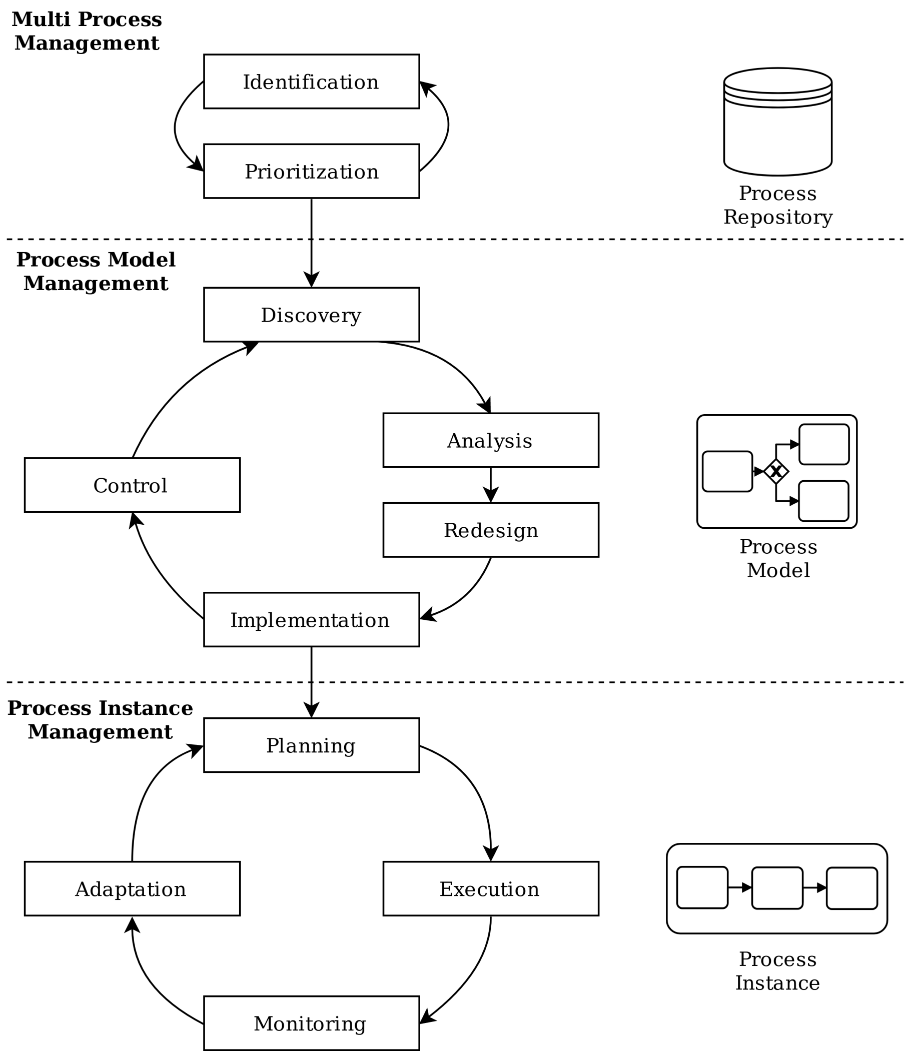


Figure 2.1: The three abstraction levels of business process management. Adapted from the figure in [10].

The project described in this Master Thesis is encompassed in an emerging field [13] in BPM, focused in the relation between human natural language and the formal methods found in other fields of research, such as *Process Mining* [14].

2.1.1 Business Process Notations

This Master Thesis presents an alternative to conventional Business Process Notations (BPN). In the literature, one can find numerous approaches to BP modelling techinques. In their survey [5], Ruopeng Lu and Shazia Sadiq, classify Business Process Notations in graph-based and rule-based notations.

Graph-based BPN represent business processes as a control flow graph. They are heavily inspired, and usually convertible to Petri Nets [15], a formal mathematical model usually employed to describe distributed systems. In graph-based BPN, the process is described as a sequence of linked nodes, indicating sequential, alternative or concurrent execution paths. To this day, the most prominent graph-based business process notation is the Business Process Model and Notation (BPMN) [2], with its first version published in 2011. However, before BPMN's quick rise in popularity, the Business Process Execution Language (BPEL) had the most widespread usage. Several other alternatives like YAWL [16] have been proposed both from the academics and the industry. Figure 2.2 shows a comparison between the three aforementioned graph-based business process notations. Despite their differences, there is a minimum subset of elements that almost all of them have in common. *(i)* The notion of activity, which is a task to be executed in the process, usually containing a text label in order to provide semantics to the process. *(ii)* The *XOR* split, which branches the execution in two alternative paths, and only one of them may be executed for any process instance. *(iii)* The *AND* split, which branches the execution into two concurrent paths, the execution of which can be interleaved arbitrarily.

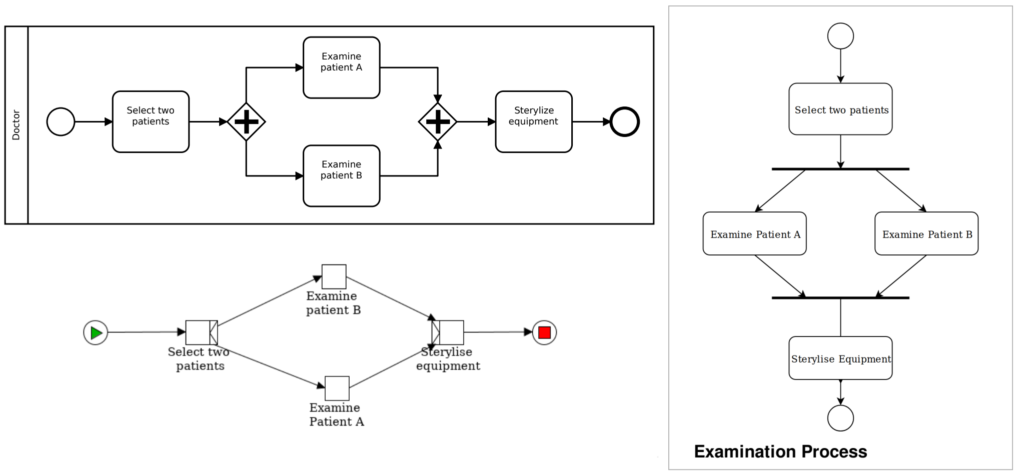


Figure 2.2: Example process in different graph-based execution languages: BPMN (top-left) YAWL (bottom-left) BPEL (right).

Rule-based BPN express the process model as a set of rules. In most rule-based systems, the rules are expressed as restrictions (e.g. "A request form can only be sent by a manager"). The main difference of these systems with respect to Graph-based alternatives is that they allow more flexible behaviour to be described in a more natural way, at the cost of higher verbosity and being more error prone. For instance, consider a variation of the process described in figure 2.2 where the order of the examinations is irrelevant, but it is mandatory for the doctor to sterilise the medical equipment between the two. In graph-based systems, duplication of activities cannot be avoided to express the semantics of this rule. On the other hand, most rule-based systems can introduce the rule that task *sterilise* must always happen between two examinations.

2.2 Natural Language Processing

Natural Language Processing (NLP) is a field of artificial intelligence which addresses the interactions between computers and human languages. NLP focuses on many problems at different levels of abstraction, from the purely linguistic ones like morphological and syntactic analysis or determining the part-of-speech of words to very challenging tasks such as the automatic summarizing of news articles.

Those tasks that go beyond parsing at a syntactic level and focus on the *semantics* of written text are usually classified under the field of *Natural Language Understanding* (NLU). The focus in NLU is to build complete semantic representations of texts in a machine-friendly format. It is thus an AI-complete [17, Section 1] problem, since text are addressed to human readers under the assumption of common sense and world knowledge: Things very difficult to encode in a computer program.

NLP techniques are applied to address a variety of use cases in the context of Business Process Management. Several of these focus on the text contained in process models themselves. This includes a variety of works that focus on the quality of process model labels, for example by restricting the use of certain labeling conventions [18, 19, 20], or common modeling errors [21]. Other approaches use NLP to augment process models with semantic or ontological information [22, 23, 24].

Other use cases involve texts that exist outside of process models. Several approaches extract process models from different kinds of text, such as from use cases [25], group stories [26], or methodological descriptions [27], while others take general textual process descriptions as input [28, 29], that is also the case of the *Text2BPMN* algorithm in *NLP4BPM*. However, these approaches have been found to produce inaccurate results, which require manual inspection [30]. Other use cases involving texts include a technique that considers *work instructions* when querying process repositories [31] for conformance checking against textual process descriptions [32].

2.2.1 Steps of Language Analysis

The NLP processing software used in this work is FreeLing[1] [7], an open–source library of language analyzers providing a variety of analysis modules for a wide range of languages. More specifically, the natural language processing layers used in this work are:

Tokenization & sentence splitting: Given a text, split the basic lexical terms (word, punctuation signs, numbers, ZIP codes, URLs, e-mail, etc.), and group these tokens into sentences.

Morphological analysis: For each word in the text, find out its possible parts-of-speech (PoS).

PoS-Tagging: Determine which is the right PoS for each word in a sentence. (e.g. the word *dance* is a verb in *I dance all Saturdays* but it is a noun in *I enjoyed our dance together.*)

Named Entity Recognition: Detect named entities in the text, which may be formed by one or more tokens, and classify them as *person*, *location*, *organization*, *time-expression*, *numeric-expression*, *currency-expression*, etc.

Word sense disambiguation: Determine the sense of each word in a text (e.g. the word *crane* may refer to an animal or to a weight-lifting machine). We use WordNet [**fellbaum98**] as the sense catalogue and synset codes as concept identifiers.

Constituency/dependency parsing: Given a sentence, get its syntactic structure as a constituency/dependency parse tree.

[1] `http://nlp.cs.upc.edu/freeling`

Semantic role labeling (SRL): Given a sentence identify its predicates and the main actors in each of them, regardless of the surface structure of the sentence (active/passive, main/subordinate, etc. For example, in the sentence *John does not want to go*, a SRL would detect two predicates (*want* and *go*), and mark that *John* is `Agent` of both. Note that a parser could detect that *John* is the `subject` of *want* but it would not detect that he is also the one supposed to *go*. SRL provides a higher level of abstraction than a parser, providing slightly deeper semantic knowledge. E.g. in a passive sentence such as *the fish was eaten by the cat*, a SRL system would detect an event *eat* with *cat* as `Agent` and *fish* as `Patient`, i.e. exactly the same that it would extract from the equivalent active sentence *the cat eats fish.*

Coreference resolution: Given a document, group mentions referring to the same entity (e.g. a person can be mentioned in the text as *Mr. Peterson*, *the director*, or *he.*)

Semantic graph generation: All the information extracted by the previous analyzers can be organized in a graph depicting events (mainly coming from predicates in the text), entities (coming from detected coreference groups), and relations between them (i.e. which entities participate in which events and with which role). This graph can be converted to triples and stored in an RDF database if needed.

2.2.2 Text Annotations

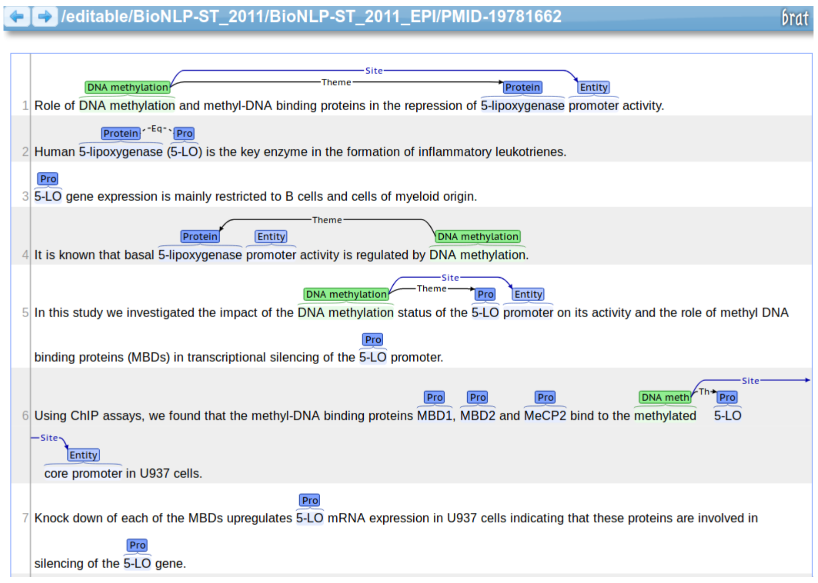
Creating annotated versions of texts is usual in NLP field, where many approaches are based on machine learning. Thus, to train a PoS-tagger, text where each word has been annotated with its part-of-speech, is required. Similarly, parsers, semantic role labelers, or coreference resolution systems, need example texts where this linguistic levels have been annotated by humans. These annotated corpora are then used to train the NLP analyzers and to evaluate their performance.

Thanks to this need for annotated data in NLP, convenient tools have been developed to ease the annotation task. One of them is *Brat* [8][2], a configurable environment that allows the annotation of texts with labelled spans, and relations. Figure 2.3 shows Brat being used to annotate a scientific text with very specific concepts from the field of biology.

A more recent alternative for text annotation are Text Annotation Graphs (TAG) [9]. TAG is a web-based software which conceptually improves upon Brat in two ways: *(i)* It provides the functionality of representing complex relationships between text elements, allowing the definition of relationships between relationships themselves (semantic hypergraphs). *(ii)* The visualization tool offers a more ergonomic and flexible representation of the annotated texts, allowing the repositioning of words, or filtering the view to only show a subset of the annotated information. Figure 2.4 shows a text annotated using TAG.

In this work, we present Annotated Textual Descriptions as a business process modeling language. ATD are based on text annotations, but its main purpose differs from the typical use cases of annotations. Ideally, ATD would be automatically performed by an NLP tool, instead of being used as training data. However, given the limitations of the current NLP state of the art, we will resort to a certain amount of human annotation

[2]`http://brat.nlplab.org`


/editable/BioNLP-ST_2011/BioNLP-ST_2011_EPI/PMID-19781662

1 Role of DNA methylation and methyl-DNA binding proteins in the repression of 5-lipoxygenase promoter activity.

2 Human 5-lipoxygenase (5-LO) is the key enzyme in the formation of inflammatory leukotrienes.

3 5-LO gene expression is mainly restricted to B cells and cells of myeloid origin.

4 It is known that basal 5-lipoxygenase promoter activity is regulated by DNA methylation.

5 In this study we investigated the impact of the DNA methylation status of the 5-LO promoter on its activity and the role of methyl DNA binding proteins (MBDs) in transcriptional silencing of the 5-LO promoter.

6 Using ChIP assays, we found that the methyl-DNA binding proteins MBD1, MBD2 and MeCP2 bind to the methylated 5-LO core promoter in U937 cells.

7 Knock down of each of the MBDs upregulates 5-LO mRNA expression in U937 cells indicating that these proteins are involved in silencing of the 5-LO gene.

Figure 2.3: One of the example annotations provided with the Brat annotation tool.

to improve the quality of our semantic representation. This is discussed in further detail in Chapter 3.

2.3 The *NLP4BPM* Project

When multiple business process representations are used inside an organization, it becomes very important to ensure availability and consistency of the existing documentation. The *Natural Language Tools for Business Process Management* (*NLP4BPM*) project, developed by our group in the Polytechnic University of Catalonia, aims to solve several related problems in this domain.

In *NLP4BPM*, several different approaches are taken in order to aid organizations in managing multiple representations of processes:

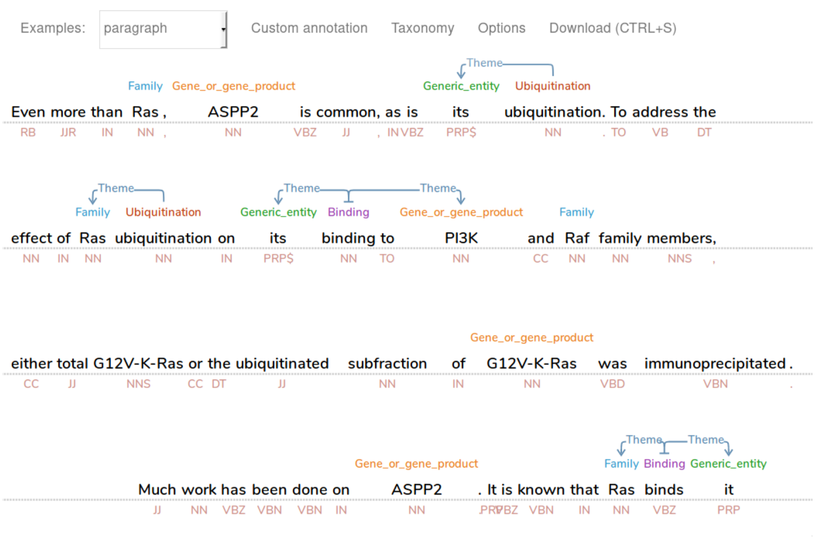
Text2BPMN Converting a natural language textual description into a BPMN business process model.

BPMN2Text Converting a BPMN business process model into a human-readable textual description.

BPMNvsText Computing an *alignment* between a model and a textual description in order to compare them. An alignment is a mapping between tasks in the model and sentences in the text. The implementation follows the technique presented in [33].

BPMNvsBPMN Computes an alignment between two different BPMN models. Here the alignment is between the set of tasks of the two process models. The techniques implemented in this module are based on Relaxation Labeling algorithms

BPMN2ChatBot Converts a BPMN diagram into an interactive chat bot that guides a user throughout the process execution.



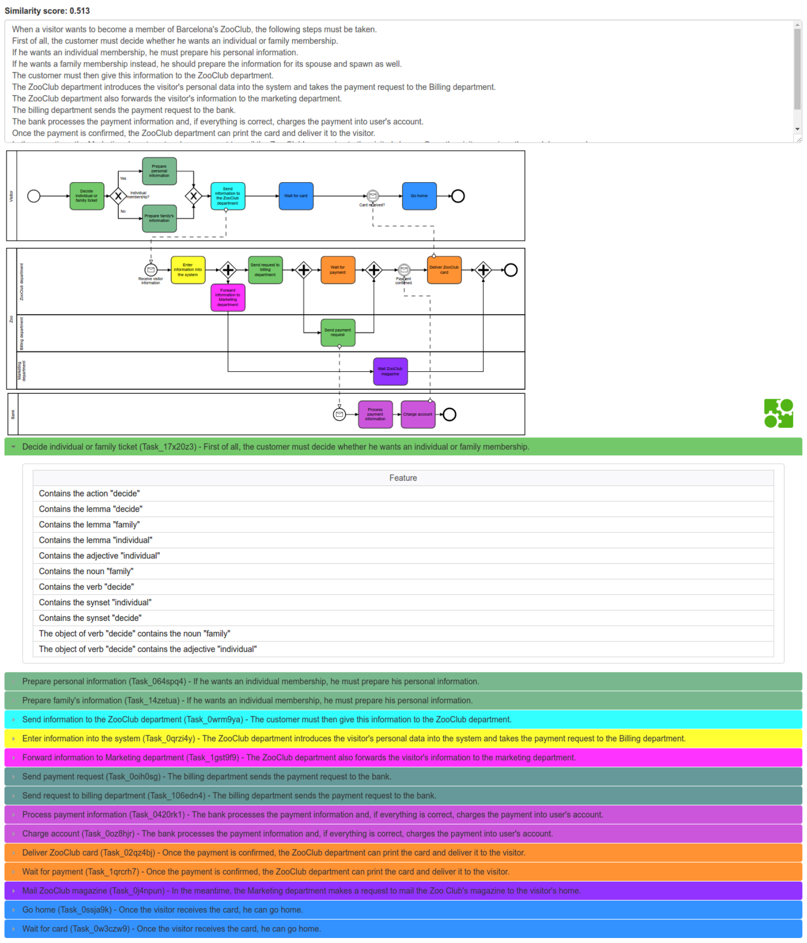
Even more than Ras, ASPP2 is common, as is its ubiquitination. To address the effect of Ras ubiquitination on its binding to PI3K and Raf family members, either total G12V-K-Ras or the ubiquitinated subfraction of G12V-K-Ras was immunoprecipitated. Much work has been done on ASPP2. It is known that Ras binds it

Figure 2.4: One of the example annotations provided with the TAG annotation tool.

The different techniques are developed as independent modules and unified under the same graphical interface, which is presented in a user-friendly manner. The tool is available online at `http://nlp4bpm.cs.upc.edu/`. Figure 2.5 shows a screenshot of the web user interface.

One of the common elements in the *NLP4BPM* project is the use of *FreeLing*, and particularly of *FreeLing*'s *semantic graphs* in order to extract semantic information from both texts and BPMN labels. A semantic graph is an abstract representation of a document where entities (e.g. people, objects) are represented as nodes, and joined by relations, such as the *temporal adjunct*, which joins an action with the words that describes the particular time during which it is performed, or the *manner adjunct*, which indicates the way in which the action is performed instead.

One of the goals of this Master's thesis is to link Annotated Textual Descriptions, the main contribution presented in this work, with FreeLing semantic graphs. This will allow using ATD in order to improve the robustness of the algorithms presented above without having to modify them. Section 4.1.1 describes how this is achieved.


Visitor
Decide individual or family ticket
Individual membership?
Yes
No
Prepare personal information
Prepare family's information
Send information to the ZooClub department
Wait for card
Card received?
Go home
Zoo
ZooClub department
Receive visitor information
Enter information into the system
Forward information to Marketing department
Send request to billing department
Wait for payment
Payment confirmed
Deliver ZooClub card
Billing department
Send payment request
Marketing department
Mail ZooClub magazine
Bank
Process payment information
Charge account

Similarity score: 0.513

When a visitor wants to become a member of Barcelona's ZooClub, the following steps must be taken.
First of all, the customer must decide whether he wants an individual or family membership.
If he wants an individual membership, he must prepare his personal information.
If he wants a family membership instead, he should prepare the information for its spouse and spawn as well.
The customer must then give this information to the ZooClub department.
The ZooClub department introduces the visitor's personal data into the system and takes the payment request to the Billing department.
The ZooClub department also forwards the visitor's information to the marketing department.
The billing department sends the payment request to the bank.
The bank processes the payment information and, if everything is correct, charges the payment into user's account.
Once the payment is confirmed, the ZooClub department can print the card and deliver it to the visitor.

Decide individual or family ticket (Task_17x20z3) - First of all, the customer must decide whether he wants an individual or family membership.

Feature
Contains the action "decide"
Contains the lemma "decide"
Contains the lemma "family"
Contains the lemma "individual"
Contains the adjective "individual"
Contains the noun "family"
Contains the verb "decide"
Contains the synset "individual"
Contains the synset "decide"
The object of verb "decide" contains the noun "family"
The object of verb "decide" contains the adjective "individual"

Prepare personal information (Task_064spq4) - If he wants an individual membership, he must prepare his personal information.

Prepare family's information (Task_14zetua) - If he wants an individual membership, he must prepare his personal information.

Send information to the ZooClub department (Task_0wrm9ya) - The customer must then give this information to the ZooClub department.

Enter information into the system (Task_0qrzi4y) - The ZooClub department introduces the visitor's personal data into the system and takes the payment request to the Billing department.

Forward information to Marketing department (Task_1gst9f9) - The ZooClub department also forwards the visitor's information to the marketing department.

Send payment request (Task_0oih0sg) - The billing department sends the payment request to the bank.

Send request to billing department (Task_106edn4) - The billing department sends the payment request to the bank.

Process payment information (Task_0420rk1) - The bank processes the payment information and, if everything is correct, charges the payment into user's account.

Charge account (Task_0oz8hjr) - The bank processes the payment information and, if everything is correct, charges the payment into user's account.

Deliver ZooClub card (Task_02qz4bj) - Once the payment is confirmed, the ZooClub department can print the card and deliver it to the visitor.

Wait for payment (Task_1qrcrh7) - Once the payment is confirmed, the ZooClub department can print the card and deliver it to the visitor.

Mail ZooClub magazine (Task_0j4npun) - In the meantime, the Marketing department makes a request to mail the Zoo Club's magazine to the visitor's home.

Go home (Task_0ssja9k) - Once the visitor receives the card, he can go home.

Wait for card (Task_0w3czw9) - Once the visitor receives the card, he can go home.

Figure 2.5: Example execution of the BPMNvsText module at the NLP4BPM web interface.

Chapter 3

Describing Processes with Natural Language

In this chapter, we describe the use case of Annotated Textual Descriptions (ATD) in natural langauge to describe business processes. Section 3.1 presents a process model example that will be used throughout this document for illustrative purposes. Next, Section 3.2 contains a rationale for ATD and Section 3.3 describes the semantics of the base ATD language. Finally, Section 3.4 describes possible theoretical applications of ATD.

3.1 Running Example

In order to provide an illustrative example for the remainder of this document, let us consider a process of patient examination[1]. The textual description of this process is provided in Figure 3.1. Additionally, figure 3.2 shows the equivalent process model in a well-known process notation: BPMN.

3.2 Annotated Textual Descriptions

In order to support the efficient creation and communication of business processes, we must establish notation that is close to the mental framework of all the actors involved in the process. Natural language is an ideal tool for this task, being the most widely used form of human communication.

However, using natural language alone has several drawbacks. Natural Language Processing tools are getting increasingly better at extracting structure out of plain text. However, some ambiguities remain an open issue to this day. Furthermore, we argue that some of these issues cannot be automatically solved reliably enough, since they require extensive domain knowledge. For instance, in the example from Figure 3.1, the sentence *"The latter [physician] is then responsible for taking a sample to be analysed in the lab later"* highlights two important sources of ambiguity in the analysis of textual

[1]Based on the description provided by the Women's Hospital of Ulm. Both the text and the model are extracted from [34].

The examination process can be summarised as follows. The process starts when the female patient is examined by an outpatient physician, who decides whether she is healthy or needs to undertake an additional examination. In the former case, the physician fills out the examination form and the patient can leave. In the latter case, an examination and follow-up treatment order is placed by the physician, who additionally fills out a request form. Beyond information about the patient, the request form includes details about the examination requested and refers to a suitable lab. Furthermore, the outpatient physician informs the patient about potential risks. If the patient signs an informed consent and agrees to continue with the procedure, a delegate of the physician arranges an appointment of the patient with one of the wards. The latter is then responsible for taking a sample to be analysed in the lab later. Before the appointment, the required examination and sampling is prepared by a nurse of the ward based on the information provided by the outpatient section. Then, a ward physician takes the sample requested. He further sends it to the lab indicated in the request form and conducts the follow-up treatment of the patient. After receiving the sample, a physician of the lab validates its state and decides whether the sample can be used for analysis or whether it is contaminated and a new sample is required. After the analysis is performed by a medical technical assistant of the lab, a lab physician validates the results. Finally, a physician from the outpatient department makes the diagnosis and prescribes the therapy for the patient.

Figure 3.1: Textual description of a patient examination process.

descriptions: *(i)* The action *analyse* is described, but it may not happen until later into the process, as hinted by the word *later*. Extracting this kind of temporal relationships from textual descriptions is a very complex task [35], with no good solutions in the context of business processes. *(ii)* Depending on the purpose of the process, the same *analyze* verb could actually be irrelevant from an organizational point of view, if the *lab* were to be considered an external entity. This presents an issue that cannot be solved without deep understanding of the organization. In order to solve this, while still preserving the idea of a flexible and natural way of describing processes, we rely on text annotations (See Section 2.2.2) as a common langauge to be used by humans and computer systems.

We have named this new documentation format Annotated Textual Descriptions (ATD). An ATD can be used as a replacement for any business process modeling notation: A human reader can easily get a grasp of the process by simply reading the text without the annotations, and can use the annotations to clarify any ambiguities encountered (e.g. whether two activities can be executed concurrently). On the other hand, a computer system can rely on the annotations to automatically extract structure and additional semantic information from the process, without the need to leverage a generic semantic graph [7]. The design goals of ATD can be summarized as follows:

Can Model Ambiguity Ambiguity is an inherent property of some processes. In some cases, an organization cannot reliably document every aspect of a business process. An example of this pehomena are error scenarios. Consider the sentence "If everything is correct, the bank proceeds to charge the payment into the user's account". This sentence is implicitly saying that, under some circumstances the

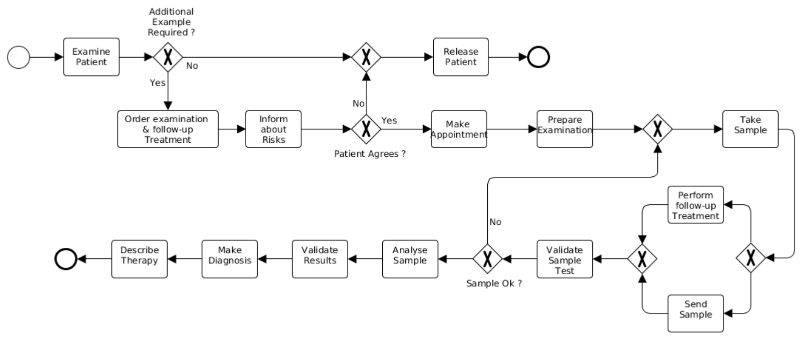


Figure 3.2: Process model for the patient examination process.

execution will end in an undesired state if the bank cannot proceed with the payment. However, the phrasing clearly indicates that such error scenarios are outside the scope of the process. In cases like this, other techniques force the modeler to disambiguate the unclear parts of the process, resulting in too restrictive models that do not fully represent the reality of the organization. On the other hand, ATD are designed to capture the ambiguity by being built on an *open world assumption*: If a part of the process is not clearly documented, anything that happens is assumed to be correct. This characterization of ambiguity also allows to automatically detect and report the ambiguous elements to the user in order to improve the process.

Rich Semantics Since natural language is used as the main building block of ATD, any idea can be communicated with them. At the human level, the expressive power of the language is almost on par with the mental model of the writer. At the annotation level, ATD are designed to easily incorporate extensions into the annotations, so they can be adapted to multiple use cases. An example of that is seen in Chapter 5, where the base annotation language is expanded to enable automatic evaluation of students.

Low Representational Bias Representational bias is the phenomena where the notation being used forces the author of a process to model things in a certain way [36]. For example, notations based on Petri Nets are well suited to model concurrency. However, there is no clear built-in mechanism to express the limits of iterative structures. This is why languages like BPMN have a built-in semantic element to model concurrency, the *parallel gateway*, while loop conditions have to be expressed as text notes which cannot be interpreted by computer systems. ATD, being close to the mental model of the actors involved in the process, have a low representational bias.

Automation To ensure adoption of a new language, ease of use is a very important factor to consider. Models must be easy to build and redundant constructs –i.e. *boilerplate*– must be kept at a minimum. In ATD, the provided tooling ensures that such redundant tasks are automated: On one hand, an NLP tool generates an initial annotation for the user to refine, which typically just consists of removing unnecessary annotations. On the other hand, intuitive rules have been encoded in an inference engine to avoid unnecessary annotating. For example, the *Coreference Relation* (See Section 3.3.2) is transitive and symmetric by default, saving redundant annotations. Similarly, the *Sequential Relation* is added by default for two consecutive activities in the text if no other relation is specified.

3.3 ATD Semantics

ATD are designed to be adaptable. In this section, we describe the base language which aims to represent the common concepts in all business processes. Additionally, extensions are allowed in the form of new *annotation* or *relation* specializations, to support more specific use cases. An example of such expansion can be seen in Chapter 5.

An ATD is defined as a set of annotations on top of a base text, written in natural language. More formally, we can define an ATD as a $\langle T, A, R\rangle$ tuple, where

T is a string of characters representing the textual description.

A is a set of *annotations*. Each annotation $\alpha = \langle type, start, end\rangle$ marks a relevant aspect of the business process. The *start* and *end* integers mark the positions of a substring of T, and the *type* is used to add semantics to the annotation. The types of annotations are defined as a hierarchy, so an annotation types can be specialized into subtypes.

R is a set of triples $\langle t, \alpha_i, \alpha_j\rangle$ representing binary relations of type t between pairs of annotations α_i, α_j. Similarly, the *type* of a relation is used to add particular semantics to it, and specialization of relation types is allowed.

Next, we describe each of the annotation and relation types that we consider for the base language of ATD. Table 3.1 shows an example of an ATD which we will use for the remainder of this section.

	Type	Start, End
α_1	*action*	"an examination and follow-up treatment order is **placed** by the physician"
α_2	*action*	"is then responsible for taking a sample to be **analysed** in the lab"
α_3	*action*	"After receiving the sample, a physician of the lab **validates** its state"
α_4	*role*	"an examination and follow-up treatment order is placed by the **physician**"
α_5	*role*	"the female patient is examined by an **outpatient physician**"
α_6	*object*	"an **examination and follow-up treatment order** is placed by the physician"
α_7	*condition*	"decides whether **she is healthy** or needs to undertake additional examination"
α_8	*optional, ending*	"the patient can **leave**"
α_9	*implicit*	"The patient is **asked** to sign an informed consent" (Added text)
α_{10}	*action*	"The required examination and sampling is **prepared** by a nurse"
α_{11}	*action*	"The required examination and sampling is **prepared** by a nurse"
α_{12}	*object*	"The required **examination** and sampling is prepared by a nurse"
α_{13}	*object*	"The required examination and **sampling** is prepared by a nurse"
α_{14}	*role*	"After receiving the sample, a **physician** of the lab validates its state"

	Type	α_i	α_j
r_1	*agent*	α_1	α_4
r_2	*patient*	α_1	α_6
r_3	*sequential*	α_2	α_3
r_4	*coreference*	α_4	α_5
r_5	*patient*	α_{10}	α_{12}
r_6	*patient*	α_{11}	α_{13}
r_7	*fusionable*	α_{10}	α_{11}

Table 3.1: Example fragment of Annotated Textual Description with annotations (left) and relations (right).

3.3.1 Annotation Types

Annotations are used to mark an important region of the text. In the base language of ATD, we consider the following types of annotations which directly map to some well-known concepts in the context of business processes.

Action. The *action* annotation is used to represent the steps of the business process model, which are often associated with verbs. An example action would be *placing*

(α_1) an examination order. Each action corresponds to a single activity in the process model.

Entity. An anything that participates in the business process, either actively or passively. This is a generic type that is specialized in two subtypes: *role* and *business object.*

Role. A type of *Entity.* The *role* annotation is used to represent the autonomous actors involved in the process, like the *outpatient physician* (α_5). Any mention of an entity that performs some Action is considered a Role.

Business Object. The business object annotation is used to mark all the relevant elements of the process that do not take an active part in it, such as an *examination order* (α_6). In other notations, correspondences with business objects are typically found in activity labels and explicit elements such as data object references

Condition. A condition annotation indicates a part of the text that represents a prerequisite for some part of the process being executed, such as the patient *being healthy* (α_7). Conditions are typically associated with XOR-splits, their labels and their surroundings.

3.3.2 Relation Types

Agent. A *role* is the *agent* of an *action* whenever the entity represented by the role performs that action. For instance, relation r_1 tells us it is the *physician* who *places* something.

Patient. An *entity*, is the *patient* of an *action* whenever it acts as the recipient of the action. Relation r_2 tells us that what is *placed* is an *examination order.*

Control Flow. The *sequential*, *exclusive* and *parallel* binary relationships, which are borrowed from behavioral profiles [37], are used to indicate the order between *action*s in the textual description. Due to the characteristics of natural language text, there is an open world assumption on the set of control flow relationships: Assuming an absence of contradictions, everything that is stated as relationship is enforced. However, no assumptions are made on things that are not specified. In our example, *analyse* (α_2) is *sequential* (r_3) –i.e. happens before– to *validates* (α_3)

Coreferences. An *entity* is a coreference of another *entity* when they refer to the same real world entity. The coreference relation forms a graph with one connected component per process entity. All occurrences of the ”patient” role in the example text are coreferences. However, there are two different ”physician” roles in the text, the ”outpatient physician” (α_4, α_5) and the ”physician of the lab” (α_{14}), which form two disconnected coreference graphs.

Fusionable Actions. Some *actions* can be expressed at multiple granularity levels depending on the context. When two actions are fusionable, it means they can be represented as a single one without changing the semantics of the process. In the sentence *”the required examination and sampling is prepared”*, we could assume two independent actions *prepare (examination)* (α_{10}) and *prepare (sampling)* (α_{11}) or a single one *prepare (examination and sampling)*, as indicated by r_7. In our particular use-case, this kind of relation can help the instructor annotate those cases in which different granularity levels should be equally accepted as an answer.

3.3.3 Automatic Reasoning

One of the design goals of ATD is to avoid unnecessary human work by automating the annotation of texts as much as possible. In order to achieve this goal, automatic reasoning is built on top of the annotations in order to infer new knowledge:

Coreference Graphs The Coreference relation specifies groups of entities that represent the same concept. To avoid having to manually annotate all the pairwise coreference relations for each coreference group, the *coreference* relation is defined as both symmetric and transitive. This expanded relations are automatically inferred, so the user only needs to mark $n - 1$ annotations to form a coreference group of n entities.

Behavioral Profiles A behavioral profile [37] is an abstract representation of the control flow of a process. The behavioral profile of a process is a function which, for every pair of elements in the process –i.e. places in a Petri net or activities in a BPMN diagram–, yields a temporal relation, which can be one of *sequential* ($\rightsquigarrow$), *exclusive* ($+$) or *parallel* ($\|$). The notion of behavioral profile can also be adapted to ATD by using the homologous relationships, but in order to compute the full profile, the following rules must be applied to the initial set of relations: *(i)* The *exclusive* and *parallel* relations are symmetric *(ii)* The *sequential* relation is transitive. *(iii)* The *sequential* relation is automatically assumed for consecutive actions unless some other action is stated. Rule *iii* can be stated more formally as:

$$\begin{gathered}(\nexists r \in \mathbb{R} \ s.t. \ \alpha_i = src(r) \land \alpha_j = dest(r)) \ \land \\ (\nexists \alpha_k \ s.t. \ start(\alpha_k) > end(\alpha_i) \land end(\alpha_k) < start(\alpha_j)) \\ \Longrightarrow \\ \mathbf{R} \leftarrow \mathbf{R} \cup \{\langle sequential, \alpha_i, \alpha_j \rangle\}\end{gathered}$$

This last rule helps avoid repetition and unnecessary annotations, but also helps enforce the property that processes should be described in a sequential fashion, which is a desirable characteristic.

Process Actors By considering the *role* annotation and the *agent* and *coreference* relations, a reasoner can automatically infer all the roles involved in a process. This can be reported to the user to quickly verify that all the relevant actors are represented in the process.

3.4 Possible Applications of ATD

ATD have been designed as a form of process documentation. However, due to the unique way in which they capture process information in natural language, we argue that they have several potential applications beyond other process model notations:

As an Intermediate Language: Being close to natural language, the preferred method of communication for non-experts in the BPM field, ATD can be used as an intermediate format for process representation. Thanks to the robustness of curated annotations, algorithms can be developed that reliably transform ATD into other process model notations that better adapt to the different situations (e.g. because some software tool requires that specific format). Automatically performing transformations from natural language to other notations is an unreliable process, which requires careful revision of the obtained results. Using ATD moves the revision process to the modeler side, where all the issues in the natural language description are disambiguated at annotation time. This implies a text can be annotated once and transformed to as many different representations as required.

For the Simulation of Textual Descriptions: Process *simulation* consists of generating an *event log* that conforms to a Business Process Model[38] assuming a certain statistical model. Thanks to the robustness introduced by ATD over raw text, simulation tools can be built on top of ATD, which would bring all the advantages of simulation (i.e. detection of issues, such as *bottlenecks*, before execution) to descriptions in natural language, easily understandable by all implied roles.

As Training Data: ATD are based on text annotations, originally conceived to serve as training data for machine learning algorithms in NLP. By creating ATD, an organization doesn't only get the benefits of such notation, but additionally can contribute to generate training data for the automatic analysis of Business Process Descriptions. This, in turn, can lead to improved algorithms specific to the BPM use case, which would not need as much human intervention in the generation of new ATD.

Chapter 4

ATDlib: A Library to Manage and Generate ATD

This chapter presents *ATDlib*, a reference implementation of the ATD language described in Chapter 3 which implements parsing, automatic generation and transformations of ATD. ATDlib is built using *Clojure* [39], a dynamic language for the Java Virtual Machine and thus can be used from any language in the JVM ecosystem. Section 4.1 focuses on the several tools provided in ATDlib and offers implementation details. Then, Section 4.2 shows an overview of the process of creating an ATD from raw text data.

4.1 Features and Architecture

The main goal behind ATDlib is to provide the necessary tooling in order to support the implementation of ATD in organizations as a way of describing business processes. This major goal can be split into the following functionalities:

Parsing and Representation of ATD ATD are stored as BRAT annotation files in *standoff format* [40]. The standoff format is a text-based file that represents *annotations* and *relations* which reference a plain text file. Therefore, an ATD will consist of a `Process.txt` with the textual description and a complementary `Process.ann` file with the annotations. ATDlib can parse this format and convert it to its internal schema, exposed in a public interface as shown in the UML diagram of Figure 4.2

Automatic Annotation ATDlib can perform an initial annotation of a raw text in natural language. The goal is to perform the most accurate annotations possible. However, due to limitations in NLP, the annotations generated need to be currently reviewed by a human. All annotation and relation types described in Section 3.3 are generated except the *sequential*, *exclusive* and *parallel* relations, which must be manually annotated.

Easy Manual Annotation One of the goals of ATDlib is to facilitate the manual annotation of ATD in order to refine an automatically derived annotation. Instead of implementing a custom annotation tool, we use well-established tools from the NLP field in order to provide an annotation front-end. For this project, we considered BRAT [8] and TAG [9]. Despite TAG having a more rich feature set, ultimately we chose BRAT due to its more mature implementation and documentation availability.

Automatic Translation Several transformations are implemented in ATDlib in order to convert ATD to other process representations that interface with existing software. The goal is to establish ATD as a unique language for which any other representation can be automatically derived. These transformations can then be implemented as additional modules in ATDlib. Currently, the implemented transformations are to convert ATD into a FreeLing [7] *semantic graphs* and a *behavioral profiles* [37]

ATDlib is structured as a single library with several entry points, corresponding to its different modules. Figure 4.1 shows an overview of the different entry points, and their main functionalities. The `brat_parser` module is intended to be used as a library, and can be used to read an annotation file into a more friendly format. The `text2atd` entry point can be used as a standalone tool or as a library, and converts textual descriptions into partially annotated ATD, which should then be checked by a human using any compatible annotation tool, such as BRAT. Finally, the `atd2fl` and `atd2bp` modules can be used as libraries to extract a FreeLing semantic graph and a behavioral profile respectively. The inference rules described in Section 3.3.3 are applied during the transformations. As mentioned in Section 2.3, the semantic graph and the behavioral profiles can then be applied directly as input data to the algorithms in the *NLP4BPM* project.

4.1.1 Converting ATD to Freeling Semantic Graphs

ATD and Freeling semantic graphs hold very similar information. Given the graph-based nature of both representations, this conversion can be thought of as a *schema mapping* problem. However, some aspects need to be taken into consideration to perform the conversion in a way that improves the quality of the algorithms in the *NLP4BPM* project, which is one the main application of this library.

The conversion performed by the `atd2fl` module consists of translating the concepts in ATD to a semantic graph, which contains the following information:

- A list of *entities*, which are nodes in the graph that correspond to the groups of connected components of *entities* in the ATD when considering the *coreference* relation.
- A list of *frames*, nodes that correspond to *action* annotations in the ATD
- Relations between entities and frames. Currently supported are *A0* (agent) and *A1* (patient). However, some ATD expansions could map new relations such as the *AM-MNR* (manner adjunct) or *AM-LOC* (location adjunct).

But in order to have a complete textual representation for other tools to use, the following information must also be extracted from an ATD:

- Lists of paragraphs, sentences and tokens
- For each token
  - The word's form and lemma.

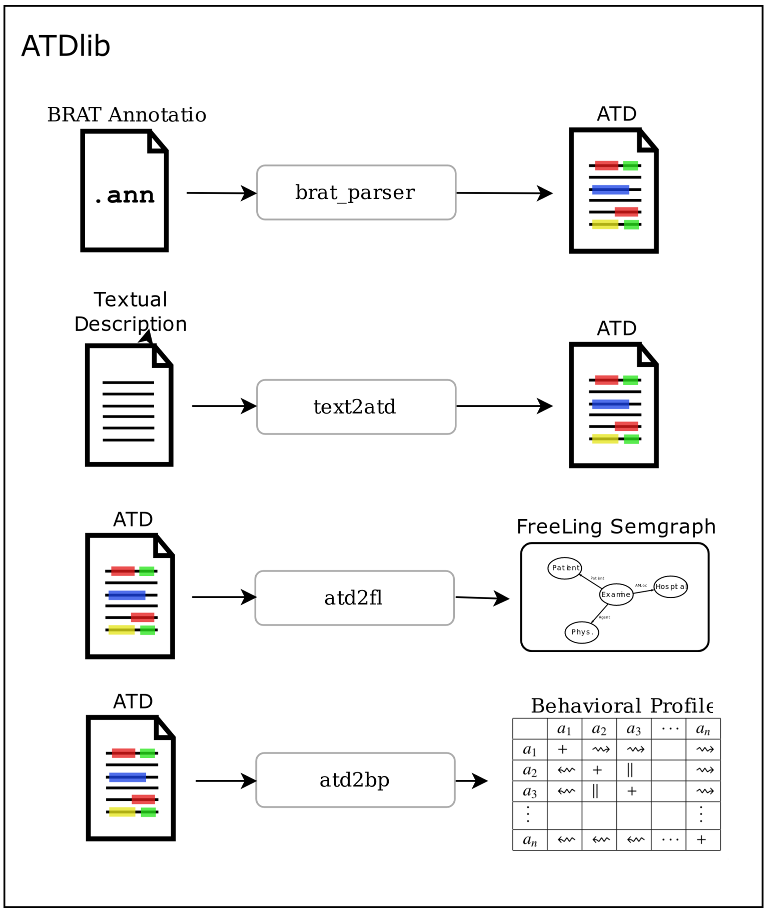


Figure 4.1: ATDlib feature overview

– The part-of-speech (PoS) of the word.
– For nouns, verbs, adjectives and adverbs; a WordNet reference to the sense of the word.

This information is necessary because FreeLing's semantic graph is expressed by referencing certain words in the text, but including this information in the ATD would require extensive unnecessary annotation, which doesn't align well with the ATD's rationale. That is why `atd2fl` performs a low-level analysis on the text (up to the sense disambiguation stage) using FreeLing to automatically extract that information. This low-level information is then merged with the ATD, in order to build a full semantic graph. During this conversion, only the tokens that are part of some annotation in the ATD are kept in the final structure, and all other words are removed.

4.2 Creating an ATD with *ATDlib*

Figure 4.4 illustrates the steps that must be taken in order to generate a full ATD with ATDlib. First, a human actor must describe the business process using natural language, creating a `MyProcess.txt` file. Next, the `text2atd` module is used in order

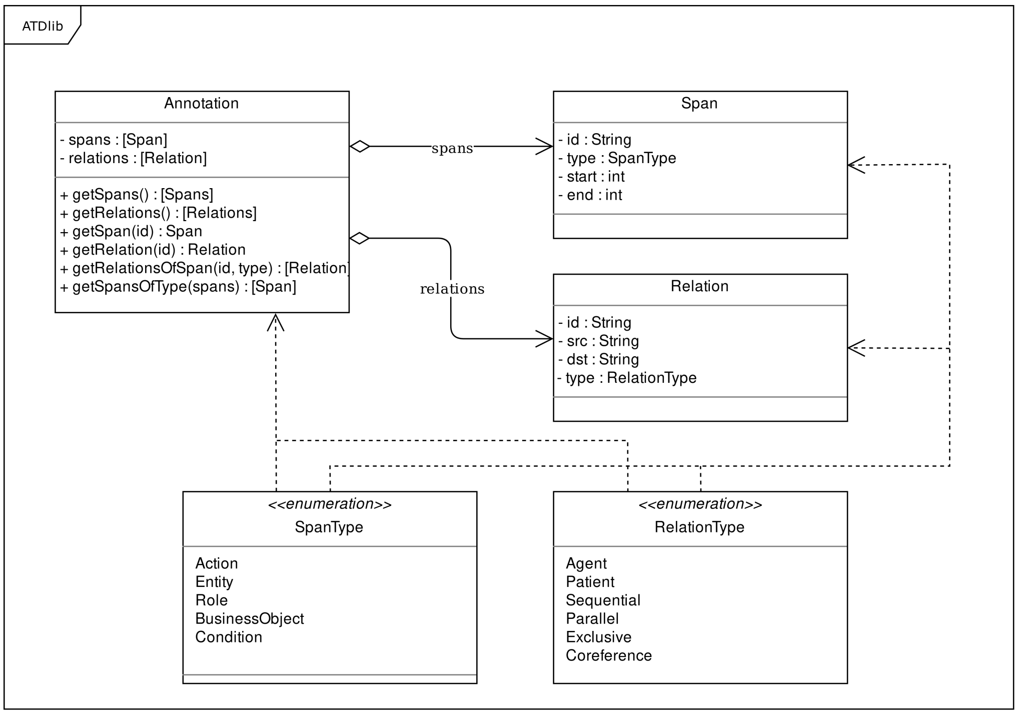
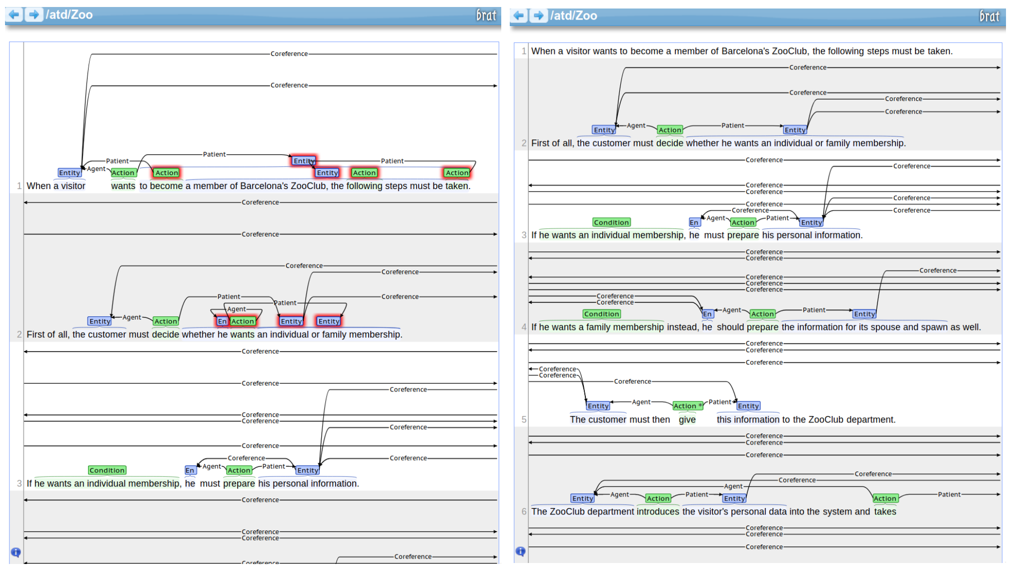


Figure 4.2: UML diagram showing the public interface used for ATD representation

to obtain an annotation file, `MyProcess.ann`. The user can then inspect the generated annotation using the BRAT web interface. Figure 4.3 (left) shows an example of what the user will see in the screen at this point. The user must then refine the annotation, usually this process consists of deleting the irrelevant actions detected by ATDlib, as well as marking the control flow for the final set of actions, resulting in an annotation file similar to the one shown in Figure 4.3 (right). Finally, if necessary, ATDlib can be used to generate other process model representations, such as a FreeLing semantic graph using the `atd2fl` module.



1 When a visitor wants to become a member of Barcelona's ZooClub, the following steps must be taken.
2 First of all, the customer must decide whether he wants an individual or family membership.
3 If he wants an individual membership, he must prepare his personal information.

1 When a visitor wants to become a member of Barcelona's ZooClub, the following steps must be taken.
2 First of all, the customer must decide whether he wants an individual or family membership.
3 If he wants an individual membership, he must prepare his personal information.
4 If he wants a family membership instead, he should prepare the information for its spouse and spawn as well.
5 The customer must then give this information to the ZooClub department.
6 The ZooClub department introduces the visitor's personal data into the system and takes

Figure 4.3: The brat user interface: (left) An automatically generated ATD. (right) After the user performs a manual validation.

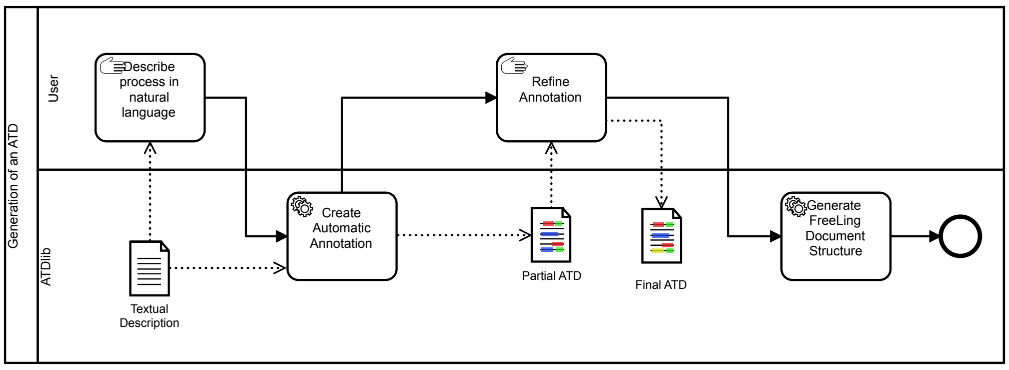

Generation of an ATD
User
ATDlib
Describe process in natural language
Refine Annotation
Create Automatic Annotation
Generate FreeLing Document Structure
Textual Description
Partial ATD
Final ATD

Figure 4.4: Workflow of the creation of an ATD using ATDlib

Chapter 5

Model Judge: A Practical Use Case of ATD

Due to the wide usage of process models in organizations, correctness and quality of models directly influences the correct execution of processes. However, research has shown that industrial process models often contain errors, which can lead to increased costs in production.

Automating the detection of syntactic errors is a common feature in modelling software. However, the error types more closely related to the natural language sections of the model are usually not checked, due to the difficulties in the automatic analysis of such elements. Part of this Master thesis' work consisted in exploring a practical application of Annotated Textual Descriptions by designing the main algorithm behind *Model Judge*. Model Judge is a web platform supporting students in the creation of business process models by automatically detecting and reporting the most common sources of semantic and pragmatic errors in modeling The motivation behind the platform and its functionalities are discussed in Sections 5.1 and 5.2.

The algorithm, which is described in detail in Section 5.3 is based on the technique for automatic computation of alignments between process model and textual descriptions presented in [33]. Section 5.4 explains how the alignment technique in [33] has been adapted to use ATD instead of plain text.

The Model Judge platform has been used as the main modeling tool as part of two Computer Science courses: One in Denmark's Technical University (DTU) and another in the Catholic University of Santa María (UCSM). Section 5.5 describes the experimental setting and analyzes the results obtained from the modeling sessions.

5.1 The *Model Judge* Platform

Model Judge is a web platform designed to provide diagnostics regarding issues on *syntactic*, *pragmatic* and *semantic* quality in business process diagrams. Students are presented with a statement describing a business process in natural language, and are asked to model the corresponding diagram.

The framework behind Model Judge is based on computing an alignment between the process model diagram to be evaluated and a reference textual description, which is an ATD based on the exercise's statement. This ATD is generated only once, with intervention from the course instructors, and can be used to support multiple students simultaneously.

5.1.1 Diagnostics

Model Judge is based on computing quality diagnostics on business process models. Instead of providing the user with a difficult to interpret numerical score, we believe more detailed feedback helps students and contributes to continuous self-improvement.

By observing the grading process of several modeling courses, we have established a set of diagnostics that are suitable for being computed automatically. We have split these diagnostics in three different categories: *Syntactic* diagnostics consider the structure of the model. *Pragmatic* diagnostics verify the phrasing of the process model labels and can be tuned to enforce certain grammatical rules. Finally, *Semantic* diagnostics check if there is no missing information from the underlying process and all the information provided is relevant. Next, the concrete diagnostics for the three aforementioned types are detailed.

Syntactic Diagnostics

A good process model should have a clear and unambiguous control flow. Syntactic diagnostics identify common patterns that typically result in less understandable and maintainable process models.

Gateway Reuse and Implicit Gateways. Gateway reuse refers to a gateway that acts both as a *split* (more than two outputs) and a *join* (more than two inputs). Implicit Gateways exist when an activity has multiple input or output flows. The semantics for these two constructs are not clear and can lead to hidden modeling errors. Because of that, avoiding gateway reuse and implicit gateways is a well-known best practice in business process models [41].

Non-Natural Loops. Due to their similarity, some desirable properties of program's control flow graphs are also relevant in the context of process model diagrams. Ideally, process models should contain only of *Natural Loops.* That is, there is only a single way to enter the loop. We have observed non-natural loops are a common pattern among novice students.

Soundness. A well-known desirable property of process models is *soundness* [42], which guarantee the process model is free from livelocks, deadlocks, and other anomalies that can be detected without domain knowledge.

Pragmatic Diagnostics

Process model diagrams define a large portion of their semantics using natural language. It is desirable to restrict the language to a strict writing style (e.g, the *verb-object* rule in [41], G6), in order to avoid ambiguous phrasing [43]. A simple and strict style is also important when considering automatic analysis of the process model language. For example, while it is acceptable in the text to include the sentence: *"The latter is then responsible for taking a sample to be analysed in the lab later"*, having that sentence as an activity label in the process model adds unnecessary complexity, since the aim is to have label text to be as simple as possible.

Previous studies [44, 43, 19] have established the common structures in label descriptions. However, these structures are not always followed by novice students. Because of that, an automatic detection of invalid writing styles is beneficial for student self-improvement.

Semantic Diagnostics

A process model diagram has to communicate the semantics of the underlying process in a clear and unambiguous way. All the information provided has to be correct, and in the right order. On the other hand, unnecessary information introduces noise that can generate confusion. Semantic diagnostics help enforcing these properties on the process model.

Missing/Unnecessary activities. This problem arises when a relevant activity of the process is omitted from the process model, or symmetrically when additional activities are added which are either wrong or add no relevant information. This can be caused by an oversight or a poor understanding of the process being modeled.

Missing/Unnecessary Roles. When process models use role information, such as swimlanes in BPMN, the same diagnostics of *missing* and *unnecessary* roles can be applied as well, to ensure all the relevant actors are properly modeled.

Control-Flow Consistency. All the information in the process model should be consistent with the control flow of the process being described. If some temporal relationship described in the text is not accurately incorporated in the process model, then a control-flow consistency violation would be communicated to the novice modeler.

5.2 A Tour through *Model Judge*

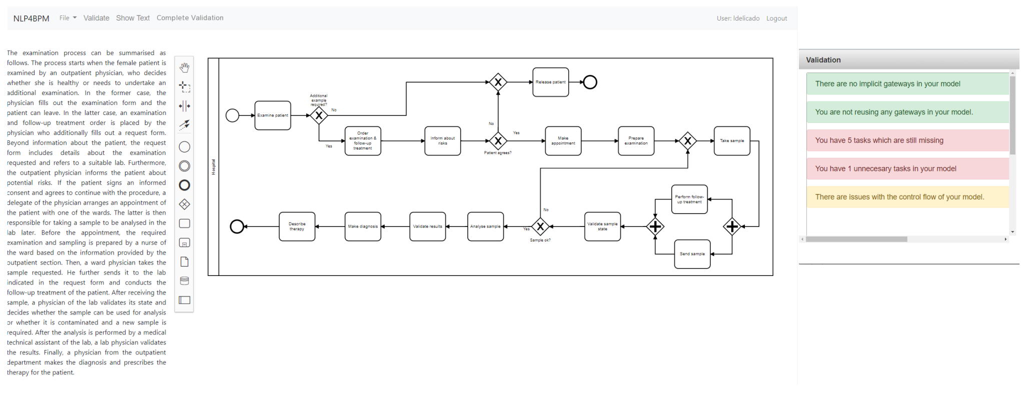
The Model Judge can be accessed using any modern web browser at `http://modeljudge.cs.upc.edu`. It is designed both for helping students in the process of creating a process model, and instructors in the task of designing modeling activities in an agile way.

The application requires registration, and users with the *instructor* role are able to create and manage courses. Additionally, support is planned to allow instructors to create and manage their exercises for the application, in the form of ATD, which is currently restricted to the developers of the application.

All registered users are able to access the exercise list and attempt solving any of them. After selecting an exercise, the user is brought to the modeling view, which is displayed in Figure 5.1. In the modeling view, the user has access to a textual description of the process, and a BPMN editor.

During the exercise, students can use the **Validation** functionality to obtain aggregated diagnostics about the process model, for example, how many unnecessary activities it contains. A more detailed version is the **Complete Validation**, which offers more fine-grained information about the diagnostics, such as the exact labels of the unnecessary activities in the process model.

Finally, for users that allow it during registration, analytics are enabled. The analytics functionality which is built into the platform periodically records snapshots of the student's process model for the duration of the validation session. Additional snapshots are created each time a user performs a *validation* or *complete validation*. Section 5.5 describes the analysis performed on this analytics data.



The examination process can be summarised as follows. The process starts when the female patient is examined by an outpatient physician, who decides whether she is healthy or needs to undertake an additional examination. In the former case, the physician fills out the examination form and the patient can leave. In the latter case, an examination and follow-up treatment order is placed by the physician who additionally fills out a request form. Beyond information about the patient, the request form includes details about the examination requested and refers to a suitable lab. Furthermore, the outpatient physician informs the patient about potential risks. If the patient signs an informed consent and agrees to continue with the procedure, a delegate of the physician arranges an appointment of the patient with one of the wards. The latter is then responsible for taking a sample to be analysed in the lab later. Before the appointment, the required examination and sampling is prepared by a nurse of the ward based on the information provided by the outpatient section. Then, a ward physician takes the sample requested. He further sends it to the lab indicated in the request form and conducts the follow-up treatment of the patient. After receiving the sample, a physician of the lab validates its state and decides whether the sample can be used for analysis or whether it is contaminated and a new sample is required. After the analysis is performed by a medical technical assistant of the lab, a lab physician validates the results. Finally, a physician from the outpatient department makes the diagnosis and prescribes the therapy for the patient.

There are no implicit gateways in your model

You are not reusing any gateways in your model.

You have 5 tasks which are still missing

You have 1 unnecesary tasks in your model

There are issues with the control flow of your model.

Figure 5.1: Model Judge modeling view

5.3 Approach

At the Model Judge's core, there is the algorithm that produces a list of diagnostics for their current business process model. The design and implementation of this algorithm has been implemented as a part of this Master Thesis.

In Figure 5.2, we can see a BPMN diagram of the diagnostic generation process. The inputs to the system are the *Textual Description*, which corresponds to the exercise statement, and the *Process Model* created by the student. First, the textual description is analyzed by the Text Annotation module (See `text2atd`, Section 4.1) to automatically produce an Annotated Textual Description, which is then completed with human intervention to include the domain expertise not available to the NLP algorithms. This step is only performed once and can be used to evaluate any number of students in that exercise. Once a particular student wants to validate a partial model, the process model is sent to Model Judge. First, the process model is analyzed: A *behavioral profile* is computed, and *semantic roles* are extracted from the text labels in the model using a specialised chart parser. After the model is analyzed, the *alignment* between the process model elements, such as activities or gateways and the annotations in the ATD is established. Finally, using all the previous information, the process model is checked against a textual description by the *diagnostics* module.

The next sections describe the multiple steps in Model Judge's algorithm in more detail. Section 5.3.1 covers the analysis of the process model. Next, Section 5.3.2 covers the ATD extensions necessary to support Model Judge. In Section 5.3.3, the necessary steps for obtaining the diagnostics are explained. Finally, the details behind the alignment of ATD and process models are discussed in Section 5.4.

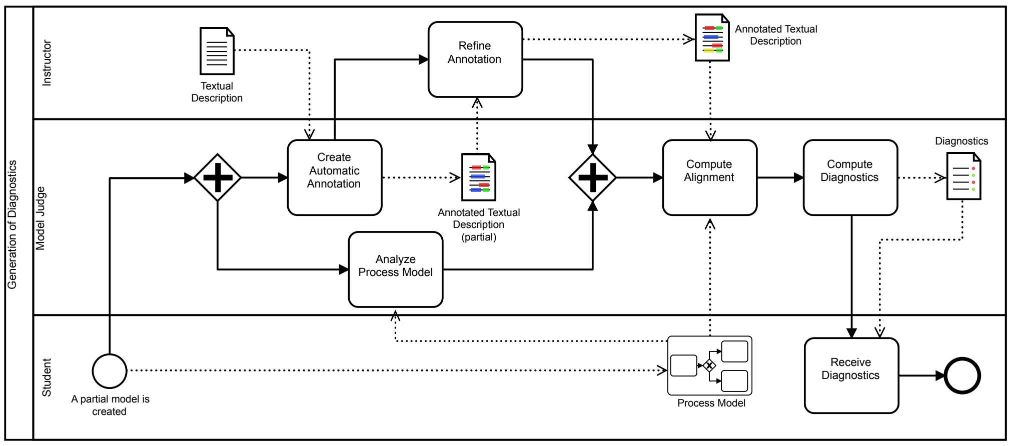


Figure 5.2: *Model Judge* process overview

5.3.1 Process Model Analysis

Model Judge performs two separate analysis of a Process Model sent by a student. First, a *behavioral profile* of the process is computed, in order to compare it to the one from the ATD during the alignment computation (See Section 5.4.3). The computation of a *behavioral profile* has well known algorithms and can be performed as long as the process' Petri net is *bounded* [45, Section IV.B]. Analyzing a process model, thus, consists of three steps:

1. Convert the BPMN process into a Petri Net, this sometimes results in unbounded Petri nets.
2. Check the boundedness of the resulting Petri net, only if the property is fulfilled, the next step is executed.
3. Compute the behavioral profile of the Petri net.

The necessary operations to compute the behavioral profile are implemented in JBPT [46], a Java library consisting of several utilities to manage business process models.

The second step in Model Judge's analysis of process models is extracting semantic roles from its text labels from activities, gateways, events, swimlanes and pools. Using FreeLing's generic parser and SRL module for this analysis, however, has some drawbacks: NLP parsers are generally trained on large corpora of written text, such as books or newspaper articles. However, this training data doesn't represent well the kind of sentences found in a process model, which are typically incomplete and use excessive capitalization[1]. A very common confusion of NLP tools when parsing text labels in models written in the english language occurs in the case of noun-verb *homographs*[2]. Consider, for example, the phrase "mail confirmation". A typical NLP parser would identify this as a noun phrase, the *confirmation of the mail*; this is because the *prior* probability of the "mail" word occurring as a noun is far higher in the typical training

[1]Excessive capitalization happens when all the words in a sentence are capitalized in order to highlight its importance. This is considered good writing style in section headers or process model labels, but most NLP parsers are not trained to recognise the phenomena and confuse some words for proper nouns when it is used.

[2]Two words are *homographs* when they share the same spelling

data used. However, in business process models, labels usually start with a verb. When consider this information, we see that the real meaning of the label is that *someone mails a confirmation.*

To deal with this, we built a custom SRL module which encodes the most common label styles classified in [19] in order to build a chart parser specific to process model labels. Due to the simplicity of most label structures, the parse tree can be directly used to extract the semantic roles using a custom set of rules. In our current implementation, we consider the most common types of labels:

Verb-Object This label style consists of a verb in imperative form (the main *action*), followed by a noun, representing the *business object* and additional complements. For example, "Examine patient" is a label in Verb-Object style.

Action-Noun In this label style, the main action is represented as a noun, and is usually preceded by the business object. An example of this label style would be "Patient examination".

Double-Action This is a compound structure, where two or more actions are described in a single label, using Verb-Object style. "Examine Patient and Request Sample" could be an example label using that style.

The grammar for the custom chart parser tries to identify any of the above structures, and is written in such a way that that the root node of the parse tree will be labeled with the identified writing style, if found. For example, a label written in the *noun-action* style will have this information reflected at the root node of the tree. Using this grammar, the parsing is then done as follows:

1. The n most likely PoS-tag sequences of the label are computed.
2. For each PoS tagging, the label is parsed using a chart parser with the aforementioned context-free grammar.
3. If some parse tree's root is tagged with one of the aforementioned writing styles, the grammar the *action* and *patient* (i.e. business object) roles will be extracted by following the identified structure of the label. Note that due to BPMN's semantics, the *agent* of the action is never described in the activity label, as that is the purpose of swimlanes.

5.3.2 ATD Exensions for Model Judge

In order to support the automatic evaluation of students, several extensions are included to the base language described in Chapter 3. This extensions are in the form of specializations of the Action annotation type, which help better represent the semantics of the process for this particular use-case. For each annotation, multiple specializations are allowed. The descriptions in this section follow the running example from Figure 3.1.

Optional. An *optional action* annotation is used to indicate that the associated action could be elided from the process description without a substantial change in its semantics. For instance, in the sentence *"the patient can leave"*, the action of *leaving* could be considered as part of the process. However, that action does not add a substantial amount of semantic value, and thus can be considered *optional.*

In Model Judge, optional actions can be used by the instructor to allow for a more flexible answer to the exercise, since both the solution that contains the action and the one that does not will be considered correct.

Ending. An *ending action* annotation is used to represent an action that ends the process. A process may have multiple ending actions to represent multiple success or failure scenarios. In the previous sentence, *"the patient can leave"*, *leave* is also an *ending action*, as well as *prescribe* in *"Finally, a physician (...) prescribes the therapy"*. Ending actions in BPMN must correspond, or immediately follow an end event. This information can be used by Model Judge to derive its diagnostics.

Implicit. Sometimes, action descriptions are not present as words in the text but can be inferred. In Model Judge, this means that students are supposed to introduce a task for which a description is not explicit in the process. Since this sort of implicit description is not suited for automatic analysis, our approach consists of adding additional text that makes the inferred assumption explicit. This text, however, is only visible to the instructor. An *implicit action* annotation is used to highlight this fact. In our example text, we can read *"If the patient signs an informed consent and agrees to continue (...)"*. However, no explicit mention is made as to what will happen in case the patient refused to sign the informed consent.

5.3.3 Diagnostics Generation

The generation of the diagnostics is the final step in Model Judge's algorithm. The goal is to generate human-friendly diagnostics by identifying the differences between the ATD and the business process model. Most diagnostics can be computed a two granularity levels. Either at a general level –e.g. The process is missing *some* activities–, or at a concrete level –e.g. The process is missing a label about the "Patient Examination"–. The general level is used when performing an *validation*, while the concrete diagnostics are only emitted for *complete validations*. The computation of the diagnostics uses the information in the ATD, the analysed process model and the alignment between the two. In this section we detail which diagnostics are currently generated by Model Judge and how they are computed.

Syntactic Diagnostics

The computation of syntactic diagnostics only requires the process model. Most checks implemented use well-known algorithms. Particularly, checking the use of implicit gateways or gateway reuse consists of a very simple structural check. On the other hand, the presence of non-natural loops can be determined by checking for cycles in the process graph after removing all its *back edges*, computed from the *dominator tree* [47].

Pragmatic Diagnostics

The approach we use for checking adherence to a writing style is based on using the custom parser for labels described in Section 5.3.1. Particularly, we use the identification of the label writing style in the root of the label's parse tree in order to restrict the labels to a certain set of accepted styles, which can be configured.

In the current implementation, the writing style check focuses on looking for double actions, that is, activity labels that consist of two actions, such as *"Close Ticket and*

Inform the Manager", for which the students are encouraged to write two separate activities. However, this technique can be adapted so any organization can enforce a label writing style in a flexible way. This adaptation would consist of expanding the grammar to handle the newly accepted new label styles, if necessary, and adding those styles to a list in Model Judge's configuration files.

Semantic Diagnostics

Semantic checks are concerned with the underlying process semantics. Computing these diagnostics requires all the information from the annotated textual description, the process model and the alignment.

To detect that an *activity is missing* from the process model, the alignment information is used. Particularly, if there is an action in the ATD with no correspondences from the process model, the activity is considered missing from the process model. The system can then inform the modeler by generating a detailed error message using the annotated data from the textual description.

Detecting *unnecessary activities* also relies on the alignment. In this case, there will be a process model activity p aligned to some ATD action a. If the similarity between p and a is low enough, according to the result of computing the predictor (as described later in Section 5.4.4), the activity is considered unnecessary, as there is no good match for the activity in the text.

The *coverage of roles* is computed similarly to activities. In this case, the similarity function is used to assess the similarity between *role* annotations and the process model's swimlanes. An unrestricted alignment is performed and the matches are used to detect missing and unnecessary roles as previously described with activities.

Finally, in order to detect *control flow consistency*, we focus on the differences between the constrained and unconstrained alignment. Particularly, if the objective function –that is, the sum of correspondence similarities– between the two alignments differs greatly, a warning is displayed to the user informing that the control flow in their process may be wrong. This careful phrasing is intended, since our testing has shown this method to not be completely reliable to detect subtle control flow errors, both in terms of false positives and false negatives.

5.4 Aligning ATD and BPMN Process Models

Establishing the correspondence between the annotations in the textual description and the elements in the business process model is necessary in order to compute some diagnostics, such as the semantic-related ones. The goal of this computation is to be able to assess which parts of the process model have been correctly modeled by the student, according to the textual description.

In order to achieve this goal, the technique presented in [33] is used, which computes an alignment from process model activities to a textual description's sentences. The most important adaptation for Model Judge is the alignment input, which now considers ATD's *annotationss* instead of the plain sentences. In order to achieve this goal, the `atd2fl` and `atd2bp` modules from *ATDlib* are used to accurately transform the input text to a *FreeLing*'s semantic graph and *behavioral profile*, which are the original expected inputs of the alignment algorithm.

Figure 5.3 shows an overview of the alignment approach presented in [33]. Next, we cover each of the steps in the algorithm, and the necessary adaptations that were performed to use the algorithm as part of Model Judge:

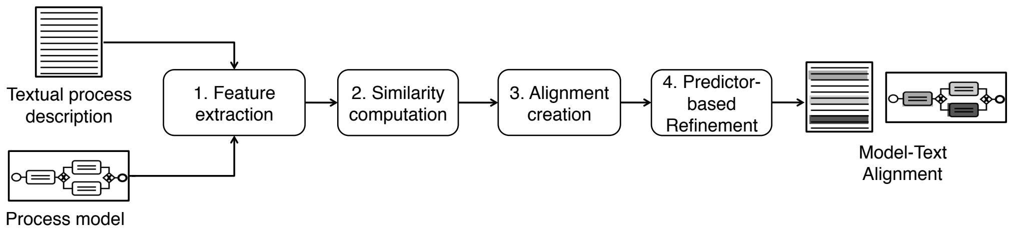

Textual process description
Process model
1. Feature extraction
2. Similarity computation
3. Alignment creation
4. Predictor-based Refinement
Model-Text Alignment

Figure 5.3: Overview of the alignment technique presented in [33]

5.4.1 Feature Extraction

In order to establish a distance metric for two essentially different sources, such as an ATD and a BPMN Process Model, the elements in the two representations are converted into a canonical representation, the *Feature Vector*. Feature vectors are extracted for each *Action* or *Condition* in the ATD, and each *process element* (i.e. *activity*, *gateway* or *event*) in the business process model

A feature vector $v \in \mathbb{R}^\infty$ is a sparse real vector computed in an infinite dimensional space. Each of the dimensions of the space correspond to a linguistic feature of the element. For instance, the first non-zero dimension of some vector v_e may represent that the agent of the element e is associated with a certain word. The strength of this association –i.e. the *weight* of the feature– is indicated by the magnitude of the vector in that dimension.

We will refer to each individual dimension as a *feature instance*. Additionally, we group conceptually similar feature instances in *feature families*. Each feature family corresponds to all the parametrizations of a certain linguistic feature:

$contains\_action(a)$ This feature family denotes the actions contained in a step p or sentence s. For instance, when comparing a process step "*Examine Patient*" to the annotated sentence "*The process starts when the female patient is examined by an outpatient physician*", we extract the action "*examine*". This implies the extracted feature vectors from the ATD and process model would share a non-zero value for the *feature instance* $contains\_action(examine)$

$contains\_role\_word(w)$ & $role\_main\_word(w)$ This feature family denotes the roles that execute steps in a process. The former feature, *contains_role_word*, is extracted for each word w that is part of a role. For instance, the "*outpatient physician*" actor comprises the two words "*outpatient*" and "*physician*". By contrast, we use the *role_main_word* feature to denote the main word of the role, typically represented as the main noun, e.g., to explicitly capture the word "*physician*". This separation allows us to give a different *weight* to the main word with respect to the others.

$contains\_object\_word(w)$ & $object\_main\_word(w)$ These features encode the same information for *business objects* as the previously described features do for *roles*.

contains_lemma(l, pos) This feature is extracted from the target text if it contains a word[3] with the lemma l and part-of-speech *pos*. In case of ATD, this would correspond to all the words contained within the *Action* and its related *Agent* and *Patient Entity* annotations, if any. This feature has a lower abstraction level than the previous ones and is included as a fall-back solution whenever the actor, role or action cannot be determined due to natural language ambiguity in the process model. In those cases the algorithm works at the word level. For example, for the annotated sentence "A ward physician takes the sample requested", the features *contains_lemma(physician, noun)*, *contains_lemma(take, verb) contains_lemma(sample, noun)* and *contains_lemma(requested, adjective)* will be extracted.

contains_synset(s) This feature is extracted whenever the WordNet synset s appears in the text sentence. It captures the semantics of words to help identify similarity when synonyms are used. For example, the WordNet synset *10020890-n* recognizes that "*physician*" is a synonym of "*doctor*", such as used in the running example.

contains_hypernym(s) This feature is extracted from a target text containing a word for which s is an hypernym[4] at distance *HL* or less. *HL* is a parameter of the algorithm. In the running example, a hypernym of "*physician*" is "*medical practitioner*" (*10305802-n*)

By using ATD, which are accurately annotated, this feature extraction phase is improved. Particularly, the high level features from the text can be reliably extracted for all *Action*s, since they are available through the *Agent* and *Patient* relations. This means the algorithm doesn't need to rely as much on the word-based features.

5.4.2 Similarity Computation

Once the heterogeneous sources are converted into a canonical representation, a standard similarity metric can be used to compare them. As shown in [33], the metric that gives best results for our problem domain is the *Weighted Overlapping Index*, computed as:

$$WeightedOverlapping(a, b) = \frac{a \cdot b}{min\{\|a\|^1, \|b\|^1\}} \tag{5.1}$$

Where $\|v\|^1$ represents the L1 norm of the vectors.

5.4.3 Alignment Creation

The next step is the computation of the alignment between process model elements and the annotations of an *ATD*. The alignment contains the corresponding ATD action for each activity, gateway and event of the process model.

Formally, we define an alignment σ to be a set of correspondences of the form $n_k \sim a_i$ for some $n_k \in N$, $a_k \in A$, where N denotes the set of elements of a process model and A is the set of actions of the ATD.

[3]Note that in all feature types stopwords are not considered.

[4]A word w_1 is a *hypernym* of w_2 if w_1 describes a superclass of w_2 (e.g. *mammal* is a hypernym of *cat*, and *document* is a hypernym of *letter*). Hypernymy is obtained from WordNet.

In order to establish an alignment between process model elements and sentences, we impose two types of constraints on the alignment: *cardinality* constraints and *ordering* constraints.

Process step-to-sentence cardinality. For process model activities and events, we enforce that each of them is aligned to exactly one ATD action, whereas we allow multiple steps to be aligned to the ATD action. This, accounts for an activity being described once in the text, but being duplicated in the model, due to some requirement in the control flow.

Process step ordering. Assuming there are no contradictions between the control flow of the ATD and the process model, we can restrict the set of possible alignments for those that would violate the natural order of the process. This is one of the points where Model Judge's approach differs from the technique described in [33, Section 5.6], where the temporal relations in the text could not be computed. By using accurately annotated ATD, we are able to extract a full *behavioral profile* for the text. This allows us to set the restriction that, for each pair of correspondences, $a \sim s, a' \sim s'$, the temporal relation between a and a' is the same for sentences s and s'. More formally expressed as: $\forall (a \sim s, a' \sim s') \in \sigma^2 : bp_{pm}(a, a') = bp_{atd}(s, s')$, where bp_{pm} and bp_{atd} are the behavioral profiles of the process model and annotated textual description respectively.

However, this restriction can lower the quality of the alignment when there are consistency errors in the control flow of the process. That is why two alignments are computed in our approach. One with ordering restrictions and one where this set of restrictions is disabled.

The optimal alignment is then found, which is the one that maximizes the sum of the similarities of the alignments, while preserving the aforementioned constraints. This problem can be solved in linear time where the *ordering restrictions* are disabled, but becomes difficult when considering them. For the latter case, we modelled the problem as an ILP optimization, which was found to be fast enough for the use case of student evaluation[5].

5.4.4 Predictor-based Refinement

This is one of the key steps in order to compute the *unnecessary activity* diagnostics in Model Judge. The optimal alignment described in the previous section, is computed under the assumption that the process model and the ATD describe the same set of process activities. However, this is not the case when evaluating partial models performed by students. Particularly, the following inconsistencies may result in an alignment that is incorrect:

1. The student has modeled an activity which is not in the ATD, meaning it is not relevant to the process.
2. The student has not modeled an activity which is in the ATD, meaning it is missing
3. The order in which the activities are described in the model

[5]Using the Gurobi [48] ILP solver, an answer from the optimization can be found in less than a second, thus, the NLP analysis becomes the bottleneck of the process

Model Judge works by considering the alignment in order to generate its diagnostics corresponding to points *1* and *2*, as seen in Section 5.3.3. However, the alignment computed at this point is not sufficient to determine which unnecessary activities the student introduced. The similarity scores of the correspondences are highly valuable indicators in order to detect this information. Because of that, we introduce a refinement step after the alignment which consists in applying a threshold on the similarity function, to discard the correspondences with very low similarity from the process.

In the refinement step, instead of applying the threshold directly to the similarity function, we introduce the concept of *predictors*. A predictor is a function of the correspondences in the alignment which assigns a likelihood value of that correspondence being valid (i.e. not unnecessary). This value will be always in the interval $[0, 1]$, but does not represent a probability. In our approach, we defined the following predictors:

- $p\text{-}sim(n \sim a)$: the likelihood that a correspondence $n \sim a$ relates to an unnecessary activity, given as the similarity score between the step n and the ATD action a i.e. $sim(n, a)$.
- $p\text{-}rel\text{-}S(n \sim a)$: the value of the previous predictor, normalized by the maximum similarity between a and any other process step, i.e. $\frac{sim(n,a)}{max\ \{sim(n',a)\ \mid\ n' \in N\}}$
- $p\text{-}harm(n \sim a)$ The harmonic mean between $p\text{-}sim(n, s)$ and $p\text{-}rel\text{-}S(p, s)$

The choice of predictors is greatly influenced by the similarity function. In our approach, the similarity function uses the *Weighted Overlapping Index* and thus is contained in $[0, 1]$. However, in practice, we have observed that depending on how different is the level of verbosity in the text and model, the similarity functions will vary overall. This motivates the choice for a relative predictor, which won't be affected by the differences of the discourse in the ATD and process model. However, using a relative predictor alone like *p-rel-S* would fail in the event that all the activities introduced by the user are unnecessary. In order to have a predictor that is robust to such scenarios, we take the harmonic mean between *p-rel-S* and *p-sim*, using *p-harm* as the predictor for Model Judge

5.5 Experiments

Model Judge has been tested in two separate modeling courses. The first was performed on the Technical University of Denmark (DTU) during February 2018. as part of a graduate course on business process management. The second course was performed at the Catholic University of Santa María (UCSM) in Peru during March 2018 as part of a practical session in the Business Process Management course in the Computer Science degree.

In order to validate how the Model Judge helps novice modelers, we performed two evaluations. The first one consisted in analyzing the recorded data from the modeling sessions (Section 5.5.1). The second evaluation arises from a survey that was handed to the students at the end of the modeling course (Section 5.5.2). Below, we detail the results obtained in both evaluations.

5.5.1 Analysis of the Modeling Session Data

There were substantial differences in the setting of the two courses. On the DTU course, students were allowed to complete the exercise at home without a time limit. On the other hand, the UCSM course consisted in two exercises that had to be completed in a two-hour session. Additionally, some technical issues affecting DTU students were corrected before the UCSM course. Despite the differences, we believe the information recorded is still relevant for the analysis we present in this session.

For every student[6], we periodically stored (every minute) information for the whole modeling session. Additionally, the same information was also saved each time the user performed a simple or complete validation. In particular, we recorded a total of 8410 intermediate models for 72 students. For the snapshots, we stored: *(i)* A unique user identifier. *(ii)* The process model in BPMN (XML) format. *(iii)* The timestamp of the snapshot. *(iv)* The type of information: automatic, validation or complete validation. *(v)* The validation results of our tool for the particular process model. Note that the validation results were computed for all snapshots, despite the students only seeing the ones they explicitly requested.

After the end of modeling course, we analyzed the snapshots with the aim of observing the evolution of the number of validation errors during the modeling sessions. The dataset used to perform this analysis can be found in the following address: `http://www.cs.upc.edu/~pads-upc/ModelJudgeData.zip`.

Modeling Behaviours

By manual inspection, we identified several modeling profiles when analyzing the sessions data. Figure 5.4 shows a representative for each of the identified profiles, when plotting number of validation errors vs. time (in seconds)[7]. An instructor can have a good summary of the student evolution by looking at these student's plots: *(i)* The first group is composed of students that frequently use the validation and complete validation functions, and ended up with almost no bad diagnostics. This group corresponds to 19.0% of the students (30.4% in the DTU course and 12.5% in the UCSM course). *(ii)* The students from the second group frequently use the simple validation, however, only check the complete validation at the end of the session. The final amount of bad diagnostics for this group is comparable to the previous one. This group corresponds to 60.3% of the students (56.5% in the DTU course and 62.5% in the UCSM course). *(iii)* The students from the third group started working on the exercise but finished before fixing the majority of bad diagnostics. We have observed that the students in this group performed substantially less validations. This group corresponds to 20.6% of the students (13.0% in the DTU course and 15.8% in the UCSM course).

Evolution of Diagnostic Types

We analyzed how the frequency of different diagnostics varies during the session. Figure 5.5 shows the evolution of the average amount of diagnostics for all students, per diagnostic type. To better observe the relative behaviour of each type regardless of the amount of diagnostics in the category, we plot the values relative to the maximum of each category. We have encountered substantial differences between diagnostic types.

[6]We did not record information for those students that didn't opt-in to the analytics

[7]There exists few outlier profiles that do not match any of the three representatives shown in Fig. 5.4.

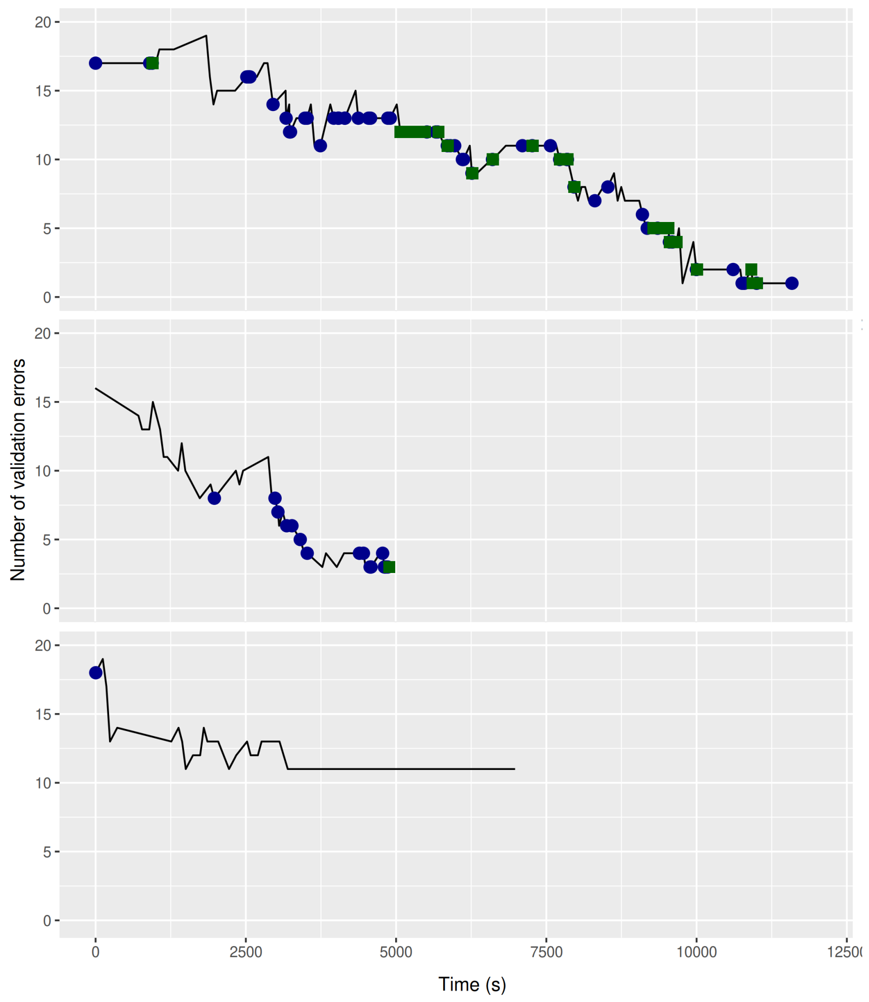


Figure 5.4: Three characteristic behaviors observed in the modeling sessions. The blue circles represent simple validations, while green squares denote complete validations.

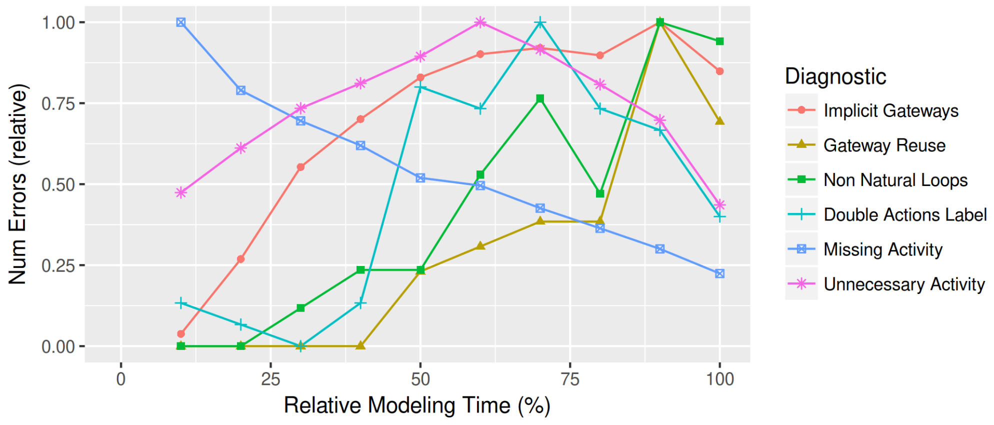


Figure 5.5: Evolution of the diagnostic types for the modeling session.

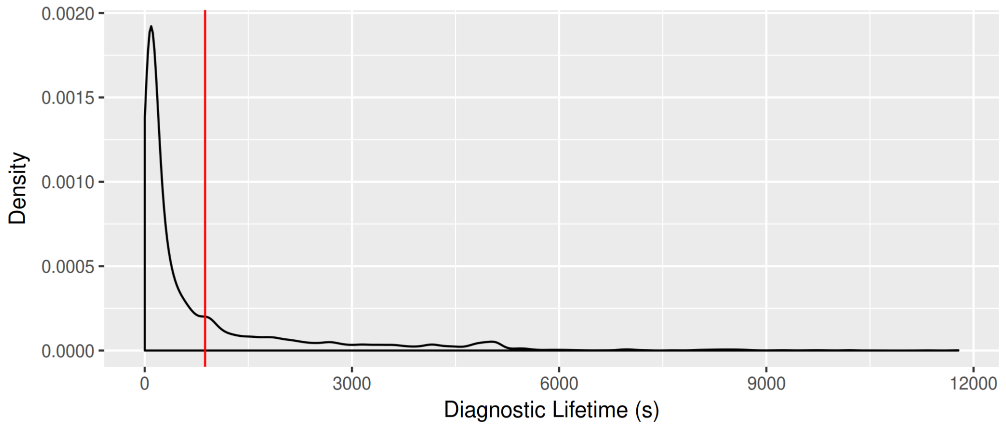


Figure 5.6: Density plot of the diagnostic lifetimes variable.

Missing Activity diagnostics decrease as the session advances, since less activities will be missing as the modeling session progresses.

Unnecessary Activity and Double Action Writing Style increase for the first half of the session, then decrease. This behavior is consistent with the fact that most students do not start using the complete validation feature until the second half of the session. The more detailed feedback of the complete validation then helps them finding the more subtle errors in their process model.

The remaining diagnostic types have an oscillatory behaviour, but still increase for the duration of the session. This can be explained by the fact that, as the model session progresses, there is a greater chance of a student introducing an error leading to one of these diagnostics. However, the drops after 75% progress could indicate that some students delay the correction of this errors until the end of the modeling session.

Lifetime of the Diagnostics

To get a deeper insight into the modeling session data, we computed the lifetime of the diagnostics given to the students. We define the diagnostic lifetime as the elapsed time between the moment a student introduces a mistake in the model, and the moment that mistake is corrected. Note that this metric is independent of the validations made by the student, since diagnostics are computed for all snapshots regardless of the student used the validation function or not.

The average lifetimes follow a long-tailed distribution (Figure 5.6). That is, in the average case mistakes are quickly corrected by the students. However, for a few cases, it can take a very long time to solve those mistakes.

Relation of Number of Validations and Errors

Finally, we studied whether there was a correlation between the number of validations performed by the students, and the number of bad diagnostics obtained. To that end, we performed a Pearson's correlation test on the following variables, measured for each modeling session: V_s = "Number of validations", V_c = "Number of **complete** validations", D_{avg} = "Average number of bad diagnostics during the modeling session",

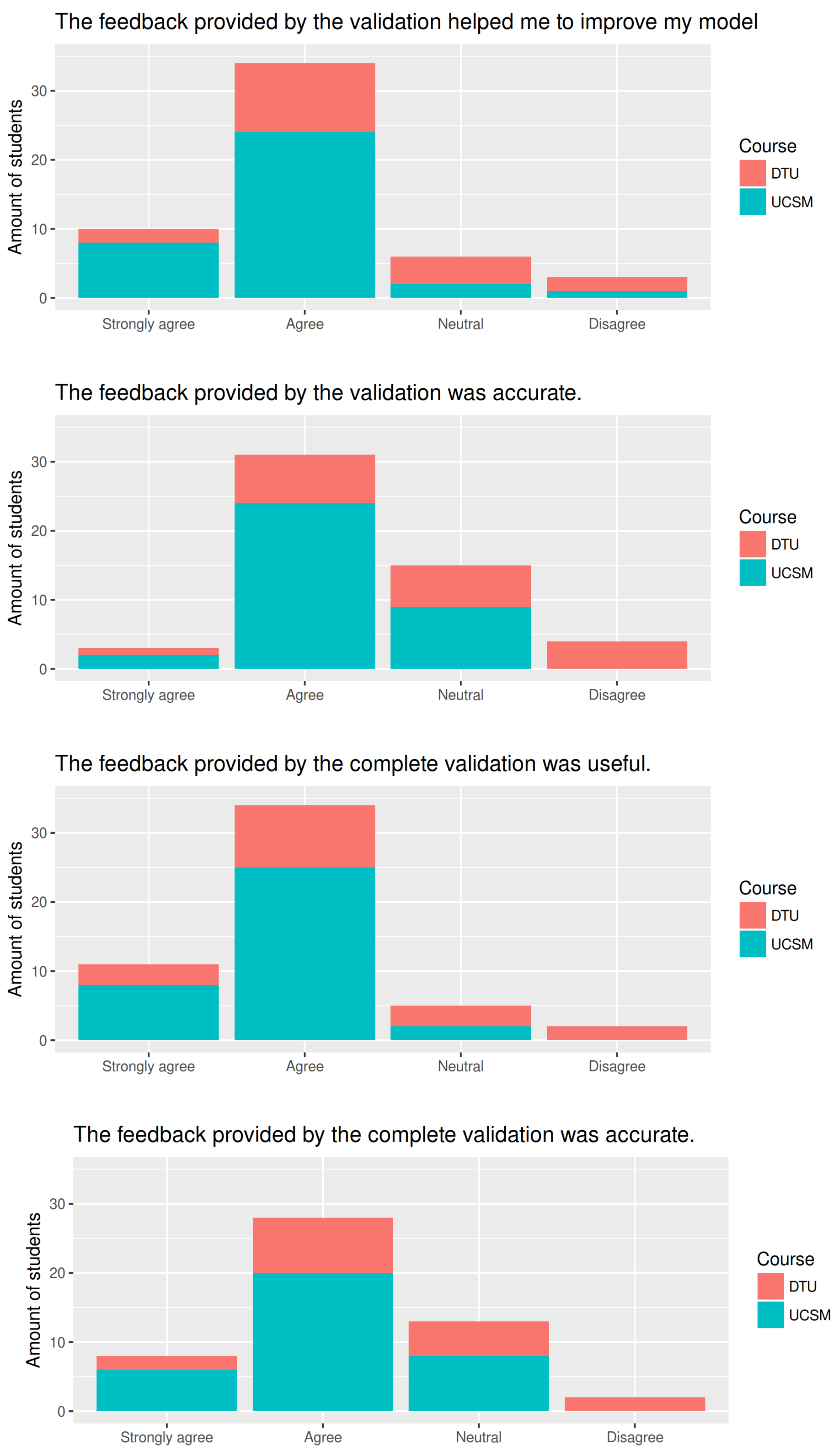


Figure 5.7: Results of the student survey for the modeling courses.

	Correlation Coefficient			**Test p-value**		
	DTU	**UCSM**	**TOTAL**	**DTU**	**UCSM**	**TOTAL**
V_s ˜D_{avg}	-0.693	-0.626	-0.664	8.77×10^{-5}	9.04×10^{-8}	3.33×10^{-12}
V_s ˜D_{end}	-0.636	-0.498	-0.558	4.74×10^{-4}	5.07×10^{-5}	2.34×10^{-8}
V_c ˜D_{avg}	-0.540	-0.305	-0.443	4.42×10^{-3}	1.77×10^{-2}	1.91×10^{-5}
V_c ˜D_{end}	-0.219	-0.406	-0.397	3.85×10^{-2}	1.27×10^{-3}	$1.57 \times 10^{-}4$

Table 5.1: Pearson's correlation test results for the relevant pairs of studied variables (shown per course and total).

D_{end} = "Number of bad diagnostics at the end of the exercise".

Besides the obvious correlations V_c ˜V_s and D_{avg} ˜D_{end}, we also found strong negative correlations between the two pairs of variables, as seen in Table 5.1. That is, when the number of validations grows, the number of errors decreases. While all the correlations were statistically significant, we can see how the ones concerning the number of complete validations are weaker than the other two. This can be explained by the fact that a large group of students did not use the complete validation, or did so only at the end of the session without addressing the feedback, while almost all students used the simple validation. Additionally, we observed that the correlations found for the individual courses are less strong than considering the 72 students as one group.

5.5.2 Student Survey

In the survey, the students were asked demographic information, native language and line of work, as well as their degree of agreement on some aspects of the application with four questions, aimed at independently evaluating the accuracy and usefulness of the validation and complete validation functionalities. Finally, the students also were asked to write down any complaints and/or improvement suggestions.

Figure 5.7 shows the results of the survey: A majority (75.4% in average) of students either agreed or strongly agreed in all questions. So we can say that the general perception is that using Model Judge was beneficial to the students.

When looking separately at the questions regarding *usefulness* versus the ones regarding *accuracy*, we can see a difference, with the students having a stronger level of agreement with the usefulness (84.0% on average) than accuracy (67.0% on average). We can thus conclude the student's perception was that the tool was useful, but not as accurate. This can be validated from the student's written suggestions, where in some cases they complained about the tool not understanding their process model labels.

By comparing independently the results of the two modeling courses, we can see the students from DTU were more critical with the tool (58.3% on average either agreed or strongly agreed) than the ones from UCSM (83.6% on average). We believe this difference can be partly explained by the presence of some technical issues during the DTU modeling course that were fixed for the UCSM one.

Chapter 6

Conclusions and Future Work

This Master Thesis project has been divided in three major goals. The design of ATD, the implementation of ATDlib, and the development of the algorithm behind the Model Judge platform. We conclude that the three main initial goals have been achieved successfully.

ATD are a novel alternative for business process documentation based on natural language. Being a more understandable alternative to classical graphical and rule-based notations, we expect ATD to be a useful documentation format for organizations, as well as enable the same automation capabilities as more formal languages, like BPMN. Finally, being annotated texts, mainstream adoption of ATD can help fill the lack of training data in the field of Natural Language Processing for Business Process Management. Future work in the development of ATD will involve solidification of the language: Deploy ATD-based solutions in different application domains to see how to adapt the base language to the wide variety of requirements in organizations.

ATDlib is the reference implementation of ATD. It can parse, automatically annotate and convert ATD to various useful representations. The current implementation is still in a very early stage, but it has been successfully deployed in the Model Judge platform. However, further effort is necessary to allow easy integration of ATDlib into any BPM system. One of the action points in that direction is to provide a Java interface into the library to aid in ATD adoption for organizations using any other JVM language. Another remaining functionality is to add support for ATD extensions into the library. Finally, to allow for easy generation and annotation of ATD, a visual front-end must be developed by fully integrating ATDlib into a text annotation interface.

Model Judge can help students learn modeling notations, such as BPMN, and has already been tested in two pilot university courses. It is thus an example of the usage of Annotated Textual Descriptions in a real use-case. Future work directions with Model Judge will come in two ways. In one hand, we want to expand the usage of *Model Judge* to more courses and universities. This will require considering new kinds of useful diagnostics, and especially implementing a robust way for fine-grained detection of control flow inconsistencies in process models. On the other hand, we want to get more detailed insights into the students' *process of process modeling*. By collecting anonymised and consented analytics information with *Model Judge*, we believe we can help get a better grasp of how people learn to model information, which will in turn help us develop better and more intuitive models.

Bibliography

[1] Arthur HM ter Hofstede, Wil MP van der Aalst, Michael Adams, and Nick Russell. *Modern Business Process Automation: YAWL and its support environment.* Springer Science & Business Media, 2009.

[2] Michele Chinosi and Alberto Trombetta. “BPMN: An introduction to the standard”. In: *Computer Standards & Interfaces* 34.1 (2012), pp. 124–134.

[3] Henrik Leopold, Jan Mendling, and Artem Polyvyanyy. “Supporting process model validation through natural language generation”. In: *IEEE Transactions on Software Engineering* 40.8 (2014), pp. 818–840.

[4] Han van der Aa, Henrik Leopold, Felix Mannhardt, and Hajo A Reijers. “On the fragmentation of process information: Challenges, solutions, and outlook”. In: *International Conference on Enterprise, Business-Process and Information Systems Modeling.* Springer. 2015, pp. 3–18.

[5] Ruopeng Lu and Shazia Sadiq. “A Survey of Comparative Business Process Modeling Approaches”. In: *Business Information Systems.* Ed. by Witold Abramowicz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 82–94. ISBN: 978-3-540-72035-5.

[6] Wil MP van Der Aalst, Arthur HM Ter Hofstede, Bartek Kiepuszewski, and Alistair P Barros. “Workflow patterns”. In: *Distributed and parallel databases* 14.1 (2003), pp. 5–51.

[7] Lluís Padró and Evgeny Stanilovsky. “FreeLing 3.0: Towards Wider Multilinguality”. In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012, Istanbul, Turkey, May 23-25, 2012.* 2012, pp. 2473–2479. URL: `http://www.lrec-conf.org/proceedings/lrec2012/summaries/430.html`.

[8] P Stenetorp, Sampo Pyysalo, Goran Topic, Tomoko Ohta, Sophia Ananiadou, and Jun’ichi Tsujii. *brat: a Web-based Tool for NLP-Assisted Text Annotation.* Apr. 2012.

[9] Angus Graeme Forbes, Kristine Lee, Gus Hahn-Powell, Marco Antonio Valenzuela-Escárcega, and Mihai Surdeanu. “Text Annotation Graphs: Annotating Complex Natural Language Phenomena”. In: *CoRR* abs/1711.00529 (2017). arXiv: `1711.00529`. URL: `http://arxiv.org/abs/1711.00529`.

[10] Jan Mendling, Bart Baesens, Abraham Bernstein, and Michael Fellmann. “Challenges of Smart Business Process Management: An Introduction to the Special Issue”. In: 100 (July 2017).

[11] Wil MP Van der Aalst. “The application of Petri nets to workflow management”. In: *Journal of circuits, systems, and computers* 8.01 (1998), pp. 21–66.

[12] Jakob Pinggera, Stefan Zugal, and Barbara Weber. “Investigating the Process of Process Modeling with Cheetah Experimental Platform–Tool Paper–”. In: *ER-POIS 2010* 13 (2010).

[13] Han van der Aa, Josep Carmona, Henrik Leopold, Jan Mendling, and Lluís Padró. “Challenges and Opportunities of Applying Natural Language Processing in Business Process Management”. In: (in press).

[14] Wil MP Van der Aalst. *Process mining: data science in action*. Springer, 2016.

[15] Niels Lohmann, Eric Verbeek, and Remco Dijkman. “Petri net transformations for business processes–a survey”. In: *Transactions on petri nets and other models of concurrency II*. Springer, 2009, pp. 46–63.

[16] Wil MP Van Der Aalst and Arthur HM Ter Hofstede. “YAWL: yet another workflow language”. In: *Information systems* 30.4 (2005), pp. 245–275.

[17] Roman V. Yampolskiy. “Turing Test as a Defining Feature of AI-Completeness”. In: (2012).

[18] Jörg Becker, Patrick Delfmann, Sebastian Herwig, L. Lis, and Armin Stein. “Formalizing Linguistic Conventions for Conceptual Models”. In: *Conceptual Modeling - ER 2009*. LNCS. Springer Berlin Heidelberg, 2009, pp. 70–83.

[19] 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”. In: *Decision Support Systems* 56 (2013), pp. 310–325.

[20] Bram Van der Vos, Jon Atle Gulla, and Reind van de Riet. “Verification of conceptual models based on linguistic knowledge”. In: *Data & Knowledge Engineering* 21.2 (1997), pp. 147–163.

[21] V. Gruhn and R. Laue. “Detecting Common Errors in Event-Driven Process Chains by Label Analysis”. In: *Enterprise Modelling and Information Systems Architectures* 6.1 (2011), pp. 3–15.

[22] Henrik Leopold, Christian Meilicke, Michael Fellmann, Fabian Pittke, Heiner Stuckenschmidt, and Jan Mendling. “Towards the automated annotation of process models”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2015, pp. 401–416.

[23] C. Francescomarino and P. Tonella. “Supporting Ontology-Based Semantic Annotation of Business Processes with Automated Suggestions”. In: *International Conference on Enterprise, Business-Process and Information Systems Modeling*. Vol. 29. LNBIP. Springer, 2009, pp. 211–223.

[24] M. Born, F. Dörr, and I. Weber. “User-Friendly Semantic Annotation in Business Process Modeling”. In: *WISE 2007 Workshops*. Vol. 4832. LNCS. Springer, 2007, pp. 260–271.

[25] A. Sinha and A. Paradkar. “Use Cases to Process Specifications in Business Process Modeling Notation”. In: *IEEE International Conference on Web Services*. 2010, pp. 473–480.

[26] João Carlos de Gonçalves, 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*. IEEE. 2009, pp. 161–166.

[27] Elena Viorica Epure, Patricia Martín-Rodilla, Charlotte Hug, Rebecca Deneckère, and Camille Salinesi. “Automatic process model discovery from textual methodologies”. In: *Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on*. IEEE. 2015, pp. 19–30.

[28] Aditya Ghose, George Koliadis, and Arthur Chueng. “Process discovery from model and text artefacts”. In: *Services, 2007 IEEE Congress on*. IEEE. 2007, pp. 167–174.

[29] Fabian Friedrich, Jan Mendling, and Frank Puhlmann. “Process model generation from natural language text”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2011, pp. 482–496.

[30] Matt Selway, Georg Grossmann, Wolfgang Mayer, and Markus Stumptner. “Formalising natural language specifications using a cognitive linguistic/configuration based approach”. In: *Information Systems* 54 (2015), pp. 191–208.

[31] Henrik Leopold, Han van der Aa, Fabian Pittke, Manuel Raffel, Jan Mendling, and Hajo A Reijers. “Searching textual and model-based process descriptions based on a unified data format”. In: *Software & Systems Modeling* (2017), pp. 1–16.

[32] Han van der Aa, Henrik Leopold, and Hajo A Reijers. “Checking Process Compliance against Natural Language Specifications using Behavioral Spaces”. In: *Information Systems* (2018).

[33] Josep Sànchez-Ferreres, Josep Carmona, and Lluís Padró. “Aligning Textual and Graphical Descriptions of Processes Through ILP Techniques”. In: *Advanced Information Systems Engineering*. Ed. by Eric Dubois and Klaus Pohl. Cham: Springer International Publishing, 2017, pp. 413–427. ISBN: 978-3-319-59536-8.

[34] Cristina Cabanillas, David Knuplesch, Manuel Resinas, Manfred Reichert, Jan Mendling, and Antonio Ruiz Cortés. “RALph: A Graphical Notation for Resource Assignments in Business Processes”. In: *Advanced Information Systems Engineering - 27th International Conference, CAiSE 2015, Stockholm, Sweden, June 8-12, 2015, Proceedings*. 2015, pp. 53–68.

[35] Paramita Mirza. “Extracting Temporal and Causal Relations between Events”. PhD thesis. International Doctorate School in Information and Communication Technologies, University of Trento, 2016.

[36] Wil MP van der Aalst. “On the representational bias in process mining”. In: *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2011 20th IEEE International Workshops on*. IEEE. 2011, pp. 2–7.

[37] Sergey Smirnov, Matthias Weidlich, and Jan Mendling. “Business process model abstraction based on behavioral profiles”. In: *Service-Oriented Computing*. Springer, 2010, pp. 1–16.

[38] Andrea Burattin. “PLG2: multiperspective processes randomization and simulation for online and offline settings”. In: *arXiv preprint arXiv:1506.08415* (2015).

[39] Rich Hickey. “The Clojure Programming Language”. In: *Proceedings of the 2008 Symposium on Dynamic Languages*. DLS ’08. Paphos, Cyprus: ACM, 2008, 1:1–1:1. ISBN: 978-1-60558-270-2. DOI: `10.1145/1408681.1408682`. URL: `http://doi.acm.org/10.1145/1408681.1408682`.

[40] Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Pierre-Francois Laquerre, Illés Solt, Jörn Kottmann, David McClosky, Antony Scerri, and Jon Crump. *brat standof format*. http://brat.nlplab.org/standoff.html. Accessed: 21-05-2018.

[41] Jan Mendling, Hajo A. Reijers, and Wil M. P. van der Aalst. "Seven process modeling guidelines (7PMG)". In: *Information & Software Technology* 52.2 (2010), pp. 127–136. DOI: 10.1016/j.infsof.2009.08.004. URL: https://doi.org/10.1016/j.infsof.2009.08.004.

[42] Wil M. P. van der Aalst, Kees M. van Hee, Arthur H. M. ter Hofstede, Natalia Sidorova, H. M. W. Verbeek, Marc Voorhoeve, and Moe Thandar Wynn. "Soundness of workflow nets: classification, decidability, and analysis". In: *Formal Asp. Comput.* 23.3 (2011), pp. 333–363. DOI: 10.1007/s00165-010-0161-4. URL: https://doi.org/10.1007/s00165-010-0161-4.

[43] Henrik Leopold, Sergey Smirnov, and Jan Mendling. "On the refactoring of activity labels in business process models". In: *Inf. Syst.* 37.5 (2012), pp. 443–459. DOI: 10.1016/j.is.2012.01.004. URL: https://doi.org/10.1016/j.is.2012.01.004.

[44] Jan Mendling, Hajo A Reijers, and Jan Recker. "Activity labeling in process modeling: Empirical insights and recommendations". In: *Information Systems* 35.4 (2010), pp. 467–482.

[45] Tadao Murata. "Petri nets: Properties, analysis and applications". In: *Proceedings of the IEEE* 77.4 (1989), pp. 541–580.

[46] Artem Polyvyanyy and Matthias Weidlich. "Towards a compendium of process technologies: The jBPT library for process model analysis". In: *Proceedings of the CAiSE'13 Forum at the 25th International Conference on Advanced Information Systems Engineering (CAiSE).* Sun SITE Central Europe. 2013, pp. 106–113.

[47] Thomas Lengauer and Robert Endre Tarjan. "A Fast Algorithm for Finding Dominators in a Flowgraph". In: *ACM Trans. Program. Lang. Syst.* 1.1 (1979), pp. 121–141. DOI: 10.1145/357062.357071. URL: http://doi.acm.org/10.1145/357062.357071.

[48] Inc. Gurobi Optimization. *Gurobi Optimizer Reference Manual.* 2016. URL: http://www.gurobi.com.

Appendix A

Source Code

All the source code that has been developed for this Master Thesis is available as a Git repository at `https://github.com/setzer22/master_thesis_source`. This appendix covers the relevant parts of the project and the necessary dependencies to build and execute it.

The repository presented consists of three main projects, divided in three subfolders in the repository root. All three projects are built using Maven and follow the Maven convention of folder structure: A `pom.xml` file specifies the build instructions and dependencies, and the `src/main/{java,clojure}` subfolders contain the source code for the corresponding project.

`nlp4bpm_commons` This project contains several utilities that are used across atdlib and all the algorithms of the *NLP4BPM* project. The relevant source files for this project are `freeling_api.clj` and `Freeling.clj`, which wrap FreeLing's JNI interface and `label_parser.clj` which implements the custom SRL module for BPMN labels described in Section 5.3.1.

`atdlib` ATDlib implements all functionalities described in Chapter 4. The source file names correspond to the same module names used in this document and thus should be self-explanatory.

`modeljudge` This project contains the implementation for the *Model Judge* algorithm. The major part of this module consists of an adaptation of the alignment between textual descriptions and BPMN process models. The parts that are specific to *Model Judge* are in the `core_students.clj` namespace.

To build some of the projects, some Java dependencies not available in any maven repository must be included. These are distributed inside the `jars` folder of the relevant project, with a bash script that installs them into the local Maven repository. Additionally, there are two native dependencies that cannot be bundled into the project due to size and/or license restrictions: FreeLing 4.0 or higher[1] and the Gurobi ILP solver[2]. Please note that Gurobi requires an academic or commercial license. These must be installed in the target system and listed in the system's `PATH` and `LD_LIBRARY_PATH` variables (or the equivalent in Windows operating systems).

[1] http://nlp.lsi.upc.edu/freeling/index.php/node/30
[2] http://www.gurobi.com/downloads/download-center