



**UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH**

Thesis presented in partial fulfillment
of the requirements for the Degree of
Ph.D. in Computing

Bridging the Gap Between Textual and Formal Business Process Representations

Josep Sànchez-Ferrerres

Advisor: Josep Carmona Vargas

Co-Advisor: Lluís Padró Cirera

Computer Science Department
Universitat Politècnica de Catalunya

Barcelona, July 2021

Preface

Please note that this thesis document is structured as an *article compendium*. In this style, all core contributions presented are included directly in the form of *publications*, which are listed in the Appendices. Thus, the chapters in this document serve as an introductory guide to provide necessary context and tie all the contributions together.

Acknowledgments

I would like to express my most sincere gratitude to my advisors, Prof. Josep Carmona and Prof. Lluís Padró, for their guidance and support throughout this journey. It is not at all an understatement that this project would have not been possible without their vast expertise, mentorship, and all those afternoons endlessly discussing algorithms in the blackboard. Moreover, I am very proud to say that I finish this Ph.D. knowing that this is only the beginning of our shared project together. I am certain the path ahead with them, together with Dr. Marc Solé, is full of interesting challenges, good experiences and that I still can learn a lot more from them.

I would also like to extend my acknowledgements to all the professors with whom I have been able to directly collaborate with and learn from during this period of my academic career: Dr. Han van der Aa, Prof. Andrea Burattin, Prof. Marco Montali and Prof. Barbara Weber among many others, have been the greatest possible example to the question of “*What it means to be a good researcher*”. I also want to express my gratitude towards the professors that have helped me in the development and growth of the *ModelJudge* platform. Not only for their feedback, but also for helping me test the system in their modelling courses: Dr. Amine Abbad Andaloussi, Prof. Guillermo Calderón and Prof. Ernest Teniente among many others.

During this time pursuing my Ph.D., and specially when put in contrast with the global situation in the 2020-21 academic year, I feel I have learned that one of the most important parts of being a researcher are the social interactions. Working together with other fellow undergrad researchers has been a very pleasant experience, helping me grow both personally and academically. Thanks to countless discussions with them, I have been able to deepen my knowledge in my own domain as well as broaden it to vastly different fields of research. And for this and many more, I would like to thank all my colleagues at the office: Alex Vidal, Alexandra Yamaui, Albert Calvo, Bernat Coma, Christian Lezcano, Ivan Paz, Lucas Machado, Luis Delicado, Luis Quishpi, Tuomas Hakoniemi, and all the other wonderful people I’ve had the pleasure to cross paths with during this start of my academic career.

Finally, I would also like to thank my family, and specially my parents as well as Montse, for their continued support. Not only for believing in me and helping me along every step towards pursuing my goals, but also for their active role in sitting through the countless rehearsals to my presentations and for offering all sorts of advice. I would like to also extend my gratitude to all of my friends, for all the support and great times we’ve been able to share during this time.

Bridging the Gap Between Textual and Formal Business Process Representations

Josep Sànchez-Ferreres

Abstract

Topics: Artificial Intelligence (1203.04), Information systems and components (1203.08)

In the era of digital transformation, an increasing number of organizations are starting to think in terms of *business processes*. Processes are at the very heart of each business, and must be understood and carried out by a wide range of actors, from both technical and non-technical backgrounds alike.

When embracing digital transformation practices, there is a need for all involved parties to be aware of the underlying business processes in an organization. However, the representational complexity and biases of the *state-of-the-art* modeling notations pose a challenge in understandability. On the other hand, plain language representations, accessible by nature and easily understood by everyone, are often frowned upon by technical specialists due to their ambiguity.

The aim of this thesis is precisely to bridge this gap: Between the world of the technical, formal languages and the world of simpler, accessible natural languages. Structured as an article compendium, in this thesis we present four main contributions to address specific problems in the intersection between the fields of natural language processing and business process management.

Bridging the Gap Between Textual and Formal Business Process Representations

Josep Sànchez-Ferreres

Resum

Temes: Intel·ligència artificial (1203.04), Sistemes d'informació i components (1203.08)

A l'era de la transformació digital, cada vegada més organitzacions comencen a pensar en termes de *processos de negoci*. Els processos són el nucli principal de tota empresa i, com a tals, han de ser fàcilment comprensibles per un ampli ventall de rols, tant perfils tècnics com no-tècnics.

Quan s'adopta la transformació digital, és necessari que totes les parts involucrades estiguin ben informades sobre els protocols implantats com a part del procés de digitalització. Tot i això, la complexitat i biaixos de representació dels llenguatges de modelització que actualment conformen l'*estat de l'art* sovint en dificulten la seva comprensió. D'altra banda, les representacions basades en documentació usant llenguatge natural, accessibles per naturalesa i fàcilment comprensibles per tothom, moltes vegades són vistes com un problema pels perfils més tècnics a causa de la presència d'ambigüitats en els textos.

L'objectiu d'aquesta tesi és precisament el de superar aquesta distància: La distància entre el món dels llenguatges tècnics i formals amb el dels llenguatges naturals, més accessibles i senzills. Amb una estructura de compendi d'articles, en aquesta tesi presentem quatre grans línies de recerca per adreçar problemes específics en aquesta intersecció entre les tecnologies d'anàlisi de llenguatge natural i la gestió dels processos de negoci.

Contents

1	Introduction and Motivation	7
1.1	Problem description	7
1.2	Goals of this thesis	8
1.3	Main Contributions	8
1.4	List of publications:	9
1.5	Structure of this document	10
2	Background	11
2.1	Business process management	11
2.2	Modelling Language Notations	12
2.3	Formal methods	14
2.3.1	Linear Temporal Logic.	15
2.3.2	Model checking	15
2.4	Natural language processing	16
2.4.1	Steps of Language Analysis	17
2.4.2	Text Annotations	18
3	State of the Art	21
3.1	Natural language processing in BPM	21
3.1.1	Creating Process Models from Textual Descriptions	21
3.1.2	Converting Process Models into Textual Descriptions	22
3.1.3	Comparing Textual and Formal Process Descriptions	23
3.1.4	Handling Ambiguity	23
3.2	End-to-end systems for NLP	24
3.3	Formal reasoning on top of process models	25
3.4	Educational systems for modelling	25

4	Main Contributions	27
4.1	Formal Reasoning on Natural Language Descriptions of Processes . . .	27
4.1.1	Related publications	27
4.1.2	Main highlights	27
4.2	Aligning textual and graphical descriptions of processes	30
4.2.1	Related publications	30
4.2.2	Main highlights	31
4.3	<i>ModelJudge</i> : Process modelling in education	32
4.3.1	Related publications	32
4.3.2	Main highlights	33
4.4	Conversational process models	35
4.4.1	Related publications	36
4.4.2	Main highlights	36
5	Conclusions and Future Work	39
5.1	Summary	39
5.2	Future Work	40
6	List of Publications	43
I	Publication: Aligning Textual and Model-Based Descriptions	51
II	Publication: Supporting the Process of Learning and Teaching ...	93
III	Publication: Aligning Textual and Graphical Descriptions ...	109
IV	Publication: Formal Reasoning on Natural Language ...	125
V	Publication: From Process Models to Chatbots	143
VI	Publication: Unleashing Textual Descriptions of Business Processes	159
VII	Publication: The Model Judge – A Tool for Supporting Novices ...	193

Chapter 1

Introduction and Motivation

1.1 Problem description

A business process is a set of activities that must be executed in compliance with a set of constraints, with the goal of offering a product or service. Business processes can be found in any organization. In order to achieve full digital transformation, companies often embrace Business Process Management (BPM) techniques, which are nowadays applied in many different contexts.

When organizations wish to apply BPM to their workflows, a way of documenting business processes is required. This is why process models, i.e. formal, unambiguous descriptions of business processes, are a fundamental pillar of BPM. The use of formal process models allows automated analysis on an organization's processes, which in turn allows for a more optimal use of their resources.

However, the subtleties in the formal semantics of these notations are often difficult to grasp for non-technical users—stakeholders and employees alike [1]–. Often, this forces organizations to maintain less structured forms of process documentation alongside the process descriptions. Misalignment between several representations of process descriptions can have serious consequences for organizations [2].

For the above reasons, no process modelling methodology has ultimately stood above the others. This has led to an 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 the formal spectrum, we encounter a plethora of modelling languages, each one aiming to solve a different subset of problems [3, 4]. Finally, even in formal notations, a huge part of the semantic descriptions are delegated to short text snippets written in natural language.

It is thus evident that natural language in process documentation is an unavoidable reality. This situation is creating an increasingly large amount of digital but unstructured or semi-structured data in organizations. There is a clear opportunity to capitalize on these data sources to obtain insights that can be used to further enhance, better monitor, and more accurately document business processes.

During the last forty years, the *Natural Language Processing* (NLP) community has provided algorithms and tools for automating the analysis of human language. And

recent advances in hardware technology have widened even more the scope of what this technology is capable of producing. By bringing these innovations to the field of BPM, companies and organizations can finally benefit from their otherwise unexploited sources of process documentation. This has been a recent trend in the BPM community for the past few years, and is the main driving factor behind this research.

1.2 Goals of this thesis

Identifying the opportunities and challenges presented by the previously described situation, this thesis sets out to achieve the following high-level goals:

1. **Improving automatic analysis of process documentation.** Acknowledging natural language as an inherent part of process documentation, this thesis aims to propose algorithmic techniques that allow automatic analysis of such unstructured and semi-structured language sources. From short text snippets present in *Business Process Model and Notation* (BPMN) documents, to plain text descriptions often employed by organizations.
2. **Making natural language a first-class asset.** Natural language is currently frowned upon when considering formal process documentation. However, it has many desirable properties that other formal notations have failed to deliver on. A goal of this thesis is to explore and propose techniques that, when enhanced by artificial intelligence, would bring informal text to the level of formal documentation. Furthermore, the aim of these techniques is to ultimately enable formal reasoning via approaches such as *model checking* on top of natural language textual descriptions.
3. **Analyzing usage of formal and informal modelling languages.** The task of modelling is an act of formalization, and is mainly the product of an interaction between a modeler and a modelling language, be it formal or unstructured. This thesis is set to analyze how this interaction takes place, in the form of modelling. By doing so, we aim to uncover valuable insights of how natural language relates to process documentation, find the natural ways in which typical control flow patterns are described, and how natural language fits into the task of formalizing reality as a business process model.
4. **Enhancing the process modelling experience.** To be used in automation, process documentation must be accurate and error-free. Unfortunately, current process documentation formats are capable of representing in practice many situations undesirable in an actual deployment. To alleviate this, current modelling tools offer feedback to improve process model quality, but this feedback is usually limited to the syntactical aspects. One of the goals of this thesis is to design algorithms capable of providing more detailed feedback to improve the modelling experience, especially when concerning the actual *semantics* of a process.

1.3 Main Contributions

This thesis is structured into several lines of research, ranging from the theoretical aspects of language formalization to more practical research focused on the *design science* methodology, in which the main focus is to validate an approach by means of constructing

an actual artifact that implements it. The four main lines of research that will be covered in this thesis are next listed:

1. **Formalizing textual process descriptions.** Described in Chapter 4, Section 4.1, in this line of research we introduce *Annotated Textual Descriptions of Processes* (ATDP): A language and accompanying methodology, rooted in temporal logics and process trees, that allows formally specifying the underlying business process in a textual description. ATDP enables reasoning on top of textual descriptions by embracing and capturing ambiguity.
2. **Computing alignments between textual and formal process models.** Described in Chapter 4, Section 4.2, in this line of research we describe the design and implementation of an algorithm to compute optimal alignments between an informal textual description and a formal BPMN business process specification. This algorithm is then empirically validated against both previously existing and newly annotated validation benchmarks, to show a high level of accuracy and a wider coverage of BPMN constructs.
3. **Proposing an educational environment for process modelling.** Described in Chapter 4, Section 4.3, in this line of research we describe the design process and implementation of the *ModelJudge*: A web platform designed to help students and teachers improve the experience behind a process modelling course. *ModelJudge* is capable of, without any human intervention, suggest corrections based on the syntactic, semantic and pragmatic aspects of a novice modeler’s business process diagram. The quality of the generated feedback is then validated using both quantitative and qualitative metrics based on data collected from two separate modelling courses held in universities.
4. **Automating the creation of conversational agents from processes.** Described in Chapter 4, Section 4.4, in this line of research we describe the implementation of a technique to convert a BPMN process diagram into a conversational agent able to explain the process. Based on existing *state-of-the-art* techniques for language generation from process models, the technique presented is able to produce agents capable of answering specific questions about a process model or provide detailed step-by-step instructions. An empirical, qualitative evaluation shows great potential for the achieved results, but also evidences the need of further iterations to achieve a more consistent level of quality.

1.4 List of publications:

This thesis is an aggregation of the the following publications. For each publication, impact factor or GGS rating is listed accordingly.

- Josep Sànchez-Ferreres, Han van der Aa, Josep Carmona and Lluís Padró. “Aligning textual and model-based process descriptions”. In: *Data Knowledge Engineering* 118 (2018), pp. 25-40. (Impact Factor: 1.583, Q3)
- Josep Sànchez-Ferreres, Luis Delicado, Amine Abbad Andaloussi, Andrea Burattin, Guillermo Calderón-Ruiz, Barbara Weber, Josep Carmona and Lluís Padró. “Supporting the Process of Learning and Teaching Process Modelling”. In: *Transactions on Learning Technologies (TLT)*. 2019. (Impact Factor: 2,315, Q2)

- Josep Sànchez-Ferreres, Josep Carmona and Lluís Padró. “Aligning Textual and Graphical Descriptions of Processes Through ILP Techniques”. In: Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CaiSE). 2017. pp. 413-427. (GGs Rating: A)
- Josep Sànchez-Ferreres, Andrea Burattin, Josep Carmona, Marco Montali and Lluís Padró. “Formal Reasoning on Natural Language Descriptions of Processes”. In: Proceeding of the 17th International Conference on Business Process Management (BPM). 2019. (GGs Rating: A)
- Anselmo López, Josep Sànchez-Ferreres, Josep Carmona and Lluís Padró. “From Process Models to Chatbots”. In: Proceedings of the 31th International Conference on Advanced Information Systems Engineering (CAiSE). 2019. pp. 383-398. (GGs Rating: A)
- Josep Sànchez-Ferreres, Andrea Burattin, Josep Carmona, Marco Montali, Lluís Padró, and Luís Quishpi, “Unleashing textual descriptions of business processes”. In: Software and Systems Modeling. 2021. pp. 1-23. (Impact Factor: 1.915, Q1)

1.5 Structure of this document

This thesis document is structured as an article compendium. The opening chapters are structured as a *reading guide*. After the full list of publications in Chapter 6, follows the appendices, with a collection of *all the peer-reviewed manuscripts* presented as part of this thesis. Publications are presented verbatim in their pre-print form, and can be read on their own as self-contained documents. However, the preceding reading guide serves as an introduction to provide necessary context and relate the four main lines of research in this thesis.

For the reading guide, Chapter 2 provides an overview of well-established topics that are connected to this work. Next, Chapter 3, discusses topics under active research which serve as a basis, and are related to, the work presented in this thesis. Chapter 4 introduces each of the main contributions presented as part of this article compendium. Finally, Chapter 5 provides general conclusions to this thesis, beyond each individual contribution and discusses opportunities for future work in this area.

Chapter 2

Background

This chapter provides an overview of well-established topics that are connected to this work. Section 2.1 discusses the field of *business process management*. Next, Section 2.2 goes over some of the most influential modelling language notations, emphasizing those that are especially relevant for this thesis. Afterwards, Section 2.3 contains a brief overview of work in the field of *formal methods* that is necessary to understand some contributions in this thesis. Finally, Section 2.4 shows an overview of the field of *natural language processing*.

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 operations research with a focus on how to *communicate, document, analyze* and *enhance* business processes.

Business Process Management approaches processes at three different levels [5], 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. Prior work in this area is discussed in Section 2.2

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

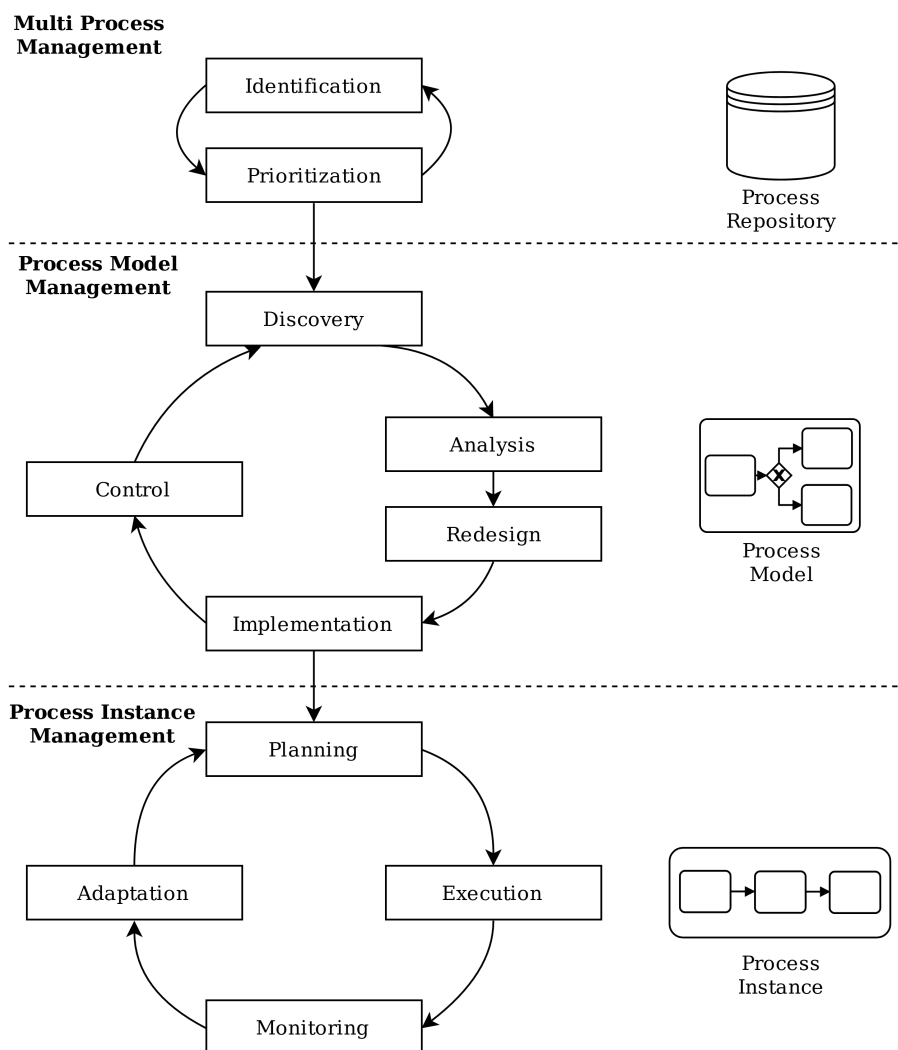


Figure 2.1: The three abstraction levels of business process management. Adapted from the figure in [5].

[6], to empirical psychological research about the *process of process modelling* [7]. The project described in this thesis is encompassed in an emerging field [8] in BPM, focused in the relation between human natural language and the formal methods found in other fields of research, such as *Process Mining* [9]. Related work in this field is discussed later on in Section 3.1

2.2 Modelling Language Notations

In the literature, one can find numerous approaches to business process modelling techniques. In their survey [3], Lu et. al. classify *business process notations* (BPN) –in this work also referred as process modelling languages– into graph-based and rule-based notations. These two categories are also often referred to as *imperative* and *declarative* process modelling languages, respectively.

Graph-based BPN represent business processes as a control flow graph. They are heavily based on, and usually convertible to Petri Nets [10], a formal mathematical model usually employed to describe distributed systems. A more restricted, but useful formalism to represent this type of processes are *Process Trees*.

A Process Tree [11] is a formalism to describe block-structured imperative processes. It is a tree structure, consisting of branching nodes and leaves. A branching node is labeled with an operator: Sequence (\rightarrow), Conflict (\times), Parallel ($||$), Inclusive or (\vee) and Loop (\hookrightarrow). Leaf nodes represent activities. The possible executions of a process described by a process tree are defined by recursively combining the semantics of each operator. Figure 2.2 shows an example process tree describing a process that starts with a loop of *a*, and continues by taking the choice of either doing *b* and *c* in parallel or executing just *d*. Process Trees are an interesting formalism because despite being more restrictive than petri nets, the processes described are guaranteed to have very interesting properties such as *soundness* [12], and are by definition considered *block-structured*, a property that's not always easy to achieve for general graphs. [13]. This is why Process Trees are often preferred when automatically constructing process models, which can then be converted to any graph-based notation.

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) [14], 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 [15] have been proposed both from the academics and the industry. Figure 2.3 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

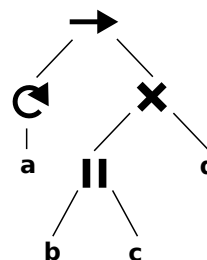


Figure 2.2: A Process Tree with 4 operators and 4 activities.

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 further specify the semantics of 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. Unlike *process trees*, most graphical notations do not enforce processes to be block-structured. This allows processes modelled in graph-based notations to follow very complex control flows, equivalent to the kind which are only obtainable by using free *goto* statements in traditional imperative programming languages.

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 2.3 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.

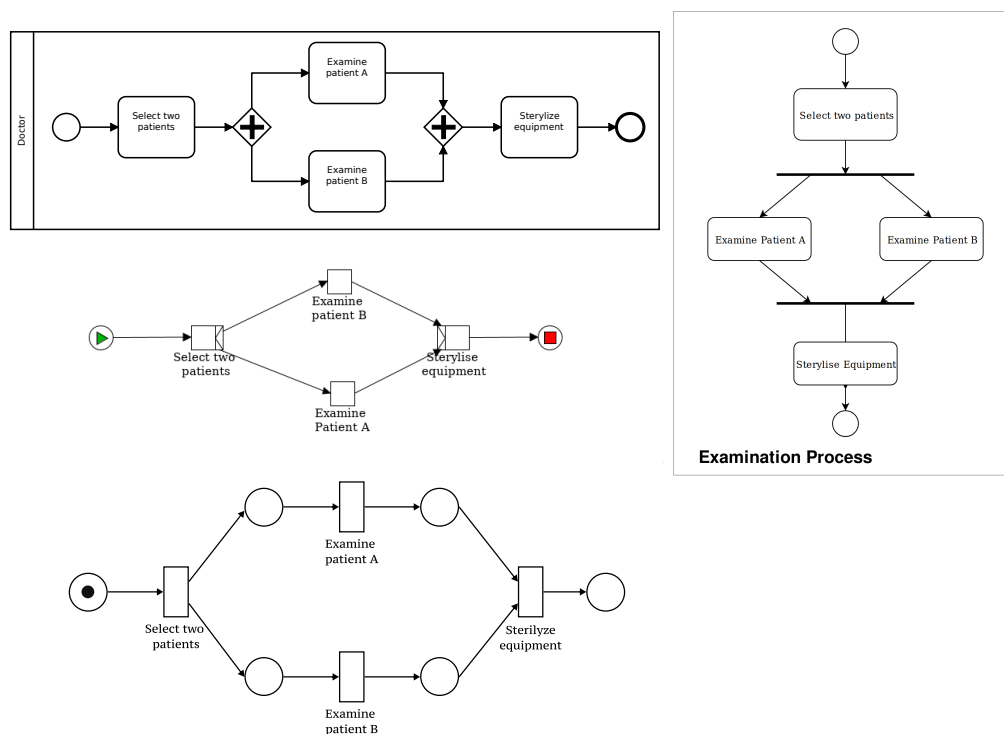


Figure 2.3: Example process in different graph-based languages: BPMN (top-left), YAWL (mid-left), Petri Net (bottom-left), BPEL (right).

One of the most prominent rule-based notations in the BPM literature is *Declare*. This language allows declarative specification of processes. Declarative modelling languages aim to not over-specify processes, and thus are useful when wide ranges of behaviour are possible within the same process. A Declare model is defined as set of constraints over a set of activities. For instance, the $response(a, b)$ constraint specifies that after any execution of a , an execution of b should eventually occur. Table 2.1 shows the informal definition of some of the most common declare constraints. The original semantics of Declare were formulated in Linear Temporal Logic (LTL), described in detail in Section 2.3. However, alternative implementations exist for Computation Tree Logic (CTL) [16] and Regular Expressions [17].

Constraint	Definition
$precedence(a, b)$	Any execution of b must be preceded by an execution of a
$response(a, b)$	After any execution of a , b will eventually occur
$succession(a, b)$	Both $precedence(a, b)$ and $response(a, b)$
$not-coexist(a, b)$	Activities a and b cannot both occur in the same trace

Table 2.1: Informal definitions of some Declare constraints.

2.3 Formal methods

Formal methods is a field of research with a focus on developing mathematically rigorous techniques used for the specification, design and verification of software and hardware

systems. In this section, we briefly overview two main contributions of the field that are of particular relevance to this thesis: *Linear Temporal Logic* and *Model checking*.

2.3.1 Linear Temporal Logic.

As seen later on in Chapter 4, a type of modal logic called *Linear Temporal Logic* (LTL) [18] is used in the definition of ATDP. This formalism is also the basis for the Declare rule-based process modelling language described in the previous section. Temporal logic are a type of logic that allows stating facts about sequences of atomic events over time. For example, by stating $\Box(a \rightarrow \Diamond b)$, we are saying that the occurrence of event a at any point in time, implies event b will be executed eventually in the future.

LTL formulae are built from a set \mathcal{P} of propositional symbols and are closed under the boolean connectives, the unary temporal operator \bigcirc (*next-time*) and the binary temporal operator U (*until*):

$$\varphi ::= a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 U \varphi_2 \text{ with } a \in \mathcal{P}$$

Intuitively, $\bigcirc\varphi$ says that φ holds at the *next* instant, $\varphi_1 U \varphi_2$ says that at some future instant φ_2 will hold and *until* that point φ_1 always holds. Common abbreviations used in LTL include the ones listed below:

- Standard boolean abbreviations, such as \top , \perp , \vee , \rightarrow .
- $\Diamond\varphi = \top U \varphi$ says that φ will *eventually* hold at some future instant.
- $\Box\varphi = \neg\Diamond\neg\varphi$ says that from the current instant φ will *always* hold.
- $\varphi_1 W \varphi_2 = (\varphi_1 U \varphi_2 \vee \Box\varphi_1)$ is interpreted as a *weak until*, and means that either φ_1 holds until φ_2 or forever.

As a means to illustrate the type of concepts expressible in LTL, consider the following temporal statements and their LTL encoding.

- *Every time it rains (r), my street floods after some time (f):* $\Box(r \rightarrow \Diamond f)$
- *Once the circuit breaker blows (b), power (p) is not restored until the leakage is resolved (r):* $\Box(b \rightarrow \neg p U r)$
- *To use this medication (m), the patient must not have had any prior surgeries (s):* $\Diamond s \rightarrow \neg s U m$

The semantics of temporal logic vary when considering their semantics over finite [19] or infinite traces of events [20]. In this work, we consider the standard interpretation of temporal logic formulae over *infinite traces*. To better illustrate this, consider the following set of statements: “**a** must be executed; **a** follows **b**; **b** follows **a**”. By interpreting the statements over infinite trace semantics, the formula is easily satisfiable with an endless loop of executions of **a** and **b**. However, this trick is not very useful to process models, where typically termination is a desirable property and the situation just described would be considered a *deadlock*. Note however that this theoretical distinction does not impose any limitation on the expressive power of the formalism for the particular use cases portrayed in this thesis.

2.3.2 Model checking

Model checking [21] is the field concerning techniques for the automatic verification of the properties of finite-state systems. Particularly, in this work, model checking techniques

are used to verify properties over processes, specified as a model representing all its possible executions.

Figure 2.4 illustrates the problem of model checking: Given a model description as a finite-state system and a property specification, usually expressed in different formalisms, check whether the property holds for the system, and in case it does not, provide a counterexample.

In the field of BPM, model checking can be used in the context of compliance checking. Business rules, as those arising from regulations or SLAs, impose restrictions that any process model in an organization may need to satisfy. On this regard, compliance checking methods assess the adherence of a process specification to a particular set of predefined rules.

Currently, there are various well-known implementations for model checking software. One of such implementations, used in this thesis (see Section 4.1), is NuSMV[22]. NuSMV allows the specification of transition systems using its own high-level language. On the other hand, properties can be specified using a wide range of well-known formalisms, including LTL and CTL.

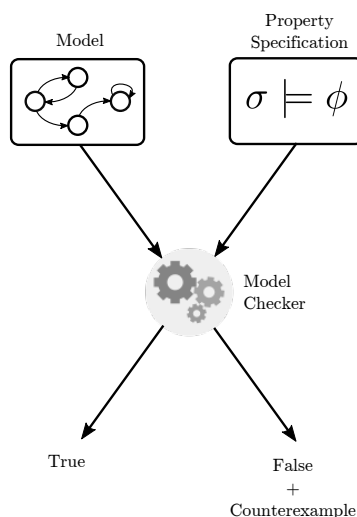


Figure 2.4: Overview of a model checker

2.4 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 [23, Section 1] problem, since texts 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 [24, 25, 26], or common modelling errors [27]. Other approaches use NLP to augment process models with semantic or ontological information [28, 29, 30]. This particular field of research is discussed in further detail later on in Section 3.1.

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 [31], group stories [32], or methodological descriptions [33], while others take general textual process descriptions as input [34, 35].

However, these approaches have been found to produce inaccurate results, which require manual inspection [36]. Other use cases involving texts include a technique that considers *work instructions* when querying process repositories [37] for conformance checking against textual process descriptions [38].

2.4.1 Steps of Language Analysis

The NLP processing software used in this work is FreeLing¹ [39], 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 [40] 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.

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

¹<http://nlp.cs.upc.edu/freeling>

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.

2.4.2 Text Annotations

Creating annotated versions of texts is usual in the 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* [41]², a configurable environment that allows the annotation of texts with labelled spans, and relations. Figure 2.5 shows *Brat* being used to annotate a scientific text with very specific concepts from the field of biology.

In Chapter 4, we present *ATDP* as a business process modelling language. *ATDP* is based on text annotations, but its main purpose differs from the typical use cases of annotations. Ideally, an *ATDP* would be automatically extracted by an NLP tool, instead of being used as training data. However, given the limitations of the current NLP state of the art, we resort to a certain amount of human annotation to improve the quality of our semantic representation.

²<http://brat.nlplab.org>

The screenshot shows the Brat annotation tool interface with the URL `/editable/BioNLP-ST_2011/BioNLP-ST_2011_EPI/PMID-19781662` and the Brat logo. The interface displays seven sentences, each with various entities and relations annotated. The entities are color-coded: green for 'DNA methylation', blue for 'Protein', 'Entity', and 'Pro', and purple for 'Site'. Relations are indicated by arrows with labels like 'Theme' and 'Site'.

- 1 Role of DNA methylation and methyl-DNA binding proteins in the repression of 5-lipoxygenase promoter activity.
Annotations: DNA methylation (green), Protein (blue), Entity (blue), Site (purple).
- 2 Human 5-lipoxygenase (5-LO) is the key enzyme in the formation of inflammatory leukotrienes.
Annotations: Protein (blue), Pro (blue).
- 3 5-LO gene expression is mainly restricted to B cells and cells of myeloid origin.
Annotations: Pro (blue).
- 4 It is known that basal 5-lipoxygenase promoter activity is regulated by DNA methylation.
Annotations: Protein (blue), Entity (blue), DNA methylation (green), Theme (relation).
- 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.
Annotations: DNA methylation (green), Site (purple), Pro (blue), Entity (blue), Theme (relation).
- 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.
Annotations: Pro (blue), Pro (blue), Pro (blue), DNA meth (green), Site (purple), Site (purple), Entity (blue), Site (purple).
- 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.
Annotations: Pro (blue).

Figure 2.5: One of the example annotations provided with the Brat annotation tool.

Chapter 3

State of the Art

This chapter introduces topics that are currently under active research, and serve as a basis for the work presented as part of this thesis. Section 3.1 discusses recent advances in the field of natural language processing in the field of BPM. Next, 3.2 briefly discusses new trends in the NLP community where the focus is shifting towards end-to-end machine learning systems. Later on, in 3.3, goes over the state of the art in formal systems for reasoning over process models. Finally, Section 3.4 describes several educational platforms for process modelling and other related disciplines.

The reader should note that, due to the structure of this thesis as an *article compendium*, the state of the art introduced in this section is meant as a broad overview of related works to this thesis. Later on, in the appendix chapters, a more in-depth exploration for the state of the art is provided in each of the publications.

3.1 Natural language processing in BPM

In order to automatically reason over a natural language process description, it is necessary to construct a formal representation of the actual process. The interaction between process models and textual descriptions has been studied from several angles in the literature. In this section overview several contributions that are of special relevance to this thesis: ranging from fully manual to automatic approaches.

3.1.1 Creating Process Models from Textual Descriptions

Converting an unstructured textual representation into a formal complete specification in a process modelling language is often desirable: This allows the resolution of ambiguities, often present in natural language, while enabling support for automation in process execution.

To achieve this, the first available option consists in converting a textual description into a process model by manually modelling the process. This approach, widely discussed [42, 43], has been thoroughly studied also from a psychological point of view, in order to understand which are the challenges involved in such process of process modelling [44, 45]. These techniques, however, do not provide any automatic support and the possibility for automatic reasoning is completely depending on the result of the manual

modelling. Therefore, ambiguities in the textual description are subjectively resolved.

On the opposite side of the automation spectrum, there are approaches that autonomously convert a textual description of a process model into a formal representation. Often, such representation is a final process model in a formal language, such as BPMN. Converting an unstructured textual representation into a formal complete specification is considered a very challenging task in the literature. With the inherent difficulties introduced by unstructured natural language, none of the *state-of-the-art* techniques to date have been able to achieve a level of quality that doesn't require further manual refinement of the produced artifact [46]. Moreover, fully automatic techniques are limited by the fact that they need to resolve ambiguities in the textual description. This can result in a single text interpretation being "hard-coded" into the resulting process model.

Often, restricting the input language from free text to a more rigid structure leads to more predictable inputs. In their work [32], Gonçalves et. al. introduce a technique capable of extracting process models from *user stories*. By building on a set of common language widely understood—especially by non-technical profiles usually involved in process definition—more accurate results can be obtained by decreasing the amount of ambiguity. Similar work in this field has also been proposed by Sinha et. al in [31].

In their work [35], Friedrich et. al. propose a technique capable of producing a complete process model in BPMN starting from an unrestricted textual description. The technique is mainly based on standard natural language processing techniques, and leverages the technology in the Stanford Parser [47] to achieve a good level of accuracy in a 47 model-text pairs validation benchmark.

Finally, the techniques introduced by Quishpi et. al. in [48] are closely related to the work introduced in this thesis. Instead of producing formal, complete BPMN specifications from a textual description, this contribution targets ATDP annotations: A new formalism introduced in this thesis which aims to be closer to natural language while still remaining convertible to more rigid representations of business process models via automatic reasoning. This approach decouples the semantic extraction phase from the technicalities of having to target a very rigid standard such as BPMN, and delegates this final conversion to other specialized tools. In Chapter 4, Section 4.1, we describe in further detail the ATDP formalism and how it relates to unstructured textual descriptions.

3.1.2 Converting Process Models into Textual Descriptions

Another common approach is to convert technical, structured process documentation into an unstructured natural language text describing the same process. The problems in this line of research are vastly different when compared to the approaches from the previous section. The challenge is shifted from having to handle ambiguous language—which is not an issue in this scenario—, and moves towards generating natural, believable sentences capable of replacing a human writer. This has often been approached in the literature as direct application of *Natural Language Generation* (NLG) techniques in the field of BPM.

However, this kind of transformation is not exclusively a NLG task. Most business process notations rely on semi-structured natural language fragments to denote most of the semantics of the process. This is why, often, NLU techniques must be combined with NLG in order to obtain a complete solution to the problem.

One of the first examples of text generation from business process models in BPMN

notation was introduced by Leopold et. al. in [49]. In that paper, a mixture of structural analysis, classical NLP-inspired heuristics and natural language generation were combined in order to automatically generate textual descriptions of Business Processes.

Other approaches, are focused with generating different kinds of unstructured documentation. In their work, Aysolmaz et. al. [50] describe a technique to generate requirements documents autonomously from a business process specification. As similarly described in the previous section, by restricting the scope of the natural language in a more rigid form, better results can be achieved without human intervention.

More recent attempts have been made at improving the results for text generation starting from a standard BPMN specification of a process model, inspired by the technique presented in [49]. That is the case for Publication V of this thesis, introduced in [51]. More details are provided later on in Section 4.4

3.1.3 Comparing Textual and Formal Process Descriptions

Establishing an alignment between two existing sources of documentation is sometimes desirable. This has been explored mainly in the field of process model matching: A set of techniques to find isomorphisms between formal business process diagrams for which there has been high interest in the BPM community [52]

In [53], Leopold et. al. propose a technique for process model matching. In it, a combination of structural information and semantic information contained in activity labels is used to perform the matching. The technique uses probabilistic optimization techniques in order to compute the best alignment between a pair of process models. This line of research is later on continued by Klinkmüller et. al. in [54]

Several authors in the literature, have proposed works combining the ideas and techniques of process model matching and NLU. Instead of matching two formal process models, techniques have been proposed with the of aligning a formal source of process documentation, such as a BPMN diagram, with an unstructured source of information, like a textual description.

One of the first contributions to tackle the problem was introduced by van der Aa et. al. [55]. In that contribution, a *bag-of-words* (BoW) method was used to extract and compare information from two distinct sources of information, which would then be compared using standard information theory techniques.

One of the main contributions of this thesis, presented in Chapter 4, Section 4.2, consisted on expanding on the original idea presented in [55] by using more advanced natural language extraction techniques and introducing a faster formalization to the same optimization problem which allowed computation in real-time. This led to the joint contribution in [56] where both techniques were combined and expanded to cover a wider range of BPMN constructs like events or gateways.

3.1.4 Handling Ambiguity

Throughout this section, we have been discussing ambiguity as one of the main barriers in the analysis of unstructured natural language data sources. This is especially important when the goal is the translation into a formal language. Human writers assume some implicit background context is needed to understand a text, and frequently introduce statements that are not self-contained, i.e. cannot be understood without external

knowledge and reasoning capabilities. Textual descriptions of business processes are a clear example of this phenomena.

Managing ambiguity in textual descriptions has been widely acknowledged as a difficult challenge in the BPM literature[57] as well as related fields, such as requirements engineering[58, 59]

Being an ever-present challenge, countless works in the NLP literature have considered tackling ambiguity. In their survey [60] Bhattacharyya presents a systematic characterization of ambiguity sources at all steps of the NLP pipeline. It then focuses on the specific problem of Word sense disambiguation, previously seen in Section 2.4.1.

One of the first works to directly tackle natural language ambiguity specifically for the case of business process descriptions was performed by van der Aa et. al. in [38]. More specifically, this work focused on *behavioural ambiguity*, that is, ambiguous statements about the control flow of a business processes. In it, the authors claim that some commonly used natural language statements are inherently ambiguous with regards to their semantics, particularly its scope. The concept of *interpretation* presented in [38] is also one of the main fundamental pillars of ATDP, presented as part of this thesis later on in Chapter 4, Section 4.1.

3.2 End-to-end systems for NLP

Recent developments in the NLP literature have been shifting the focus from the procedural pipeline described in Section 2.4.1 towards full data-driven methods that blur the line between the different phases of language understanding. In this thesis, we englobe this new developments in the NLP field under the umbrella term of *end-to-end NLP methods*, in contraposition to the *classical NLP methods*.

Typically, end-to-end systems are based on the idea of having a single, complex statistical model of the language to run all the tasks in a single pass: From tagging and parsing to semantic inference. On one hand, this makes the complete process less explainable, since models trained for this task—Typically variants of recurrent neural networks—are implicit models that act as black boxes [61]. On the other hand, by not having barriers between the different pipeline steps, the system has the capability of correcting previous assumptions from prior steps. For example, in a classical pipeline, an error during the parsing step can lead to subsequent issues during the semantic role labelling phase. In principle, by blurring the line between those two parts, the system can be more robust by mimicking a kind of iterative back-tracking process more similar to the kind of cognitive effort done in real language understanding.

End to end systems systems are based on in prior ideas like word2vec [62] and ELMo [63], where a single neural network is trained on a large corpus of text to compute *word embeddings*, *i.e.* numerical vectors that encode the semantics of words, based on the weights of the final layers of the network itself. Unlike the aforementioned systems, more recent methods like BERT [64] propose going beyond embeddings, and have started training large language models to perform the complete pipeline of language understanding tasks.

In this thesis, we do not propose techniques that take advantage of end-to-end NLP systems. That is because, unlike classical systems, the newly proposed end-to-end systems have the disadvantage of requiring large amounts of tagged training data to perform specific tasks. In some preliminary experiments, even when using techniques such as *transfer learning*, the amount of data required turned out to be a problem: For the kind

of classification tasks necessary to understand textual descriptions of business process models, there is no freely available data repository with enough high quality textual descriptions of business processes to train such systems. Later on, in section 4.2 we describe some efforts that were made to that end as part of this thesis.

3.3 Formal reasoning on top of process models

Formal reasoning on top of business process descriptions is a widely researched topic in the literature. The main goal of this kind of analysis is to allow for the creation of intelligent process management systems that are able to automatically assess the correctness and efficiency of business processes.

There have been several attempts in the literature at classifying process modelling notations. One such example is the previously stated survey by Lu et. al. [3], where a clear distinction is made between graph-based and rule-based notations, mainly focusing on the expressive power of each formalism. In a later work [65], S. Morimoto provides an alternative classification with respect to reasoning capabilities, where a distinction is made between automata-based methods, petri-net methods and process algebras.

One typical use case of *automata-based* reasoning are notations based on temporal logics—which have an equivalence to finite state automata—. Many of the efforts in creating LTL-based process modelling notations have their origin in Declare [66], previously discussed in Section 2.2, and from which the ATDP language presented in 4.1 draws its inspiration.

An example of work leveraging the formal roots of Declare to provide reasoning on top of process models is the work by M. Maggi et. al. In their paper [67], a runtime verification framework is introduced. Based on LTL and colored automata, the system is capable of performing compliance checking in real-time. The technique is shown to be robust to constraint validations, providing meaningful diagnostics even in those cases. The authors specifically reference Declare in their contribution, providing an encoding of the Declare LTL semantics into finite state automata, and showing how the technique has direct applicability for process models described in this declarative style.

Other works in the literature have leveraged similar techniques for graph-based notations, such as BPMN. In [68], Arzac et. al. present a model checking technique on top of BPMN, with a focus on user friendliness allowing validation security properties for deployments in a BPM execution engine. The authors show a proof of concept implementation based on the NetWeaver (NW) BPM industrial environment. Similar work in this line of research has also been introduced in other works [69].

In this thesis, we propose a set of formalisms and techniques with the focus of bringing the same kind of innovations in formal reasoning over process models to textual descriptions. This is covered in detail when discussing the ATDP language, later on in 4.1.

3.4 Educational systems for modelling

In Chapter 4, Section 4.3 an e-learning framework for process modelling—The *Model-Judge*—is presented. There exist very few works in the literature that jointly propose such a framework, and exploit NLP capabilities for automating the validation of created

process models. Still, we can find many examples of learning frameworks designed to aid students in conceptual modelling.

One such example is the one presented by Pimentel et. al. in [70]. In this work, a gamification setup is defined to engage students in the task of creating i^* models. The contribution is presented as a learning framework. However, it relies on manual effort and offers little automation support.

There have also been prior efforts to design automatic e-learning systems for other modelling notations, such as the work by Demuth et. al. [71], focused on UML diagrams instead of process models. In this system, shallow textual validation for the main model components is done, which requires establishing all the possible solutions for a given exercise *a priori*.

There is related work that explores the human behavior in modelling, an aspect that influences the work in *ModelJudge*. In [72], Störrle et. al. present a study to evaluate how human modellers visually parse UML diagrams using eye-tracking technology. A more recent study by Burattin et. al. [73] follows a similar approach for the case of reading BPMN process models during a modelling session. Incorporating insights arising from these studies in form of guidelines can be used to automatically assess the readability of a model.

The framework proposed in *ModelJudge* is inspired on code judges for programming, which are extremely successful on guiding novice programmers towards obtaining solutions to programming. One of the earliest examples of such framework is the work introduced by Kurnia et. al. in [74]. In it, a tool capable of automatically grading students' programming assignments—the *Online Judge*—is described. The authors argue that a tool like *Online Judge* is beneficial for the correction of programming exercises, freeing the instructors from repetitive work, avoiding biases towards a particular solution and solving the potential security issues of running untrusted programs on personal machines.

Later works improved on the ideas introduced in [74] by providing better test case functionality [75] or focusing on providing more detailed diagnostic feedback to students [76, 77]. For the *ModelJudge*, we considered the advantages of these innovations when proposing our technique for a different problem domain. Particularly, in *ModelJudge* we shift the focus from automatic grading to qualitative feedback in the form of detailed diagnostics, enabling the modeler to easily localize errors.

Chapter 4

Main Contributions

In this section, we introduce an overview of the four main contributions that conform this thesis, briefly introduced already in 1.3. First, Section 4.1 describes the design and formalization work carried out to define the ATDP language. Next, Section 4.2 introduces a technique for automatic alignment of structured and unstructured sources of process documentation. Later on, Section 4.3 presents the work in the development of *Model-Judge*: An educational platform to aid in teaching process modeling. Finally, Section 4.4 presents a technique for the automatic generation of autonomous conversational agents from business process descriptions.

The reader should note that the order of the contributions shown in this chapter does not correspond to the original chronological order of publication. Instead, the order has been chosen to present the publications in the recommended reading order: Ranging from the abstract and theoretical to more concrete, practical works.

4.1 Formal Reasoning on Natural Language Descriptions of Processes

4.1.1 Related publications

- IV. Josep Sànchez-Ferreres, Andrea Burattin, Josep Carmona, Marco Montali and Lluís Padró. “Formal Reasoning on Natural Language Descriptions of Processes”. In: *Proceeding of the 17th International Conference on Business Process Management (BPM). 2019*. (GGs Rating: A)
- VI. Josep Sànchez-Ferreres, Andrea Burattin, Josep Carmona, Marco Montali, Lluís Padró, Luís Quishpi, “Unleashing Textual Descriptions of Business Processes”. Submitted: *International Journal on Software and Systems Modeling* (Impact Factor: 1.915, Q1)

4.1.2 Main highlights

Process specifications are the usual way of communicating how and in which order process activities should be executed. To that end, several formal languages have been

introduced over the years to allow for unambiguous formal process specifications. However, process specifications are often provided using simple, easy to understand natural language. This helps all the related roles understand a business process without having to rely on teaching formal, rigid notations to non-technical actors.

Despite their better accessibility, natural language descriptions of processes have several issues that make them unsuitable to be an appropriate language for formal specifications: Unstructured text does not support automation nor easy extraction of information. Moreover, it is oftentimes inherently ambiguous and usually written in a subjective style, which may lead to multiple interpretations, given the same written specification.

Taking on these challenges, this contribution proposes *Annotated Textual Descriptions of Processes* (ATDP). Both a language, and an associated framework that serves as the first step towards bringing informal textual descriptions and formal specifications together under the same formalism. The core concept in this framework is the ATDP specification, which is shown in its visual form in Figure 4.1

In publication IV, we introduce the core ATDP formalism. This work was later extended in a later extension—publication VI—, where more details about the core formalism were provided, as well as presenting multiple real use cases of the ATDP framework.

More specifically, this line of research produced the following specific contributions to the field:

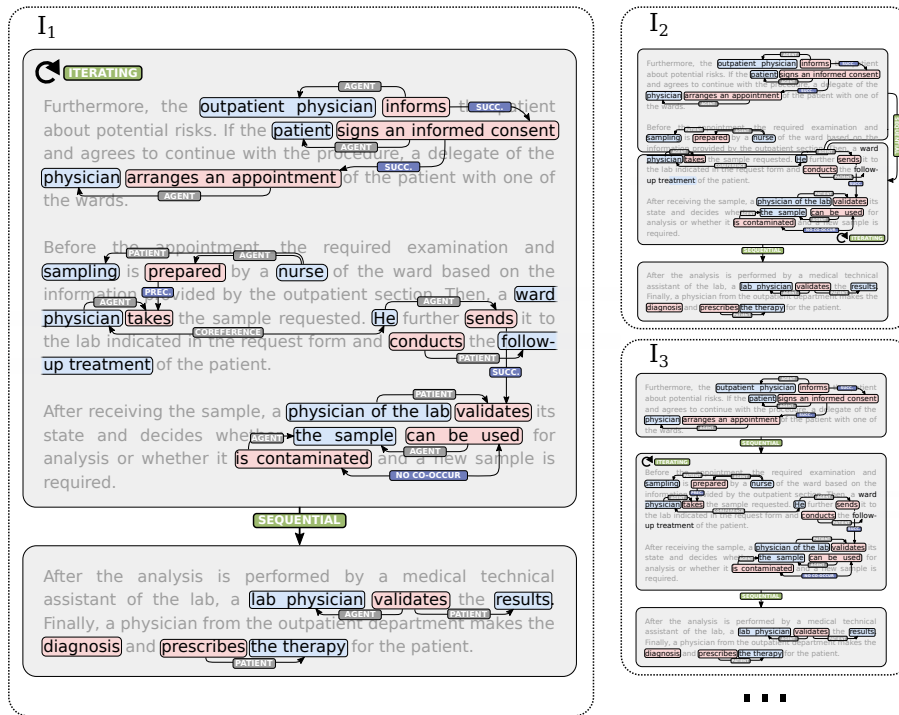


Figure 4.1: Visual representation of an ATDP specification.

1. Formalizing Natural Language Descriptions

As previously seen in Chapter 3, there have been multiple prior efforts in the literature to deal with textual descriptions of processes. Most prior attempts,

however, relied on introducing some *ad-hoc* formalization of the problem tailored to the specific task the authors were attempting to solve.

One of the contributions presented in this work is the creation of a common framework on top of which other researchers can build several tools that work on the relation between formal and informal process descriptions. By embracing the ATDP framework, further research can focus on the specifics of how to solve a certain problem, avoiding repetition in having to re-define the same formal framework multiple times.

2. An end-to-end framework with tool support

The work in ATDP presents an end to end framework: From the formal specification of a new modelling notation, to the implementation of several techniques enabling automation on top of it. Figure 4.2 overviews the current main contributions in the ATDP ecosystem:

- The ATDP extractor, presented by Quishpi et. al. in [48]—outside the work of this thesis—, introduces a tool to automatically bootstrap ATDP specifications out of plain text descriptions of process models.
- The ATDP reasoner, introduced in Publication VI, is a tool to automatically encode ATDP specifications into NuSMV model checking instances. By leveraging a standard model checker, arbitrary queries can be presented and resolved on top of an ATDP specification, which, as shown by the contribution in [48], is partially obtainable from a plain textual description.
- The ATDP simulator, also introduced in Publication VI, leverages two well-known simulation techniques for process trees and LTL automata to automatically generate event logs out of ATDP specifications. The tool offers a fine degree of control over the generated traces and can be useful to validate event-trace-based techniques from the field of *process mining*.

4.2 shows

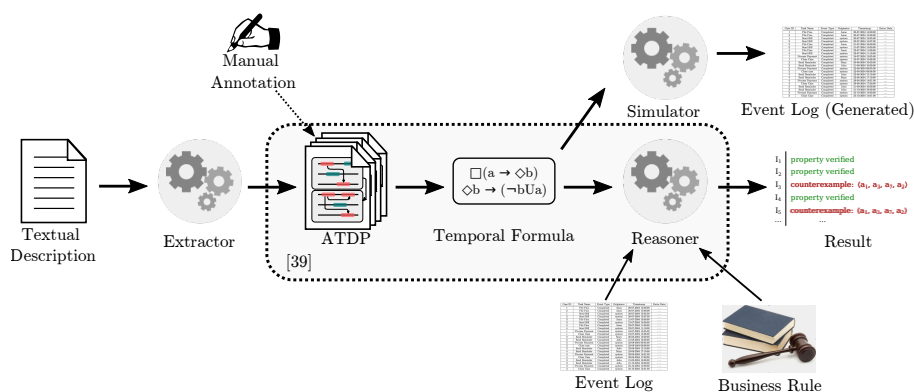


Figure 4.2: An overview of the ATDP framework.

3. A notation that is ready for machine learning

Being based on textual annotations, ATDP are well suited to be used not only as formal documentation, but also as an annotated training corpora.

By agreeing on a common annotation language and producing annotations, modelers do not only get the benefits of being able to formally document their textual descriptions, but are also producing annotations that are ready to be used by researchers to improve automatic extraction and generation of such models, making the modeler's job more easily automated over time.

Such annotated corpora can be of great use in the training of *state-of-the-art* systems for NLP, as discussed in Chapter 3, Section 3.2.

4. Tackling ambiguity

As mentioned in 3.1.4, ambiguity is one of the most important challenges towards making textual descriptions a first-class language for process specification.

In this contribution, we attempt to bridge this gap by proposing a system that is not only aware of ambiguity, but is theoretically capable of formal reasoning even in the presence of ambiguity. We do so by introducing the concept of a *interpretations*, represented as I_1 , I_2 , I_3 in Figure 4.1

5. Characterizing how textual descriptions are written

Another important contribution in this work are the logical relations in the ATDP language. By abstracting over the low-level details of temporal logics and process trees, ATDP provides a set of high-level constructs that are inspired in the way human modelers typically write textual descriptions of processes.

This set of relations, while still being open to further improvements, is a good first step to encapsulate and characterize how people describe processes textually, and serves as a good starting point for further research that aims to automatically extract this kind of information from textual descriptions

6. A hybrid process modelling language

To the best of our knowledge, ATDP is the first process modelling formalism to offer a mix of declarative and procedural styles. It was observed very early on in the design process that textual descriptions would often not fit into either style: Sometimes using very clear procedural sequences of steps—e.g. “First, do X. Next, Y is performed. Finally, Z happens”. While other kinds of requirements are more naturally stated in a declarative style—e.g. “X must occur within 5 minutes of Y”. By being able to mix the two representations, ATDP is able to adapt to a wider range of textual patterns.

4.2 Aligning textual and graphical descriptions of processes

4.2.1 Related publications

- I. Josep Sànchez-Ferrerres, Han van der Aa, Josep Carmona and Lluís Padró. “Aligning textual and model-based process descriptions”. In: *Data Knowledge Engineering* 118 (2018), pp. 25-40. (Impact Factor: 1.583, Q3)
- III. Josep Sànchez-Ferrerres, Josep Carmona and Lluís Padró. “Aligning Textual and Graphical Descriptions of Processes Through ILP Techniques”. In: *Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CaiSE)*. 2017. pp. 413-427. (GGS Rating: A)

4.2.2 Main highlights

Accessibility for business process documentation is paramount when adopting BPM practices in any organization. Often, to achieve this, it is desirable to maintain an easy-to-understand informal textual description alongside an automation-enabled process description language. This is a very clear use case for notations like ATDP, where the formal and unstructured aspects are aligned in a single artifact.

However, nowadays BPMN is still by far the most widespread modelling standard. This is why, when faced with this challenge, organizations usually choose to maintain a textual document describing each BPMN process diagram in their repository. There are obvious drawbacks to choosing this dual representation: Maintenance and improvement tasks performed to one side are often not fully translated to the other, making the informal documentation drift away from the formal specification that is actually being executed underneath. This, can be the source of potential losses and can also lead to lack of compliance with regulations.

With the aim of improving the situation of keeping multiple separate documentation synchronized, in this contribution we presented a technique to compute optimal alignments between business process descriptions and BPMN business process model specifications. The aim of the tool is to automatically compute a text-to-process alignment like the one shown in Figure 4.3

The reader should note that this contribution was developed prior to the formalization of ATDP introduced in the previous section. This is why, despite the obvious fit, there is no mention to ATDP in the publications of this section. By adopting a text-friendly notation like ATDP, text-to-process alignment is done by the modeller with relatively low effort and in a much more controlled way.

Publication III of this thesis is presented as an iteration of the work introduced by van der Aa et. al. in [78]. The other work in this line, Publication I, is a joint journal extension where the core ideas behind this and the work of [78] were merged, while improving the range of accepted BPMN items.

Next, we detail each of the specific contribution points introduced in this research:

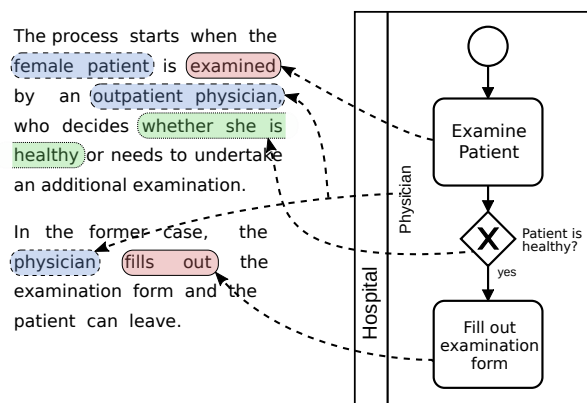


Figure 4.3: Text to business process alignment example.

1. A faster technique with *state-of-the-art* accuracy

By casting the problem as an Integer Linear Programming (ILP) problem instance, the technique presented in this publication is able to perform the same optimal

alignment computation as the *state-of-the-art* technique in a fraction of the time. In some instances, the execution time of the technique is brought down from an order of magnitude of seconds to an order of milliseconds.

This execution time improvement enables using the technique in real time where before only batch processing was possible, in turn enabling new use cases for the technology. One of such use cases is the *ModelJudge*, described in detail in later on in

2. A new benchmark for process to text comparison

As previously discussed in 3.2, one of the largest impeding factors in exploring the potential of *state-of-the-art* machine learning systems for the kind of language comprehension tasks presented in this work is the lack of high quality annotated training data that can be used to train and validate such systems.

In order to contribute to this, the work presented in Publication I also includes the manual collection, curation and annotation of a completely new dataset for model to text alignment evaluation. This dataset is used in the evaluation and has also been freely distributed¹ to foster future research in the field.

3. Wider coverage of BPMN constructs

In BPMN, there is a wide range of constructs that define not only the behavioural relations between activities, but also other perspectives of business process execution.

Thus, when comparing a textual description a rich business process notation such as BPMN, it is important to consider all the available information in the alignments.

This work is, to the best of our knowledge, the first one to introduce an alignment technique that is able to consider alignments for the most important BPMN constructs: Activities, Events and Gateways, as well as Swimlanes and Pools. Furthermore, results show that this newly proposed technique is not only able to align more elements than prior approaches, but also benefits from this additional information by offering better activity detection accuracy.

4.3 *ModelJudge*: Process modelling in education

4.3.1 Related publications

- II. Josep Sànchez-Ferreres, Luis Delicado, Amine Abbad Andaloussi, Andrea Burrattin, Guillermo Calderón-Ruiz, Barbara Weber, Josep Carmona and Lluís Padró. “Supporting the Process of Learning and Teaching Process Modelling”. In: *Transactions on Learning Technologies (TLT)*. 2019. (Impact Factor: 2,315, Q2)
- VII. Luis Delicado, Josep Sànchez-Ferreres, Josep Carmona and Lluís Padró. “The *ModelJudge* - A Tool for Supporting Novices in Learning Process Modelling”. In: *BPM 2018 Demonstration Track*. 2018.

¹https://github.com/setzer22/alignment_model_text/tree/master/datasets

4.3.2 Main highlights

The creation of a process model is primarily a formalization task, that faces the difficulty of constructing a syntactically correct entity which accurately reflects the semantics of a real process while remaining readable and understandable. This sort of task often poses a challenge to novice modellers, who often struggle with syntactic, pragmatic and semantic aspects of process modelling.

A typical process modelling course will, at some point, feature a very specific kind of exercise: Given a textual description explaining a business process, the goal is to generate the corresponding diagram by including all the information from the text, and without over-specifying any of the requirements.

With the aim of improving the process of both learning and teaching process modelling, we developed *ModelJudge*. A tool that is able to provide accurate diagnostics during a student's modelling session, by continuously comparing their work to the exercise statement. That is, given a textual description of a business process, *ModelJudge* helps students propose a business process model using the BPMN standard notation. At any point during this process, students can request detailed feedback, which is automatically reported by *ModelJudge* without teacher intervention. Figure 4.4, shows the modelling interface of the tool.

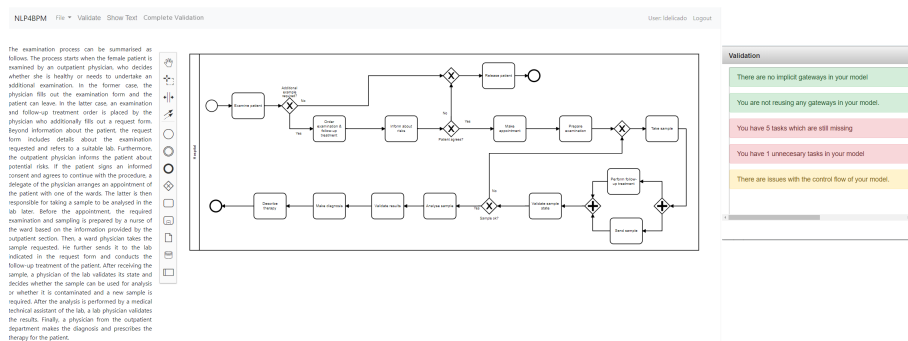


Figure 4.4: The ModelJudge student view.

Next, we detail the specific points of interest of this line of research.

1. A direct application of text to model alignment algorithms

At its core, the *ModelJudge* has an algorithm to constantly compare the student's proposed model to a reference textual description, *i.e.* the exercise statement. To achieve this, we developed a modified version of the algorithm previously introduced in 4.2.

The new alignment algorithm uses the same underlying principles as the one from 4.2, but differs in two key points.

- While the original technique works without placing trust on either side, text or model, this new technique places full trust on the textual description, which in *ModelJudge*, is the exercise statement. This improves feedback quality helping generate more meaningful error messages.
- The algorithm has been adapted to compute *partial* alignments. This was necessary because in *ModelJudge* students are able to request for feedback at

any point in the modelling session, even when the BPMN does not include all the models in the text.

2. A use case for ATDP annotations

The unique setup of the *ModelJudge* presents an opportunity to improve the accuracy of the underlying AI by means of *manual refinement*: A teacher can spend the effort of curating the results of the NLP analyzer for a single exercise once, all of their students will benefit from that effort.

This situation is a perfect fit for the ATDP framework presented in 4.1. The workflow of creating an exercise in *ModelJudge* is thus as follows:

- (a) First, the instructor uploads a text from *ModelJudge*'s teacher zone
- (b) At this point, the instructor is presented with an editor, portrayed in Figure 4.5, and is also prompted to request bootstrapping an initial automatic annotation.
- (c) Using the technique presented in [48], an initial annotation is computed and presented to the teacher.
- (d) Finally, the teacher can choose to keep the automatic annotation as-is, or introduce manual refinements to improve the accuracy. Examples of such manual refinements include removing activity annotations from the text that do not correspond to actual activities in the process or manually resolving some coreferences that require specific domain knowledge.

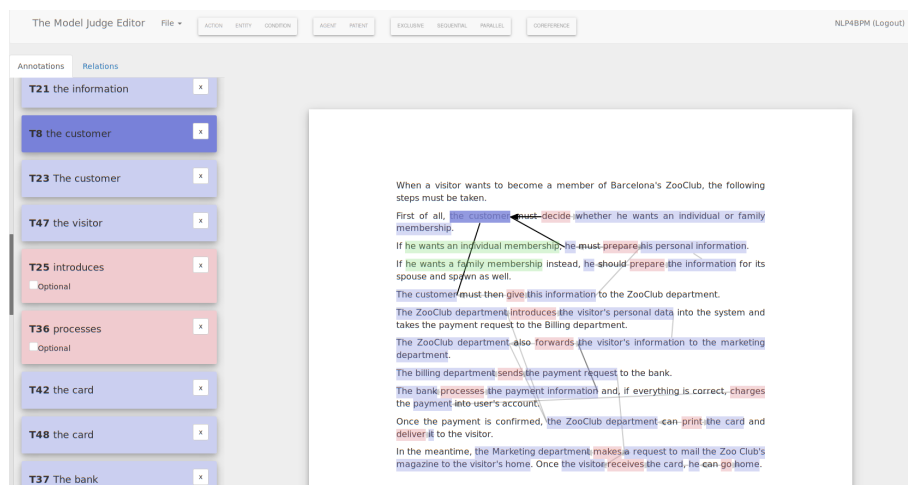


Figure 4.5: The ModelJudge exercise editor.

3. A robust end-product used by hundreds of students

ModelJudge is not only a theoretical exercise: It has been developed and deployed in a production environment, and currently has thousands of registered users from multiple universities with courses focused on process modelling. Some of the early adopters of *ModelJudge* include the Danmarks Tekniske Universitet (DTU), Universidad Catolica de Santa María (UCSM), Katholieke Universiteit Leuven (KU Leuven), the AGH University of Science and Technology (AGH) and the Universitat Politècnica de Catalunya (UPC).

ModelJudge is currently available at <https://modeljudge.cs.upc.edu>. Students accounts can be freely registered. To obtain an instructor account, course organizers must contact modeljudge@cs.upc.edu.

4. Empirically Validated

In Publication II, we study in detail empirical data of how *ModelJudge* is used in two university courses from DTU and UCSM. On one hand, a qualitative study has been performed, which finds that the majority of students consider *ModelJudge*'s feedback useful during a modelling session. On the other hand, a quantitative study has also been performed by analyzing usage data that was automatically recorded during modelling sessions with student consent.

The same kind of information collected during the quantitative study is also exposed in a real-time dashboard, usable by instructors to monitor their student's progress during the modelling session. A screenshot of this dashboard page is shown in Figure 4.6

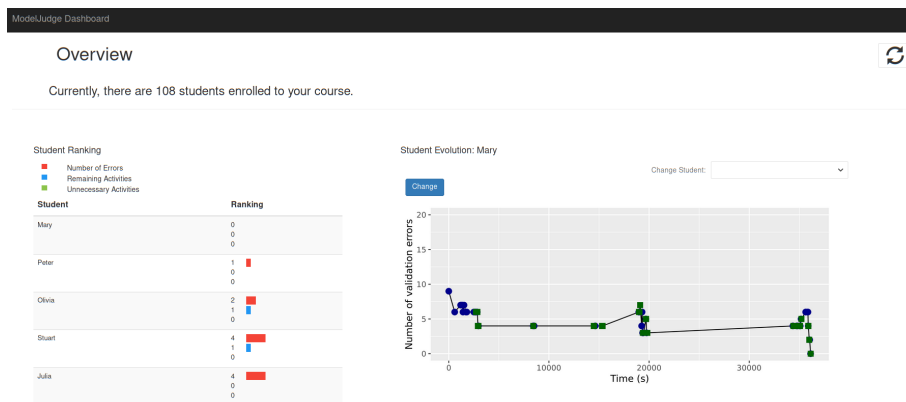


Figure 4.6: The ModelJudge teacher dashboard.

5. Avoiding representational bias

When compared to existing solutions in the literature, as seen in Section 3.4, *ModelJudge* introduces a framework that avoids the issue of *representational bias*. Existing solutions for automatic assessment of modelling exercises, like the work introduced in [71], often require pre-establishing one or several solutions in the same modelling notation as the exercise target. For instance, in such systems, for a BPMN modelling exercise the expected solution is encoded as a BPMN diagram. This leads to rigid solutions that do not handle the situation where student solutions do not conform to the expected diagram while still being correct. This issue is one of the main motivating factors in the design of automatic programming exercise grading tools like the *Online Judge* [74], from which *ModelJudge* is based on.

Being based in textual descriptions, *ModelJudge* analyzes the conformance of the student's model directly against the problem statement, avoiding the potential bias of having to translate to a more rigid notation for evaluation. This avoids the aforementioned drawbacks of tools like [71],

4.4 Conversational process models

4.4.1 Related publications

- V. Anselmo López, Josep Sànchez-Ferreres, Josep Carmona and Lluís Padró. “From Process Models to Chatbots”. In: *Proceedings of the 31th International Conference on Advanced Information Systems Engineering (CAiSE)*. 2019. pp. 383-398. (GGS Rating: A)

4.4.2 Main highlights

Formal process modelling notations like BPMN are ubiquitous in organizations. However, as previously discussed, there is a gap between the technical language used to generate and consume this formal representations, and the non-technical users that are sometimes involved in the execution of such processes.

In order to achieve full digital transformation, processes need not only to be automated, but also be well documented and understood by all the relevant actors. In order to help improve comprehension of business processes, in this work, we present a technique capable of automatically generating a conversational agent able to explain the most important aspects of a business process to a non-technical user via a chat interface. Figure 4.7 shows an example of a conversation with one of the generated assistants.

The following specific contributions were made as part of this line of work:

Bot: Hi, which process do you want to see?
You: I'm interested in the procure parts process.
Bot: The process begins when the parts are required by the Department.
What do you want to do now?
You: Who checks the purchase order?
Bot: The Central Purchasing checks the purchase order.
You: Follow process.
Bot: Parts out of stock/Parts in stock ?
You: Parts out of stock.
Bot: The Department specifies and quantifies the requirements.
You: Next.
Bot: The Department completes the purchase order.
You: Next.
Bot: The Central Purchasing checks the purchase order. Vendor known?
You: Yes.
Bot: The Central Purchasing creates the order template from purchase order.
You: Who did you say before that checks the purchase order?
Bot: The Central Purchasing checks the purchase order.
You: ok.
Bot: The Approver approves the purchase order.
You: Next.
Bot: After that, the Central Purchasing executes process called “receipt of goods”.
You: Next.
Bot: Then, the process is finished.

Figure 4.7: Example dialog with the conversational assistant.

1. An iteration on existing process to text conversion techniques

At the core of the chatbot application, is a system that can explain parts of a process in plain natural language. For this, we have implemented a technique

based on the same ideas and principles as the one introduced by Leopold et. al. in [49]. Notably, the technique features two key differences with respect to the original approach:

- In order to generate more accurate descriptions of task labels, a new custom parser has been developed from scratch, able to extract the semantic components of activity, swimlane, event and role labels. Unlike previous approaches, this new parser is based on context-free grammars, and is more easily adaptable to several patterns and languages.
- By considering the new chatbot use case, some adaptations have been made to the generated texts. For instance, exclusive gateways representing a decision are rephrased as questions to the user, whereas parallel sections are not described as such, but rather by presenting a choice of what step to perform next to the user.

2. Further accessibility opportunities

BPMN is, by nature, a complex visual standard. The research introduced in this work opens the door to improved accessibility in companies embracing digital transformation. Being able to query a conversational for different aspects of a process model not only allows people without deep understanding of BPMN to better understand their role in a process, but also allows people with different kinds of visual or motor disabilities to take part in the digital transformation of companies. This is an aspect that is often overlooked when discussing the benefits and drawbacks of purely visual notations like BPMN.

Improving the techniques presented in Publication V with a direct focus on accessibility is planned as a follow-up work to this thesis.

3. Empirical validation

The technique has been empirically validated using a qualitative evaluation. We collected detailed feedback from 33 individuals, both from academic and industrial background. Overall, there is a consensus in that there is huge potential to this technique. However, most interviewed individuals agree there is room for improvement in the current implementation.

The following five questions were collected in the survey. The first three asked for a direct numerical evaluation, with the results portrayed in Figure 4.8. The latter two questions, asking for general feedback, were an interesting source of ideas for improvement but also confirmed the limits of the current implementation.

Q1: *How was your interaction with the chatbot?* (1: not fluent – 4: very fluent)

Q2: *Did the Process Model Chatbot answer your questions about the process?* (1: it did not – 4: it did)

Q3: *Do you see potential for this kind of application in organizations?* (1: no potential – 4: large potential)

Q4: *What did you like / dislike about the tool?*

Q5: *Do you have any suggestions in order to improve the Process Model Chatbot?*

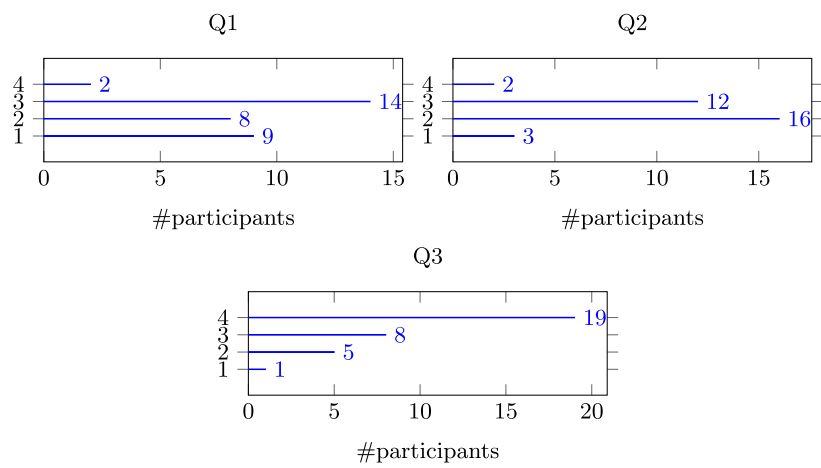


Figure 4.8: Results of questions Q1 to Q3.

Chapter 5

Conclusions and Future Work

Natural language is an unavoidable reality, even in the most formal business process documentation formalisms to this date. Thus, techniques enabling exploitation of these unstructured data sources open the door to great benefits in companies and organizations adopting digital transformation practices. In this final chapter we summarize the main results obtained as part of this thesis and as discuss promising future work directions in the area.

5.1 Summary

The first main contribution in this thesis has been the introduction of the ATDP language. The design principles behind ATDP, while introduced first in this document, are the end result of all of our previous research: It was only after having carefully considered several use cases for natural language technology in the field of BPM that the need for a more text-friendly way of annotating and reasoning over textual descriptions of processes was made evident. We believe that by embracing the same annotation standard, future research can avoid unnecessary repetition of *ad-hoc* formalization work, increasing compatibility and reproducibility between future work embracing this new formalism.

Closely following the same principles introduced in ATDP, and with the goal of allowing a mixture of textual descriptions and business processes in organizations, three more main contributions have been developed, each with a full open sourced implementation and a robust empirical validation.

The next main contribution is a technique allowing computations of an alignment between a fully unstructured textual description and its counterpart process model. This opens the door to several use cases: From improving feedback during a modelling session—as explored later on in *ModelJudge*—to the more difficult challenge of creating autonomous systems capable of finding document matches from large process model repositories.

In the case of *ModelJudge*, our third main contribution, not only a full implementation has been provided as a proof of concept, but the platform itself has been deployed and is now currently being used by hundreds of students each year in academic courses. The *ModelJudge* platform was developed as a direct application of the ideas explored in the implementation the previously discussed alignment algorithm. By seeking to im-

prove the accuracy of existing NLP tools for this new particular use case, we developed an initial version of many of the ideas that later became ATDP.

Finally, the fourth main contribution is a technique that can autonomously generate conversational agents capable of explaining business processes. While the approach has currently shown great potential, we are also aware that there is large room for improvement in the consistency of the generated results. We consider this to be a very interesting topic for future work. In a more broad view, we believe exploring the idea of conversational agents not only for explaining, but also for *modelling* business processes is something that can help lower the cognitive bar, and approach even a technical topic like process modelling to non-technical profiles.

Overall, it is clear that the problems at intersection of human language analysis and business process management, both human-centric fields are yet far from being fully solved. Nevertheless, we believe this thesis has indeed contributed in closing this large gap between the world of human-centric textual information and the one of computer-centric process documentation. Furthermore, we believe the work we introduced opens the door to a large number of both challenges and opportunities that are yet to be explored.

5.2 Future Work

The work introduced in this thesis sets the basis for multiple promising lines of future work. We believe there is still room for improvement in all of the presented works. Additionally, there are several ongoing projects that, while developed during the span of this thesis, have not yet produced enough results to be included in a publication. Finally we also consider interesting future work that still unexplored in the literature.

1. **Improving the formalization of the ATDP language.** We observed large potential for improvement in the ATDP language itself. In fact, we do not consider our initially proposed specification closed, quite the opposite: We want ATDP to become a more widespread standard that is able to support a wide range of use cases in the intersection between human natural language and BPM practices. Thus, we are open for exploring new improvements to the language, formalization or overall framework.
2. **Tackling large model repositories.** Organizations maintaining large process model portals and repositories can benefit from the work introduced in this thesis. By natural language analysis techniques, a new level of automation can be built on top of existing BPM technology.

An interesting opportunity for future work in this regard, is an expansion of our technique for the automatic generation of conversational agents. The generated agents would be expanded to allow querying information of complete model repositories, instead of single process diagrams. When enhanced by a formal back-end as ATDP this new approach would allow answering complex organization-wide queries like “What are the most frequently performed tasks done by sales managers?” or “How often do sales and marketing departments communicate?” via translation to logical propositions.

The alignment technique presented on this thesis, can also benefit from a similar extension by considering complete process repositories. A system based on this

technique can be developed in order to find matches between textual documentation—such as rules and regulations documents—, and processes in an organization’s repository.

3. **Bringing new developments in the NLP community to BPM.** A very interesting use case we are currently considering is using *state-of-the-art* end-to-end machine learning systems for NLP, such as Bert [64], to automatically extract process-related information from business process textual descriptions.

However, after some preliminary experiments it was determined that the amount and quality of freely available training data for this kind of task was not sufficient to improve current results using classical techniques. Nonetheless, we believe this to be a very promising line of future work

4. **Exploring the automation of training data extraction.** As just discussed in our previous point. Lack of training data is one of the main drawbacks behind the adoption of new developments from the NLP community to the field of BPM. Part of the work in this thesis consisted of manual collection and annotation of examples, which have been used in the development of the presented contributions, and have been made available to the community.

However, the process of collecting and annotating this training data is both slow and error prone. Currently, we are exploring the idea of using techniques such as *bootstrapping* in order to obtain a large dataset of textual data describing processes. A system combining *web scrapping* and *machine learning classification* can be used in conjunction with the algorithmic techniques introduced in this thesis to aid in the work of obtaining a large amount of unlabeled textual data, one of the first steps towards the creation of a reference training corpus specific for BPM.

5. **Improvements to the proposed alignment technique.** The text to process alignment technique introduced in this thesis has shown potential and been empirically validated with good results. However, we believe there are future improvements that would both widen the scope and improve the existing results of the technique.

One important fact to consider is the *granularity of alignments*. To this date, all text to model alignment techniques have worked by aligning text sentences to process model activities. By reducing this granularity to that of *predicates*, more robust and meaningful alignments could be obtained in cases where a single sentence describes multiple process model activities: A phenomena that we have been able to observe in many texts, in and outside our validation benchmark.

Another possible improvement to the alignment technique would consist of widening the scope of supported constructs. Currently, the technique presented is capable of dealing with activities, gateways events and swimlanes. There are more constructs to be considered, both in the data perspective of business processes—e.g. data object references—and in the semantics of currently supported types—e.g. extract and consider even type information and its semantics.— We believe this to be a promising line of future research.

6. **Exploring statistical biases in simulation of process models.** The work we introduced in the ATDP simulator uncovered an interesting fact about current process model simulation techniques: We observed very early on that the distribution of trace interleavings when simulating a parallel construct, such as a BPMN parallel gateway, is *non-uniform*. That is, there is a bias towards certain event interleavings over others in the generated traces.

After some preliminary exploration, it has been determined that these biases exist, and are different depending on which kind of simulation technique is used. With different *state-of-the-art* tools producing significantly different results.

Simulation is widely employed in the industry in the context of BPM analytics, and is closely related to the validation of techniques in the field of *process mining*. Thus, we believe a study characterizing the different kinds of biases and relating them to the real world phenomena being studied could be beneficial, and poses a topic of future work. This future contribution is something we are currently actively exploring.

- 7. Explore more robust NLU techniques for chatbot generation.** The currently presented technique for chatbot generation is based on the AIML specification. Currently, this specification does not allow a flexible interpretation of input sentences, offering only a system equivalent to regular expressions for input parsing. This limitation leads to very rigid conversational agents.

A possible improvement to this work would consist of using newer techniques based on machine learning typically used in NLU. One possibility is to enhance the same kind of generated conversation graphs with semantic similarity metrics like the ones provided by *word embeddings*.

- 8. Improve diagnostic accuracy for *ModelJudge*.** While the current quality of diagnostics has been validated with real usage in modelling courses, there is still large room to continue exploring and improving the existing techniques in order to provide more detailed model quality diagnostics to students.

One possibility is to explore better alignment techniques for the specific use case of grading partial models. The current technique in use has the main drawback of relying on *hard constraints* for control flow validation. However, this can lead to lower quality alignments when students' control flow vastly differs from the one of the reference text.

Recently, we have explored the idea of relaxing the constraints in the optimization problem. We believe using a technique capable of encoding *soft constraints* will make it possible to provide more meaningful and accurate diagnostics about the control flow of the process model.

- 9. Further empirical work in interaction during process modelling.** The work introduced in *ModelJudge* opens the door to new interesting lines of empirical research about the process of process modelling. With advanced analytics functionalities fully integrated in to the platform, *ModelJudge* provides a good basis to better explore how students, or even experienced modelers interact with their modelling tools.

- 10. Exploring the discursive creation of process models.** A different use case for conversational agents in process modelling is to allow for *modelling* instead of *querying* business processes. We believe such a tool can help lower the cognitive effort required during the modelling of a process.

This is a line of work we are very keen on exploring, which has currently materialized in *Process Talks*¹: A new technology based on the central idea of delegating the technical parts of process modelling to a reasoning engine which is capable of understanding human language and converting it to high level modifications to a process model such as “The block conformed by activities A and B forms a loop” or “All activities from A to B form a subprocess”.

¹<https://www.processtalks.com>

Chapter 6

List of Publications

This chapter briefly summarizes the list of publications that are presented as part of this article compendium. Roman numerals have been used to easily differentiate between the contributions that are part of this thesis, and other references from the bibliography.

Journals and Conference Proceedings

- **I.** Josep Sànchez-Ferreres, Han van der Aa, Josep Carmona and Lluís Padró. “Aligning textual and model-based process descriptions”. In: *Data Knowledge Engineering* 118 (2018), pp. 25-40. (Impact Factor: 1.583, Q3)
- **II.** Josep Sànchez-Ferreres, Luis Delicado, Amine Abbad Andaloussi, Andrea Burattin, Guillermo Calderón-Ruiz, Barbara Weber, Josep Carmona and Lluís Padró. “Supporting the Process of Learning and Teaching Process Modeling”. In: *Transactions on Learning Technologies (TLT)*. 2019. (Impact Factor: 2,315, Q2)
- **III.** Josep Sànchez-Ferreres, Josep Carmona and Lluís Padró. “Aligning Textual and Graphical Descriptions of Processes Through ILP Techniques”. In: *Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CaiSE)/*. 2017. pp. 413-427. (GGs Rating: A)
- **IV.** Josep Sànchez-Ferreres, Andrea Burattin, Josep Carmona, Marco Montali and Lluís Padró. “Formal Reasoning on Natural Language Descriptions of Processes”. In: *Proceeding of the 17th International Conference on Business Process Management (BPM)*. 2019. (GGs Rating: A)
- **V.** Anselmo López, Josep Sànchez-Ferreres, Josep Carmona and Lluís Padró. “From Process Models to Chatbots”. In: *Proceedings of the 31th International Conference on Advanced Information Systems Engineering (CAiSE)*. 2019. pp. 383-398. (GGs Rating: A)
- **VI.** Josep Sànchez-Ferreres, Andrea Burattin, Josep Carmona, Marco Montali, Lluís Padró, and Luís Quishpi, “Unleashing textual descriptions of business processes”. In: *Software and Systems Modeling*. 2021. pp. 1-23. (Impact Factor: 1.915, Q1)

Demos and Workshops

- **VII.** Luis Delicado, Josep Sànchez-Ferreres, Josep Carmona and Lluís Padró. "The Model Judge - A Tool for Supporting Novices in Learning Process Modeling". In: *BPM 2018 Demonstration Track*. 2018.

Bibliography

- [1] 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.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] 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).
- [6] 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.
- [7] 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).
- [8] 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).
- [9] Wil MP Van der Aalst. *Process mining: data science in action*. Springer, 2016.
- [10] 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.
- [11] Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. “Discovering Block-Structured Process Models from Event Logs - A Constructive Approach”. In: *Proceedings of the 34th International Conference on the Application and Theory of Petri Nets and Concurrency (PETRI NETS)*. Ed. by José Manuel Colom and Jörg Desel. Vol. 7927. Lecture Notes in Computer Science. Springer, 2013, pp. 311–329.

- [12] Oliver Kopp, Daniel Martin, Daniel Wutke, and Frank Leymann. “The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages”. In: *In: Ulrich Frank (ed.): Enterprise Modelling and Information Systems. Vol. 4(1), pp. 3-13* 4 (June 2009).
- [13] Artem Polyvyanyy, Jussi Vanhatalo, and Hagen Völzer. “Simplified computation and generalization of the refined process structure tree”. In: *International Workshop on Web Services and Formal Methods*. Springer. 2010, pp. 25–41.
- [14] Michele Chinosi and Alberto Trombetta. “BPMN: An introduction to the standard”. In: *Computer Standards & Interfaces* 34.1 (2012), pp. 124–134.
- [15] Wil MP Van Der Aalst and Arthur HM Ter Hofstede. “YAWL: yet another workflow language”. In: *Information systems* 30.4 (2005), pp. 245–275.
- [16] Robert Goldblatt. *Logics of Time and Computation*. 2nd ed. CSLI Lecture Notes 7. Stanford, CA: Center for the Study of Language and Information, 1992. ISBN: 978-0-937073-94-0.
- [17] Claudio Di Ciccio, Mario Luca Bernardi, Marta Cimitile, and Fabrizio Maria Maggi. “Generating Event Logs Through the Simulation of Declare Models”. In: *EOMAS 2015*. 2015, pp. 20–36. DOI: 10.1007/978-3-319-24626-0_2.
- [18] Amir Pnueli. “The Temporal Logic of Programs”. In: *IEEE*, 1977, pp. 46–57.
- [19] Giuseppe De Giacomo and Moshe Y. Vardi. “Linear Temporal Logic and Linear Dynamic Logic on Finite Traces”. In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*. IJCAI/AAAI, 2013, pp. 854–860.
- [20] Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. “Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness”. In: *Proceedings of the 28th AAAI Conference on Artificial Intelligence*. AAAI Press, 2014, pp. 1027–1033.
- [21] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. Jan. 2001. ISBN: 978-0-262-03270-4.
- [22] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. “NUSMV: A New Symbolic Model Checker”. In: *STTT* 2.4 (2000), pp. 410–425.
- [23] Roman V. Yampolskiy. “Turing Test as a Defining Feature of AI-Completeness”. In: (2012).
- [24] 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.
- [25] 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.
- [26] 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.
- [27] 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.
- [28] 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.

- [29] 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.
- [30] 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.
- [31] 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.
- [32] 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.
- [33] 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.
- [34] 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.
- [35] 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.
- [36] 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.
- [37] 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.
- [38] Han van der Aa, Henrik Leopold, and Hajo A Reijers. “Checking Process Compliance against Natural Language Specifications using Behavioral Spaces”. In: *Information Systems* (2018).
- [39] 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, pp. 2473–2479. URL: <http://www.lrec-conf.org/proceedings/lrec2012/summaries/430.html>.
- [40] Christiane Fellbaum. *WordNet. An Electronic Lexical Database*. Language, Speech, and Communication. The MIT Press, 1998.
- [41] 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.
- [42] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management, Second Edition*. Springer, 2018. ISBN: 978-3-662-56508-7. DOI: 10.1007/978-3-662-56509-4. URL: <https://doi.org/10.1007/978-3-662-56509-4>.

- [43] Remco Dijkman, Irene Vanderfeesten, and Hajo A. Reijers. “Business process architectures: overview, comparison and framework”. In: *Enterprise Information Systems* 10.2 (Feb. 2016), pp. 129–158. ISSN: 1751-7575. DOI: 10.1080/17517575.2014.928951.
- [44] J. Pinggera. “The Process of Process Modeling”. PhD thesis. University of Innsbruck, Department of Computer Science, 2014.
- [45] Jan Claes, Irene Vanderfeesten, J. Pinggera, H.A. Reijers, B. Weber, and G. Poels. “A visual analysis of the process of process modeling”. In: *Information Systems and e-Business Management* 13.1 (2015), pp. 147–190. DOI: 10.1007/s10257-014-0245-4.
- [46] Maximilian Riefer, Simon Felix Ternis, and Tom Thaler. “Mining Process Models from Natural Language Text: A State-of-the-Art Analysis”. In: *Multikonferenz Wirtschaftsinformatik (MKWI-16), March 9-11, Ilmenau, Germany*. Universität Ilmenau, Jan. 1, 2016.
- [47] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. “The Stanford CoreNLP Natural Language Processing Toolkit”. In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 2014, pp. 55–60.
- [48] Luis Quishpi, Josep Carmona, and Lluís Padró. “Extracting Annotations from Textual Descriptions of Processes”. In: *International Conference on Business Process Management*. Springer. 2020, pp. 184–201.
- [49] Henrik Leopold, Jan Mendling, and Artem Polyvyanyy. “Generating natural language texts from business process models”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2012, pp. 64–79.
- [50] Banu Aysolmaz, Henrik Leopold, Hajo A Reijers, and Onur Demirörs. “A semi-automated approach for generating natural language requirements documents based on business process models”. In: *Information and Software Technology* 93 (2018), pp. 14–29.
- [51] Anselmo López, Josep Sànchez-Ferreres, Josep Carmona, and Lluís Padró. “From process models to chatbots”. In: *International Conference on Advanced Information Systems Engineering*. Springer. 2019, pp. 383–398.
- [52] Goncalo Antunes, Marzieh Bakhshandeh, Jose Borbinha, Joao Cardoso, Sharam Dadashnia, Chiara Di Francescomarino, Mauro Dragoni, Peter Fettke, Avigdor Gal, Chiara Ghidini, et al. “The process model matching contest 2015”. In: *GI-Edition/Proceedings: Lecture notes in informatics* 248 (2015), pp. 127–155.
- [53] Henrik Leopold, Mathias Niepert, Matthias Weidlich, Jan Mendling, Remco M Dijkman, and Heiner Stuckenschmidt. “Probabilistic optimization of semantic process model matching”. In: *International Conference on Business Process Management*. Springer. 2012, pp. 319–334.
- [54] Christopher Klinkmüller, Ingo Weber, Jan Mendling, Henrik Leopold, and André Ludwig. “Increasing recall of process model matching by improved activity label matching”. In: *International Conference on Business Process Management*. Springer, 2013, pp. 211–218.
- [55] Han van der Aa, Henrik Leopold, and Hajo A Reijers. “Detecting inconsistencies between process models and textual descriptions”. In: *International Conference on Business Process Management*. Springer. 2016, pp. 90–105.
- [56] Josep Sànchez-Ferreres, Han van der Aa, Josep Carmona, and Lluís Padró. “Aligning textual and model-based process descriptions”. In: *Data & Knowledge Engineering* 118 (2018), pp. 25–40.

- [57] Han Van der Aa, Josep Carmona Vargas, Henrik Leopold, Jan Mendling, and Lluís Padró. “Challenges and opportunities of applying natural language processing in business process management”. In: *COLING 2018: The 27th International Conference on Computational Linguistics: Proceedings of the Conference: August 20-26, 2018 Santa Fe, New Mexico, USA*. Association for Computational Linguistics. 2018, pp. 2791–2801.
- [58] Muneera Bano. “Addressing the challenges of requirements ambiguity: A review of empirical literature”. In: *2015 IEEE Fifth International Workshop on Empirical Requirements Engineering (EmpiRE)*. IEEE. 2015, pp. 21–24.
- [59] Fabiano Dalpiaz, Ivor Van der Schalk, and Garm Lucassen. “Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and NLP”. In: *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer. 2018, pp. 119–135.
- [60] Pushpak Bhattacharyya. “Natural language processing: A perspective from computation in presence of ambiguity, resource constraint and multilinguality”. In: *CSI journal of computing* 1.2 (2012), pp. 1–13.
- [61] Marina Danilevsky, Kun Qian, Ranit Aharonov, Yannis Katsis, Ban Kawas, and Prithviraj Sen. “A Survey of the State of Explainable AI for Natural Language Processing”. In: *arXiv preprint arXiv:2010.00711* (2020).
- [62] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).
- [63] ME Peters, M Neumann, M Iyyer, M Gardner, C Clark, K Lee, and L Zettlemoyer. “Deep contextualized word representations. arXiv 2018”. In: *arXiv preprint arXiv:1802.05365* (1802).
- [64] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).
- [65] Shoichi Morimoto. “A Survey of Formal Verification for Business Process Modeling”. In: *Computational Science – ICCS 2008*. Ed. by Marian Bubak, Geert Dick van Albada, Jack Dongarra, and Peter M. A. Sloot. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 514–522. ISBN: 978-3-540-69387-1.
- [66] M. Pesic, H. Schonenberg, and W. M. P. van der Aalst. “DECLARE: Full Support for Loosely-Structured Processes”. In: *Proc. of the Eleventh IEEE Int. Enterprise Distributed Object Computing Conference (EDOC’07)*. IEEE Computer Society, 2007, pp. 287–298.
- [67] Fabrizio Maria Maggi, Marco Montali, Michael Westergaard, and Wil M. P. van der Aalst. “Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata”. In: *Business Process Management*. Ed. by Stefanie Rinderle-Ma, Farouk Toumani, and Karsten Wolf. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 132–147. ISBN: 978-3-642-23059-2.
- [68] Wihem Arsac, Luca Compagna, Giancarlo Pellegrino, and Serena Elisa Ponta. “Security Validation of Business Processes via Model-Checking”. In: *Engineering Secure Software and Systems*. Ed. by Úlfar Erlingsson, Roel Wieringa, and Nicola Zannone. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 29–42. ISBN: 978-3-642-19125-1.

- [69] Alessandro Armando and Serena Elisa Ponta. “Model Checking of Security-Sensitive Business Processes”. In: *Formal Aspects in Security and Trust*. Ed. by Pierpaolo Degano and Joshua D. Guttman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 66–80. ISBN: 978-3-642-12459-4.
- [70] João Pimentel, Emanuel Santos, Tarcisio Pereira, Daniel Ferreira, and Jaelson Castro. “A gamified requirements inspection process for goal models”. In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. ACM, 2018, pp. 1300–1307.
- [71] B. Demuth and D. Weigel. “Web Based Software Modeling Exercises in Large-Scale Software Engineering Courses”. In: *2009 22nd Conference on Software Engineering Education and Training*. Feb. 2009, pp. 138–141. DOI: 10.1109/CSEET.2009.38.
- [72] Harald Störrle, Nick Baltsen, Henrik Christoffersen, and Anja M Maier. “On the impact of diagram layout: How are models actually read?” In: *PSRC@ MoDELS*. 2014, pp. 31–35.
- [73] Andrea Burattin, Michael Kaiser, Manuel Neurauter, and Barbara Weber. “Eye tracking meets the process of process modeling: a visual analytic approach”. In: *International Conference on Business Process Management*. Springer, 2016, pp. 461–473.
- [74] Andy Kurnia, Andrew Lim, and Brenda Cheang. “Online Judge”. In: *Computers & Education* 36.4 (2001), pp. 299–315. DOI: 10.1016/S0360-1315(01)00018-5. URL: [https://doi.org/10.1016/S0360-1315\(01\)00018-5](https://doi.org/10.1016/S0360-1315(01)00018-5).
- [75] Mike Joy, Nathan Griffiths, and Russell Boyatt. “The boss online submission and assessment system”. In: *ACM Journal of Educational Resources in Computing* 5.3 (2005), p. 2. DOI: 10.1145/1163405.1163407. URL: <http://doi.acm.org/10.1145/1163405.1163407>.
- [76] Rishabh Singh, Sumit Gulwani, and Armando Solar-Lezama. “Automated feedback generation for introductory programming assignments”. In: *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI ’13, Seattle, WA, USA, June 16-19, 2013*. 2013, pp. 15–26.
- [77] Jordi Petit, Salvador Roura, Josep Carmona, Jordi Cortadella, Amalia Duch, Omer Giménez, Anaga Mani, Jan Mas, Enric Rodríguez-Carbonell, Albert Rubio, Javier de San Pedro, and Divya Venkataramani. “Jutge.org: Characteristics and Experiences”. In: *IEEE Transactions on Learning Technologies* (2017). DOI: 10.1109/TLT.2017.2723389.
- [78] Han Van der Aa, Henrik Leopold, and Hajo A Reijers. “Comparing Textual Descriptions to Process Models: The Automatic Detection of Inconsistencies”. In: *Information Systems* 64 (2017), pp. 447–460.

Appendix I

Aligning Textual and Model-Based Process Descriptions

- **Publication Type:** Journal paper
- **Published In:** Data Knowledge Engineering
- **Impact Factor:** 1.583, Q3
- **Full Reference:** Josep Sànchez-Ferreres, Han van der Aa, Josep Carmona and Lluís Padró. “Aligning textual and model-based process descriptions”. In: *Data Knowledge Engineering* 118 (2018), pp. 25-40.

Aligning Textual and Model-Based Process Descriptions

Josep Sànchez-Ferreres^{a,*}, Han van der Aa^b, Josep Carmona^a, Lluís Padró^a

^a*Universitat Politècnica de Catalunya, Barcelona, Spain*

^b*Humboldt University of Berlin, Berlin, Germany*

Abstract

Process model descriptions are an ubiquitous source of information that exists in any organization. To reach different types of stakeholders, distinct descriptions are often kept, so that process understandability is boosted with respect to individual capabilities. While the use of distinct representations allows more stakeholders to interpret process information, it also poses a considerable challenge: to keep different process descriptions aligned. In this paper, a novel technique to align process models and textual descriptions is proposed. The technique is grounded on projecting knowledge extracted from these two representations into a uniform representation that is amenable for comparison. It applies a tailored linguistic analysis of each description, so that the important information is considered when aligning description' elements. Compared to existing approaches that address this use case, our technique provides more comprehensive alignments, which encompass process model activities, events, and gateways. Furthermore, the technique, which has been implemented into the platform `nlp4bpm.cs.upc.edu`, shows promising results based on experiments with real-world data.

Keywords:

Business process management; process models; natural language processing; alignments.

*Corresponding author. *Phone number:* 934137861

Email addresses: `jsanchezf@cs.upc.edu` (Josep Sànchez-Ferreres), `han.van.der.aa@hu-berlin.de` (Han van der Aa), `jcarmona@cs.upc.edu` (Josep Carmona), `padro@cs.upc.edu` (Lluís Padró)

1. Introduction

Organizational processes can be highly complex chains of inter-related steps, involving numerous stakeholders with various roles [52]. Due to this complexity, it is crucial that the coordination among process actors is well defined [30]. Therefore, having access to the right information on business processes is vital to their proper execution [5] and their compliance to rules and regulations [1]. To provide various stakeholders with the information that they need, organizations have recognized the value of capturing process descriptions in model-based as well as text-based representations [22, 33, 50]. The reason for maintaining both representation forms is that they each have their merits. Process models have been found to be better suited to express complex execution logic of a process in a more comprehensive manner than natural language [29]. By contrast, some stakeholders, especially workers who actually execute the process, have difficulties reading and interpreting process models and, therefore, prefer textual process descriptions over process models [7].

Despite these benefits, the usage of multiple descriptions of the same process can also lead to considerable difficulties. In particular, it is vital that the process information contained in different formats is correct, even when these formats are maintained independently from each other [50]. If users access inaccurate process descriptions, they can develop different expectations about what a process aims to establish or how it should be executed [48]. Such situations can have negative effects on the efficiency with which processes are executed and, furthermore, can lead to business process non-compliance [50]. A problem in this regard is, however, that processes are subject to continuous change [58] and, therefore, considerable manual effort is required to maintain process descriptions and clear up any conflicts. Given that organizations can have hundreds or even thousands of different different process models [36], this means that manually maintaining multiple representations for all processes is hardly manageable.

In this paper, we present an alignment approach that supports organizations in maintaining process information in both textual and model-based representa-

tions. Our approach aims to establish alignments between both representations by identifying *correspondences* between parts of a textual process description and elements in a process model. These alignments enable the identification of discrepancies between descriptions [48] and, furthermore, provide a starting point for the propagation of changes from one description to the other [55]. A quantitative evaluation demonstrates that our proposed approach outperforms alignment approaches previously developed by the authors [48, 37]. Furthermore, because our approach also identifies correspondences involving process model events and gateways, the alignments we obtain are also more comprehensive. Therefore, our approach provides an important foundation for the maintenance of process information in different representation formats.

The remainder of this paper is structured as follows. Section 2 provides necessary background information in the form of a problem illustration and discussion of related work. Section 3 presents our proposed alignment approach. Section 4 presents and discusses the results of a quantitative evaluation. Finally, Section 5 concludes the paper.

2. Background

This section discusses background information relevant to the alignment of textual and model-based process descriptions. In particular, Section 2.1 presents a running example and discusses the main challenges associated with the alignment task. Section 2.2 provides an overview of related work.

2.1. Problem Illustration

To illustrate the challenges that are associated with the alignment of textual and model-based process descriptions, consider the process model shown in Figure 1 and the textual process description contained in Table 1.

The process model is defined using the Business Process Model and Notation (BPMN), a standard notation for process models. The model contains nine *activities*, depicted as rounded rectangles. These activities denote the main tasks

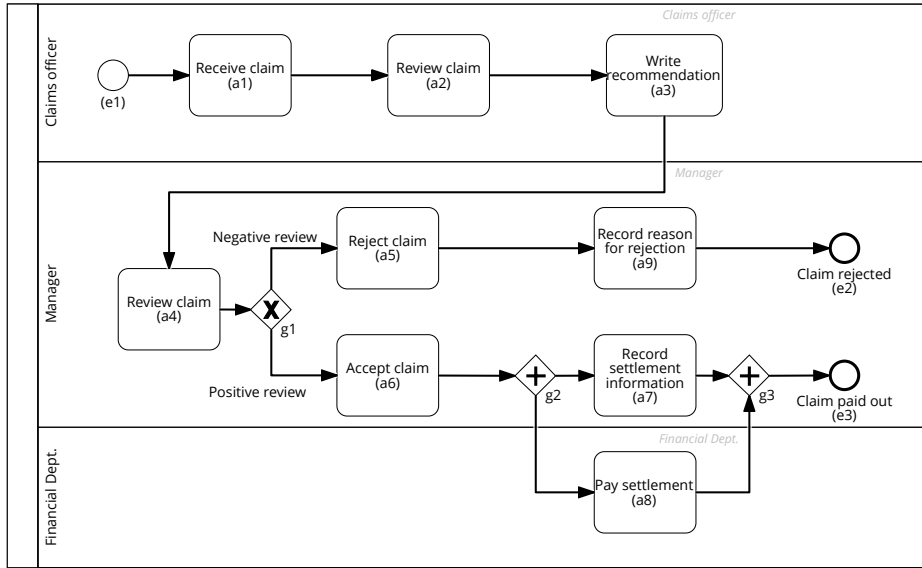


Figure 1: Exemplary process model of a claims handling process in BPMN.

performed in order to execute the process. The process model, furthermore, contains three *events*, illustrated by labeled circles, which describe the start and end points of the process. The directed edges connecting activities and events denote the control-flow of the process. The diamond shapes in the process model indicate special routing constructs in the control-flow, called *gateways*. Gateways with an X indicate a process choice, e.g. after $g1$, a claim can be either *rejected* (activity $a5$) or *accepted* ($a6$). By contrast, gateways with a + symbol indicate parallel or concurrent execution patterns. This means that the activities $a7$ and $a8$ can be executed at the same time. Finally, the horizontal lines denote so-called *swimlanes* in the process model. These swimlanes indicate which resource roles are involved in the execution of specific process steps.

The goal of aligning a process model and a textual process description is to identify correspondences between the elements of the process model and parts of the textual process description. In the context of this paper, we set out to align all elements that describe the flow in a process model, i.e. we align *activities*, *events*, and *gateways*. Our goal is here to identify the sentences of a textual process description that correspond to these process model elements,

Table 1: Textual process description of the claims handling process with correspondences to the model from Figure 1.

ID	Sentence	Corresp.
s_1	After receipt of a claim, a claims officer reviews the request.	$a1, a2$
s_2	Then, the claims officer writes a settlement recommendation and forwards it.	$a3$
s_3	A manager reviews the claim based on the written recommendation.	$a4$
s_4	If the review is negative, the claim is denied, otherwise it is accepted.	$g1, a5, a6$
s_5	In case of rejection, the process completes here.	$e2$
s_6	In case of acceptance, the manager records the settlement information, while the financial department pays the claimant.	$g2, a7, a8$

referred to as *correspondences*. The right-most column in Table 1 indicates these correspondences between the model and text.

To establish such alignments, several challenges must be overcome. These challenges are primarily caused by the flexibility of natural language, that allows the expression of the same concept via a large variety of words or phrases. In particular, we identified the following four main challenges during our earlier works on the development of alignment approaches [37, 48]:

1. *C1: Different grammatical structures:* Textual process descriptions can use a broad variety of grammatical structures to describe a process. As a result, there can exist considerable differences between the way in which a text and a model describe similar aspects of a process. Consider, for instance, the first phrase in sentence s_1 , “*After receipt of a claim*”, and the corresponding model activity “*receive claim*”. The former uses a noun-based structure, whereas the latter used a verb-based description of the same task. Therefore, an alignment technique must be able to detect such correspondences despite the presence of grammatical differences.

2. *C2: Different terminology:* Next to differences in grammatical structures, there can exist considerable differences in the terminology used between model and text. Consider, for instance, sentence s_4 , which refers to a claim being “denied” and activity a_4 , “reject claim”. An alignment technique must be able to detect that the sentence and activity refer to the same step in a process. It is here important to note that terminological differences play an even bigger role when there are also differences in the level of detail used by the two types of descriptions [46].
3. *C3: Activities with identical labels:* A different challenge occurs when process models use activities with similar or even identical labels. This can, for instance, happen in larger processes with multiple actors or when the labels used in a model are fairly coarse-granular. Activities a_2 and a_4 provide an example of this, because they both have the label “Review claim”. When establishing alignments, an alignment should be able to recognize that these identically labeled activities refer to process steps that are performed in different parts of the process, either by different actors (i.e., a_2 by a *claims officer* and a_4 by a *manager*), or even by the same actor but in different contexts.
4. *C4: Partial alignments:* Finally, it is important to recognize that a process model and textual description may not describe exactly the same steps that comprise a process, whether intentional or not [48]. For instance, activity a_9 , “Record rejection reason”, does not have a corresponding sentence in the textual description and should, therefore, not be part of a correspondence. As a result, to produce a correct alignment, alignment techniques must also be able to detect when certain process model elements actually do not appear in the text. Those process model elements should, therefore, not be included in any correspondence.

These challenges illustrate the complexity associated with the alignment of textual process descriptions and process models. To overcome these challenges, we build on techniques from the areas of *natural language processing* and *match-*

ing, as discussed next.

2.2. Related Work

In this section we discuss how natural language processing (NLP) is applied in the context of Business Process Management (Section 2.2.1) and discuss various alignment approaches that exist in this context (Section 2.2.2).

2.2.1. Natural Language Processing in Business Process Management

NLP techniques are applied to address a variety of use cases in the context of business process management [43]. 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 detecting violations of labeling conventions [3, 20, 53], inconsistent use of terminology [18], or common modeling errors [16]. Other approaches use NLP to augment process models with semantic or ontological information [21, 12, 4].

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 [41], group stories [8], or methodological descriptions [11], while others take general textual process descriptions as input [15, 13]. However, these approaches have been found to produce inaccurate models, which require extensive manual revision [39]. Other use cases involving texts include a technique that considers *work instructions* when querying process repositories [25] for conformance checking against textual process descriptions [49].

2.2.2. Process Matching

The establishment of alignments between artifacts, often referred to as *matching*, has received considerable attention in the context of BPM. In particular the importance of *process model matching* has been recognized, in which alignments between two process models are established. This has resulted in the development of a considerable number of matching techniques. Nearly all process model matchers focus on the analysis of similarity between the labels of process model elements. To achieve this, techniques consider label similarity from a *syntactic*

perspective [57, 9, 27] as well as from a *semantic* perspective [23, 35, 38]. The former set focuses on how similar the characters used in labels are, whereas the later set focuses on similarity in the meaning of labels. Aside from the analysis of process model labels, existing matching techniques have also recognized the importance *behavioral* or *structural* characteristics [17]. By taking such characteristics into account, matchers, such as [9, 17, 26], are able to recognize if activities occur in the same parts of a process. Other techniques also focus on matching based on other information, such as a technique that take work instructions associated with models into account [57], as well as a technique that matches based on event-log information [44]. A new approach by Meilicke et al. [28] provides a means to create an *ensemble* of various process model matchers that can combine their different strengths.

Aside from techniques that establish alignments between different process models, focus has recently shifted towards the establishment of alignments among a broader range of process-related artifacts. For example, several techniques exist that establish correspondences between *event logs* and process models [2, 40], and a technique for the alignment of process performance indicators and process models [45].

There are several key differences that distinguish the technique proposed in this paper when compared to those existing works. First, our technique establishes more comprehensive alignments between model and text, because the alignments cover activities, events, and gateways, whereas the previous works only focus on the alignment of activities. Second, in order to address challenge C3 described in the previous section, which relates to activities with identical labels, our technique explicitly considers resource-related information. Third, we here encode the constraints into an *Integer Linear Problem (ILP) optimization*, rather than using a *best-first* search algorithm, which greatly improves the computational efficiency of our technique and provides more flexibility with respect to the inclusion of additional constraints and their weights. Finally, our approach applies predictors defined in [48] in a novel manner to detect and adapt to differences between process model and text. This allows us to also

address challenge C4, which implies that alignments between model and text are not always complete.

3. Alignment Approach

This section describes our proposed alignment approach. It takes as input a textual process description and a process model. Our approach is tailored towards graph-based process model notations like BPMN, Petri nets or Event-Driven Process Chains, and on the other hand imposes no restrictions on the structure of the textual process description.” Furthermore, these two formats may have been defined and maintained independently from each other. Given this input, the approach aims to establish an optimal alignment between the sentences of the textual description and the elements, i.e., the activities, events, and gateways, of the process model. To achieve this, our approach consists of four main steps, as visualized in Figure 2.

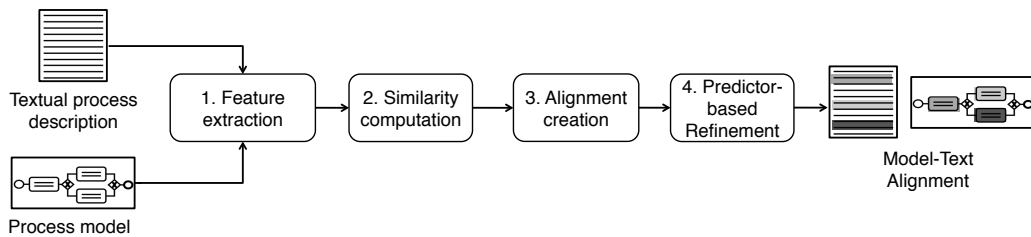


Figure 2: Overview of our alignment approach

For a model-text pair, our approach first aims to extract features that correspond to important process-related and linguistic information from the provided inputs. For example, we extract information about actions, actors, and business objects. Second, based on the features extracted, we set out to quantify the semantic similarity between process model elements and sentences. Third, we combine the semantic similarity scores with ordering information (i.e., information about the process flow) in order to establish an alignment between the textual description and process model. In the fourth and final step, our approach uses so-called predictors to detect if a provided model-text pair is likely

to contain inconsistencies. If this is the case, our approach uses this information to refine the previously established alignment in order to produce the final result.

In the remainder of this section, we describe the steps of our approach in detail.

3.1. Process Information Extraction

The goal of the first step in our approach is to extract process-related information from both a textual process description and a process model. Through this extraction step, we convert process information contained in the two heterogeneous sources into a format that enables their accurate comparison.

3.1.1. Extraction from Process Models

To describe the extraction approach, we first need to define the notion of a process model. Process models can be created using a variety of modeling languages, such as Petri nets, Event-Driven Process Chains (EPCs), and the Business Process Model and Notation (BPMN). The contributions of this paper are independent of the specific notation used to define a process model. Therefore, we define process models using the relevant parts of the generic definition provided in [19, p.13], given in Definition 1.

Definition 1 (Process Model) We define a process model as a tuple $M = (A, E, G, R, L, N, F, t, \rho, \lambda)$, where:

- A is a finite set of activities,
- E is a finite set of events,
- G is a finite set of gateways,
- R is a finite set of resources,
- L is a finite set of labels,
- $N = A \cup E \cup G$ is a finite set of nodes,
- $P = A \cup E$ is a finite set of process steps,
- $F \subseteq N \times N$ is the flow relation, such that (N, F) is a connected graph,

- $t : G \rightarrow \{and, xor\}$ is a mapping that associates each gateway with a type,
- $\rho : R \rightarrow A \cup E$ is a surjective mapping that associates a resource r to an activity $a \in A$ or an event $e \in E$,
- $\lambda : N \cup R \rightarrow L$ is a surjective mapping that relates process model nodes and resources to labels.

Note that this definition does not contain inclusive OR-gateways because of their marginal relevance in industrial process models [51] and since these are not generic to all graph-based modeling notations (e.g. Petri nets). From a given process model, we aim to extract the information depicted in Figure 3. In particular, we aim to extract information regarding activities, events, and gateways, their inter-relations, as well as the semantic components of their labels.

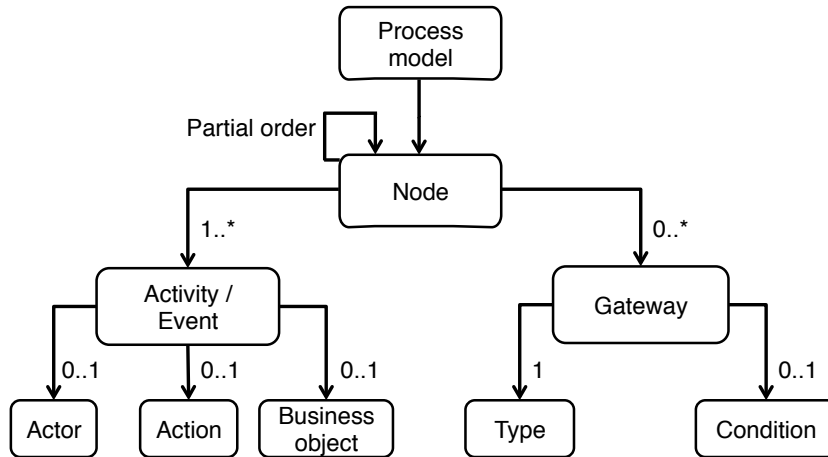


Figure 3: Process information extracted from process models

For process model activities and events, also referred to as *process steps*, we extract three semantic components: (i) an *actor* performing the step, (ii) an *action* that characterizes the step, usually described with a verb, and (iii) a *business object* on which the action is performed. We note that process steps are required to at least consist of an action, whereas the other components are optional, e.g. there can be steps without a defined actor or business object. Aside from process steps, we also extract information from the *gateways* in a process model, which denote routing aspects of a process, such as choices or

parallelism. A gateway is assigned a type, i.e., either *and* or *xor*. Furthermore, if an xor-gateway has an associated *execution condition*, such as illustrated in Figure 1 for *g1*, we augment the gateway with this information.

Process models have explicit constructs to denote activities, events, their actors, gateways, and the flow relation. Therefore, most information depicted in Figure 3 can be directly derived for process models abiding to Definition 1. However, the extraction of semantic components, namely actions and business objects, requires further processing of the natural language labels associated with process model elements:

Actions and business objects. Natural language labels associated with activities and events define their essential semantics [19]. In particular, labels generally convey the *action*, *business object*, and some additional information of a process step [31]. Therefore, the action and business objects used in the canonical format for process information need to be extracted from the labels associated with events and activities. A considerable problem here is that these labels often represent textual fragments, rather than proper sentences [20]. As a result, standard NLP techniques often fail to get accurate results for them [24]. To still be able to extract semantic components from labels, dedicated techniques have been developed that specifically aim to extract verbs, business objects, and *auxiliary* objects. For instance, Leopold [19] proposes a technique that uses knowledge about common label structures and an analysis of the model context to decompose activity and event labels. In our approach, we utilize a similar technique to extract the *action* and *business object* of process steps.

3.1.2. Extraction from Textual Descriptions

Unlike process models, textual process descriptions do not have explicit constructs that represent activities, events, or gateways. Therefore, all process-related information needs to be extracted from a textual process description using natural language processing. Our alignment approach aims to align process model elements to individual sentences. Therefore, our extraction step sets

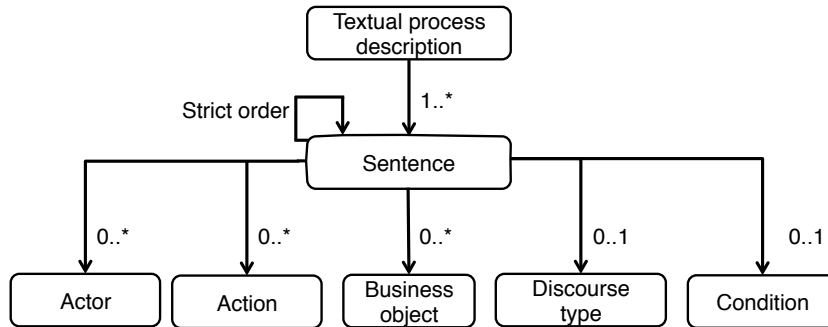


Figure 4: Process information extracted per sentence

out to identify process-related information at a sentence level. In particular, we aim to extract the components depicted in Figure 4.

As shown in the figure, we extract similar components from a sentence as we do from a process model element. The *actor*, *action*, and *business object* components correspond to the equally-named counterparts from Figure 3. These three describe the semantic components that characterize process steps. Furthermore, *discourse type* and *condition* are used to indicate if sentences contain information regarding the control-flow of a process, similar to the purpose of gateways in a process model. Finally, we also store the *strict order* relation that exists between sentences, which captures the order in which sentences appear in the text.

We extract the semantic components as follows:

Actions, actors, and business objects. In order to extract the desired process information from sentences in a textual process description, we can build on general-purpose NLP technology, such as techniques that analyze the grammatical and semantic structure of sentences and techniques for the resolution of anaphoric references (e.g., [34]). These techniques have been widely applied in the context of textual process description for the extraction of activities and their actors, cf. [13, 47, 25]. We can employ such existing techniques in order to extract *actions*, *actors*, and *business objects* from a text.

For instance for the sentence 2 from the running example, “*Then, the claims officer writes a settlement recommendation and forwards it.*”, the technique will

extract the following information: there are two *actions*: “*writes* and *forwards*”, one actor, “*claims officer*”, and a single business object, “*settlement recommendation*”. Note due to *anaphora resolution* that is part of state-of-the-art techniques, our approach is also able to identify that the term “*it*” at the end of the sentence refers to the business object “*settlement recommendation*”. This reference resolution is also possible if the business object is described in a separate sentence and can also be applied to actors that perform process steps.

Discourse types. To determine if a sentence describes a discourse marker, i.e., a choice or parallelism, we employ a technique similar to the method applied in the text-to-model generation technique from Friedrich et al. [13]. We use NLP techniques to detect discourse markers. A dictionary of multi-word expressions is applied to phrases like “in the meantime” to merge them into a single token *in_the_meantime*. As a result, our approach is able to detect if a sentence contains conditional statements (typically corresponding to process choices), such as seen for sentences *s4*, *s5*, and *s6* of the running example, or describes steps that can be executed concurrently.

Conditions. Conditions are associated with the discourse markers used to identify conditional statements, as described above. For instance, statements such as “if” or “in case of” are followed by conditions, such as seen in the phrase “*If the review is negative*” in *s3* of the running example. In those cases, the grammatical structure of the corresponding sentence is matched against several patterns in order to extract the clause containing the execution condition, e.g. to extract “*review is negative*” as a condition in sentence *s3*.

3.2. Similarity Computation

After extracting process information from a textual process description and process model, we encode the extracted information into feature vectors that can be used to accurately quantify the similarity between process model elements and sentences from a textual process description.

3.2.1. Feature Vectors

To quantify the similarity between process model nodes and sentences, we use *feature vectors* as a means to encode the information extracted from process models and sentences. These vectors represent a linearization of the extracted information that allows for an easy comparison. We define a number of different feature *types*, such as types related to *actions*, *actors*, or *business objects*. By doing so, we are able to assign different weight to the various types in order to tailor the quantification of similarity.

To quantify the similarity between a process model step p , i.e., an activity or an event, and a sentence s , we use the following feature types:

contains_action(a) This feature type denotes the actions contained in a step p or sentence s . For instance, when comparing the task “*write recommendation*” to sentence 2, we extract two actions: “*write*” and “*forward*”, that produce two feature instances *contains_action(write)* and *contains_action(forward)*. When considering these only two actions, on the task side this generates the vector $\langle 1, 0 \rangle$, whereas for the sentence we obtain the vector $\langle 1, 1 \rangle$.

contains_actor_word(w) & *actor_main_word(w)* These features denote the actors that execute steps in a process. The former feature, *contains_actor_word*, is extracted for each word w that is part of an actor. For instance, the “*claims officer*” actor comprises the two words “*claims*” and “*officer*”. By contrast, we use the *actor_main_word* feature to denote the main word of the actor, typically represented as the main noun, e.g., to explicitly capture the word “*officer*”. This separation allows us to give different importance to the main word with respect to the others. See Section 3.2.2 for more details.

contains_object_word(w) & *object_main_word(w)* These features encode the same information for *business objects* as the previously described features do for *actors*.

contains_lemma(l, pos) This feature is extracted from the target text if it contains a word¹ with the lemma l and part-of-speech pos . This feature has a lower abstraction level than the previous ones. This feature is included as a fall-back solution whenever the actor, role or action cannot be determined due to natural language ambiguity. In those cases the algorithm works at the word level. For example, for the event “Claim paid out”, the features *contains_lemma(claim, noun)* and *contains_lemma(pay-out, verb)* will be extracted.

contains_synset(s) This feature is extracted whenever the WordNet [32] 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 *05747582-n* recognizes that “review” is a synonym of “evaluation”, such as used in the running example.

contains_hyponym(s) This feature is extracted from a target text containing a word for which s is an hyponym² at distance HL or less. HL is a parameter of the algorithm. In the running example, a hyponym of “review” is “assessment” (*05733583-n*)

3.2.2. Vector Similarity

Instead of treating features as binary values, we opted for associating weights to the features individually. This way, the different importance each feature has can be considered in the comparison. The weight of each feature is the product of two magnitudes:

The feature instance weight is different for each instance of a feature type, and solves the problem of all instantiations not being equally important in terms of information provided. This is used in all lemma-based features,

¹Note that in all feature types stopwords are not considered.

²A word w_1 is a *hyponym* of w_2 iff w_1 describes a superclass of w_2 (e.g. *mammal* is a hyponym of *cat*, and *document* is a hyponym of *letter*). Hyponymy is obtained from WordNet.

where the extracted lemma has an instance weight equal to the *tf-idf* score of the word³. It is also used in synset-based features, where the instance weight of the feature is K^{-l} , where l is the length of the hipernymy chain with respect to the original synset and K is a parameter of the algorithm.

The feature family weight is a weight defined for each feature type that accounts for the problem that not all feature types are equally likely and some of them provide more information than others. For example, a process step label sharing a word with a sentence is less important than having the same main action. The second reason for family weights is to adapt the scale of the instance weights, since they measure different magnitudes depending on the feature family.

Finally, the real-valued feature vectors are compared by using a standard similarity metric. In particular, we apply the *Weighted overlapping index*, defined as follows:

$$WeightedOverlapping(A, B) = \frac{\sum_{f \in A \cap B} w_f}{\sum_{g \in smallest(A, B)} w_g} \quad (1)$$

Where w_x is the weight, i.e. the product between the family weight and the instance weight, of a feature x .

The rationale for using this metric arises from the nature of the alignment problem we address. In general, textual process descriptions are more verbose than process model. Therefore, textual descriptions have larger feature vectors. Other metrics like the *Jaccard Index* or the *Cosine Similarity* [42], produce overall lower values when the compared elements differ in size. By contrast, the *Overlapping index* is able to deal with such size differences and, hence, provides more intuitive results.

³We define the *tf-idf* of a token t as the product of $tf := (\text{Number of appearances of } t \text{ in its sentence} / \text{Number of tokens in that sentence})$ and $idf := \log_e(\text{Total number of sentences} / \text{Sentences containing } t)$

3.3. Alignment Creation

The next step in our approach is the computation of the alignment between process model elements and the sentences of a textual description. The alignment contains the corresponding sentence for each activity, gateway and event of the process model (see the rightmost column in Table 1 for an example of alignment).

Formally, we define an alignment σ to be a set of correspondences of the form $n_k \sim s_i$ for some $n_k \in N$, $s_i \in S$, where N denotes the set of nodes of a process model and S is the set of sentences that comprise a text.

3.3.1. Alignment Constraints

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, which we shall jointly refer to as *process steps* in a set $P = A \cup E$, we enforce that each of them is aligned to exactly one sentence, whereas we allow multiple steps to be aligned to the same sentence. This constraint imposes the assumptions that each step is described in the text and that steps are not described repeatedly. Furthermore, it enables the proper alignment of sentences that describe multiple process steps, such as seen for sentence 5. This sentence corresponds to both the “*record settlement information*” and “*pay settlement*” activities.

Gateway-to-sentence cardinality. Gateways require different cardinality constraints. In particular, we allow gateways from the set G to be aligned to one sentence or to none at all. Furthermore, we allow at most one gateway to be aligned to a single sentence, given that sentences typically describe at most one control-flow structure.

Process step ordering. If we, for the purposes of this step in our approach, assume that a textual process description and a process model do not contradict each other, we can impose ordering constraints on the alignments that our

approach establishes. To define these ordering restrictions, we use the following notations. First, given two nodes $n, n' \in N$, we use $n \succ n'$ to denote that there is a path from node n to n' according to the flow relation $F \subseteq N \times N$, as given in Definition 1. Second, given two sentences $s, s' \in S$, we use $s \rightsquigarrow s'$ to denote that sentence s occurs before s' in a textual process description.

We require that process steps that precede each other in a process model cannot be aligned to sentences that occur in the reverse order. Therefore, we impose the following restriction on the order of process steps from the set $P = A \cup E$: if a process step $p \in P$ precedes the execution of a step $p' \in P$, then node p cannot be aligned to a sentence s that occurs after the sentence s' to which the node p' is aligned. Formally, this means that if it holds that $p \succ p'$ and $s' \rightsquigarrow s$, an alignment cannot contain both $p \sim s$ and $p' \sim s'$. Note that, in situations where two activities p and p' respectively follow each other in a loop, we only consider the order relation $p \sim p'$ and not $p' \sim p$. This enables the approach to appropriately apply the ordering constraints to loops.

Gateway ordering. The ordering constraints required for the alignment of gateways should account for several challenges. To illustrate these, consider the fragment of a textual process description and process model depicted in Figure 5. To align text and model, the following correspondences are required: $\{s7 \sim a7, s8 \sim g2, s8 \sim a8, s9 \sim e3\}$. These correspondences indicate that, unlike for process steps-to-sentences, the order in which gateways and process steps are described in a text can be reversed. In particular, the model contains relation $g2 \rightsquigarrow a7$, whereas $a7$ occurs *before* the description of $g2$ in the text. Furthermore, none of the sentences explicitly denotes when the parallel construct is closed, i.e., none of the sentences describes gateway $g3$.

Because of these complications, we can only impose fairly weak constraints. Specifically, we can state that:

- Process steps appearing before the opening gateway in the model cannot be described after the corresponding discourse marker in the text. E.g. if $g2 \sim s_i$ holds, any activity occurring before $g2$ cannot be aligned to a

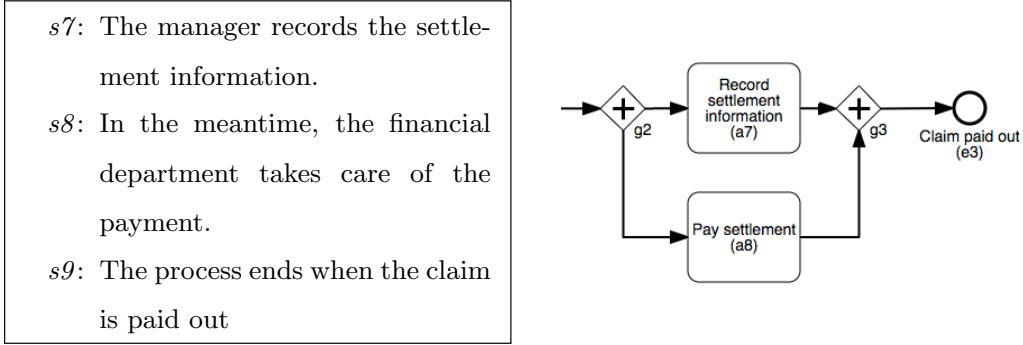


Figure 5: Description of a parallel construct in text

sentence s_j for which $s_i \rightsquigarrow s_j$ holds.

- Process steps appearing after the closing gateway in the model cannot be described before the corresponding discourse marker in the text. E.g. if $g2 \sim s_i$, then event $e3$ cannot be aligned to a sentence s_k for which $s_k \rightsquigarrow s_i$ holds.

3.3.2. Optimal Alignment

For an alignment σ to be considered optimal $\hat{\sigma}$, the following three properties must hold:

Cardinality consistency The alignment follows the cardinality constraints described in the previous section.

Order consistency The alignment is consistent with the order restrictions described in the previous section.

Optimality The value of $\sum_{n_k \sim s_i \in \hat{\sigma}} sim(n_k, s_i)$ is the maximum value such that the two other properties hold.

To define the order restrictions in a formal way we first introduce the following definitions:

$S_{disc} \subseteq S$: the set of sentences containing a discourse marker.

$G_{split} \subseteq G$: the set of gateways with one input flow and more than one output flow⁴.

$\nu : G \rightarrow G \cup \{\perp\}$: a function that given a gateway returns the corresponding gateway such that the two delimit a single-entry single-exit region in the model [54], or \perp if there is no such gateway.

In order to obtain a solution, the aforementioned properties are encoded in the following ILP:

maximize:

$$\sum_{s \in S} \sum_{n \in N} \sigma_{n,s} \cdot sim(n, s)$$

subject to:

$$\begin{aligned} \sum_{s \in S} \sigma_{p,s} &= 1 & \forall p \in P \\ \sum_{s_d \in S_{disc}} \sigma_{g,s_d} &\leq 1 & \forall g \in G \\ \sum_{g \in G} \sigma_{g,s_d} &\leq 1 & \forall s_d \in S_{disc} \\ \sigma_{p,s'} + \sigma_{p',s} &\leq 1 & \forall (s, s') \in S \times S, (p, p') \in P \times P, p \preceq p' \wedge s \rightsquigarrow s' \\ \sigma_{g,s_d} + \sigma_{s,p} &\leq 1 & \forall s \in S, s_d \in S_{disc}, g \in G_{op}, p \in P, p \preceq g, s_d \rightsquigarrow s \\ \sigma_{g,s_d} + \sigma_{s,p} &\leq 1 & \forall s \in S, s_d \in S_{disc}, g \in G_{op}, p \in P, \\ & & \nu(g) \neq \perp, \nu(g) \preceq p, s \rightsquigarrow s_d \end{aligned}$$

variables:

$$a_{n,s} \in \{0, 1\} \quad \forall s \in S, n \in N$$

The variables $\sigma_{n,s}$ can be interpreted as binary variables meaning: “Model node n corresponds to sentence s ”, i.e: $\sigma_{s,n} = 1 \iff s \sim n$. The first set of three constraints enforce the *Cardinality consistency* property⁵. This is

⁴Note that we purposefully avoid considering anomalous cases such as multiple inputs and outputs or single-input single-output gateways and restrict ourselves to the subset of well-formed BPMN models.

⁵Note that these equations can also be encoded using the *Special Ordered Sets (SOS)* constraint of the form: $a_{t,1}, \dots, a_{t,|S|}$ for all model elements t , which denotes exactly the same constraint, and has better performance on the ILP solvers that implement it.

naturally encoded in ILP by restricting the sum of a subset of binary variables to either exactly one, or at least one depending on the desired cardinality. The second set of three constraints encodes the *Order consistency* property. This is done by explicitly restricting pairs of $s \sim p$ correspondences, i.e. they cannot be both true at the same time, for those pairs that constitute an order violation.

3.4. Predictor-based Refinement

The alignments that results from Section 3.3 are established under the assumption that a process model and a textual process description describe the same process, i.e., that the two representations do not contain inconsistencies. By operating under this assumption, we are able to impose constraints that have considerable positive effects on the quality of the resulting alignments, most notably resulting from the application of the ordering constraints described in Section 3.3.1. Nevertheless, it is important to acknowledge that varying descriptions of a process can contradict each other in practical settings [50]. Therefore, this section presents the final step of our approach in which we set out to detect the presence of inconsistencies and, if so, adapt obtained alignments accordingly.

To perform this refinement step, we first apply so-called *predictors* that quantify the likelihood that an obtained alignment contains inconsistencies. This notion of a predictor is inspired by according notions used to analyze alignments in the context of schema and process model matching [14, 56] and were originally applied in Van der Aa et al. [48]. The core premise underlying predictors is that alignments associated with consistent and inconsistent model-text pairs differ. For a consistent model-text pair, all process model elements should be aligned to a sentence with a high similarity score. By contrast, the similarity scores of the correspondences in the alignment of an inconsistent model-text pair will have different characteristics. Predictors quantify these characteristics and, as such, quantify the likelihood that an alignment contains a particular kind of inconsistency. If these predictors indeed detect likely inconsistencies, we subsequently weaken the alignment constraints accordingly in order to establish a refined alignment that takes the presence of likely inconsistencies into account.

s : “A manager must review the claim before he can accept it”

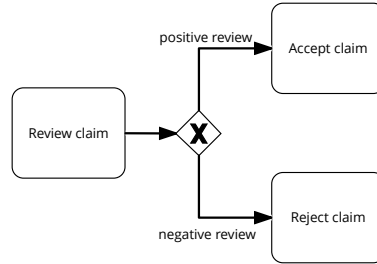


Figure 6: Example of a missing activity

We perform this refinement step for two types of inconsistencies: (i) process model elements that are missing from a textual description and (ii) ordering conflicts.

3.4.1. Missing Process Steps

The alignment technique described in Section 3.3 aligns each activity and event to a single sentence. However, it can happen that not all model elements are actually described in the text. Consider for instance the example depicted in Figure 6. In this fragment of a model-text pair, the textual description only focuses on the positive outcome of a review step and, therefore, does not describe the possibility that a claim can also be rejected. By contrast, the process model shows both possibilities.

The similarity scores of correspondences included in an alignment represent highly valuable indicators to identify missing elements. Process model elements that are contained in a textual description are expected to be aligned to sentences with a high similarity score. By contrast, if a process model element is not described in a textual description, it cannot be aligned to a sentence with a high similarity score, because none of the sentences in the text describe the same process step as the model elements. For instance, none of the sentences related to the example from Figure 6 contain terms related to the rejection of a claim, which results in considerably lower similarity scores for the “*reject claim*” activity. Given these lower similarity scores of missing process model elements,

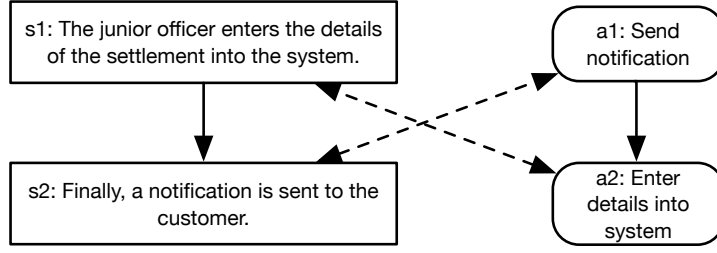


Figure 7: Example of an ordering conflict

we use predictors that recognize such instances.

To distinguish between *high* and *low* similarity scores, we evaluate similarity scores in an optimal alignment according to the following predictors:

- $p\text{-sim}(p, s)$: the likelihood that a correspondence $p \sim s$ relates to a missing step, given as the similarity score between the step p and the sentence s i.e. $\text{sim}(p, s)$.
- $p\text{-rel-}S(p, s)$: the value of the previous predictor, normalized by the maximum similarity between s and any other process step, i.e.

$$\frac{\text{sim}(p, s)}{\max \{ \text{sim}(p', s) \mid p' \in P \}}$$

Subsequently, we remove correspondences from the obtained alignment that are likely to be missing from the textual description, i.e. the correspondences for which the predictor values are below a certain threshold. In this manner we can improve the quality of obtained alignments in terms of their *precision*, because incorrect correspondences will be excluded.

3.4.2. Ordering Conflicts

The ordering constraints imposed on alignments, described in Section 3.3.1, assume that a process model and a corresponding textual description describe the various steps of a process in the same order. However, when a model-text pair is inconsistent, it can happen that text and model contain ordering conflicts. Figure 7 presents an example of this, in which the two artifacts denote a different order of the “*send notification*” and “*enter details into system*” activities.

Due to the applied ordering constraints, alignments cannot contain both correspondences $a1 \sim s2$ and $a2 \sim s1$, even though these denote actual correspondences between model and text. To refine such alignments, we apply a predictor that aims to identify ordering conflicts, which allows us to remove the ordering constraints that impede the ability to establish alignments. The existence of conflicting orders between model and text can manifest itself in the form of large differences between the similarity scores contained in an optimal alignment and potential similarity scores that could have been achieved without these ordering constraints. For instance, in the above example, the total similarity scores of having the true correspondences $a1 \sim s2$ and $a2 \sim s1$ will be much higher than the similarity scores of the correspondences that abide to the ordering constraints.

We capture this characteristic difference between consistent and inconsistent model-text pairs in the predictor *max-constrained*. This predictor quantifies the maximum difference that exists between the aligned and potential score for a single process step in a model-text pair. It thus captures the largest similarity difference caused by imposing ordering restrictions on the optimal alignment. We operationalize this as follows:

- *max-constrained*(\sim): the maximal difference between the potential and aligned similarity scores for a process step $p \in P$.

For the cases where the predictor detects a likely ordering conflict, i.e. where the value of *max-constrained* is above a certain threshold, our approach recomputes a renewed alignment without ordering constraints. For instance, for the provided example, the refined alignment will contain both correspondences $a1 \sim s2$ and $a2 \sim s1$.

4. Evaluation

To demonstrate the capabilities of our approach, we conduct a quantitative evaluation by comparing automatically generated model-text alignments to a

manually created *gold standard*. The goal of this evaluation is to assess the quality of the automatically generated alignments. Both the data collection and the implementation of our approach used to conduct the evaluation are publicly available ⁶.

4.1. Test Collection

To perform the evaluation, we use a set of 74 model-text pairs obtained from various sources. Given the different nature of these sources, we partition the set into two data collections.

The first collection consists of 49 model-text pairs, corresponding to the original dataset used for the evaluation of existing model-text alignment approaches [48, 37]. In this test collection, we detected 30 models (61%) containing at least one missing activity and an average of 1.32 nodes per sentence. The model-text pairs in this dataset have been obtained from 11 different industrial and academic sources, including *inubit AG*, the *Federal Network Agency* of Germany, and various universities. [48] provides an in-depth overview of these sources. The gold standard alignments of this collection were built on the already established ones used in [37]. However, since the existing gold standard only included activity-to-sentence correspondences, we augmented it with correspondences involving events and gateways.

The second collection consists of 25 additional model-text pairs. For this second test collection, we observed 12 models (48%) with at least one missing activity and an average of 1.54 nodes per each sentence. The process models in this collection have been obtained from the repository of the *BPM Academic Initiative*[10]. The texts accompanying these models were authored by 8 experienced modelers. In order to obtain textual descriptions covering a variety of styles, the experts were asked to perform the following three steps: (i) Study the process model diagram. (ii) Write the textual description without looking at the source model. (iii) Compare the textual description with the source model to

⁶https://github.com/setzer22/alignment_model_text

make sure the text accurately describes the process model. This final step was introduced to reduce the amount of inconsistencies between texts and models. Furthermore, the authors of the text were not involved in the development of the approach of this paper nor aware of the exact purpose of their task. For the 25 model-text pairs in this new collection, the gold standard alignments were annotated and subsequently verified by the authors of this paper.

Table 2 provides an overview of the characteristics of the model-text pairs included in the evaluation set.

Table 2: Characteristics of model-text pairs in the test collections

Collection	P	P_{ma}	N	A	G	E	S	N/S
Original	49	30	11.99	8.12	1.66	2.21	9.13	1.32
New	25	12	12.40	7.44	2.80	2.16	7.72	1.54
Total	74	42	12.13	7.89	2.05	2.19	8.65	1.39

Legend: P = Model-text pairs, P_{ma} = Amount of model-text pairs with missing elements, N = Nodes per model (avg.), A = Activities per model (avg.), G = Gateways per model (avg.), E = Events per model (avg.) S = Sentences per text (avg.), N/S = Nodes per sentence (avg.)

4.2. Setup

To conduct the evaluation, we have implemented our proposed alignment approach in the form of a Java prototype. For this implementation we used Freeling [34] for the natural language processing and Gurobi as the ILP solver to compute optimal alignments.

To quantify the quality of a generated alignment for a given model-text pair we compute the widely-employed *accuracy* metric. This metric quantifies the number of correct correspondences with respect to the total number of correspondences. Let σ be the alignment generated by our approach over a set of sentences S and a set of process model nodes N . furthermore, let σ^* be

the gold standard alignment. We then define the *accuracy* of our approach as follows:

$$\text{accuracy} = \frac{|\sigma \cap \sigma^*|}{|N|}$$

In order to operationalize our implementation, the parameters of the approach have been set in the following way: The feature family weights used were tuned using a genetic algorithm exploration. The predictor used for missing elements is *p-rel-S*, with a threshold value of 0.1. The predictor used for order conflicts was *max-constrained*, with a threshold value of 0.8. In Section 4.4 we explore the impact of alternate parameter settings on the performance of our approach. Finally, as a benchmark, we compare the performance of our approach to the two earlier proposed alignment techniques from [37, 48].

4.3. Results

Table 3 shows an overview of the evaluation results for our approach with and without predictors, as well as for the two benchmark approaches. In particular, we depict the accuracy our approach achieves on the alignment of activities, events, gateways, and the overall accuracy.

The result shows that our approach achieves an overall high accuracy of 0.71 for the total test collection. When considering the different types of nodes, we observe that the approach performs best for the alignment of activities, achieving an accuracy of 0.79. For events (0.56) and gateways (0.50), the approach is less accurate. This difference in performance could be explained by the fact that our approach is more informed with respect to activities than the other two element types. Particularly, the *agent*, *action* and *business object* features are not extracted for gateways and do not always fit the writing style of event descriptions. Furthermore, we can observe that the use of predictor-based refinement has a positive impact on the accuracy of the approach, increasing the accuracy from an overall of 0.69 to 0.71. The benefits of using predictors are most apparent for the alignment of gateways, where the accuracy increases from 0.40 to 0.50.

Table 3: Overview of the evaluation results

Collection	Configuration	Accuracy			
		Activities	Events	Gateways	Overall
Original	Approach from [37]	0.76	n/a	n/a	n/a
	Approach from [48]	0.78	n/a	n/a	n/a
	Without predictors	0.78	0.42	0.32	0.66
	With predictors	0.78	0.44	0.43	0.68
New	Approach from [37]	0.79	n/a	n/a	n/a
	Approach from [48]	0.77	n/a	n/a	n/a
	Without predictors	0.81	0.80	0.55	0.75
	With predictors	0.80	0.80	0.63	0.76
Total	Approach from [37]	0.78	n/a	n/a	n/a
	Approach from [48]	0.77	n/a	n/a	n/a
	Without predictors	0.79	0.55	0.40	0.69
	With predictors	0.79	0.56	0.50	0.71

The results show that our approach slightly improves upon the accuracy achieved by the existing approaches from [37, 48]. However, the primary difference, as clearly shown in Table 3 is that our proposed approach provides much more complete alignments in terms of the process model elements that it considers.

When comparing the performance of the approach across the two test collections, we observe that the performance is comparable with respect to the alignment of activities (0.78 versus 0.80). However, for events (0.44 versus 0.80) and gateways (0.43 versus 0.63), the approach performs considerably better on the model-text pairs in the new collection. There are several factors that may have influenced this performance gap. On one hand, the new dataset has a substantially higher amount of labelled gateways, which help create more informed alignments. On the other hand, generating textual descriptions from models may favor a more literal style when describing events when compared

to the some independently developed descriptions of the original dataset. Despite these differences, the evaluation results show that the approach achieves promising results for model-text pairs from a wide range of sources.

By taking a more in-depth look at the results, we can furthermore make the following observations:

- Our approach is able to establish alignments even when a model contains multiple activities with identical or near-identical labels. This is achieved because the approach considers additional semantic information obtained from features like actors and the structural information provided by the ordering constraints.
- The size of a process models, in terms of the notes to be aligned, has no significant impact on the performance of our approach. The average alignment accuracy for models with $N > 15$ is 0.70 and 0.65 for $N > 20$, which does not differ significantly from the average performance of 0.75 for all models (with average $N = 12.13$). This observation is confirmed through a Pearson correlation test [6], which yielded a correlation coefficient of -0.11 and a p-value of 0.33, indicating no significant correlation between model size and alignment accuracy.
- Using the *tf-idf* as a multiplicative factor for word-based features helps automatically regulate their importance: A word that is used throughout the process does not contribute as much to the similarity as one that is only mentioned in a subset of the description. This is important in the case of gateways, where the high-level information –i.e. agent, action and business object– cannot be obtained and the word-level features alone determine the similarity.
- Our approach currently treats events in the same way as activities. While some events are typically described like activities, some other event types are more naturally described in a different style. One such example is timer events, where sentences like “After X days have passed” are more

common. Parsing errors that result from trying to find activity structure in those events result in less informed alignments.

- Most gateway alignments missed in the evaluation were implicitly described in the textual description. For example, a process model described different alternatives for sorting invoices with an exclusive gateway, but the textual description simply stated: “The invoices can be sorted in two ways: by amount and by vendor.” Such implicit descriptions cannot be detected by our alignment tool since no discourse marker is present.

4.4. Influence of the Parameters

The parametrization of our proposed approach can have a considerable impact on the quality of the obtained results. In order to understand how this occurs, we study the effect of the following parameters:

Feature family weights (Section 3.2.2) define the importance of features with respect to each other.

Predictors, and their thresholds (Section 3.4) define the strategy and likelihood of detecting a *missing* node or an *order inconsistency* in the process.

In the next two sections we report the exploration done which additionally lead us to the final parameters parameters used to obtain the results shown in Table 3.

4.4.1. Effect of Feature Weights

To assess the influence of feature weights, we considered several exploration techniques. Using an exhaustive exploration technique, or an experimental design requires discretising the weights. Due to the high sensibility of the parameters and the computation time required to evaluate each individual combination, we avoided such an exhaustive analysis. Instead, we opted for an heuristic search based on *genetic algorithms*, which are well known for metaparameter



Figure 8: Evolution of the genetic algorithm optimization showing the population maximum, average and minimum fitness values.

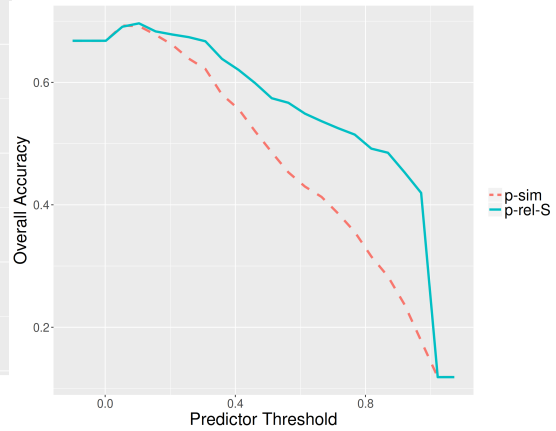


Figure 9: Accuracy of the tool when varying the threshold for different predictors.

optimization. In order to guide the search, the fitness function was defined as the overall accuracy obtained by the tool using the original dataset.

Figure 8 shows the evolution of the optimization for one of the executions. The maximum and average values for the population fitness have a tendency to increase, which shows the positive effect of a good parametrization. On the other hand, the minimum values fluctuate uniformly because of random individuals being added at each generation. This shows the impact of the weight parameters in our technique. While a bad set of weights affects the results negatively, the algorithm can still perform with a reasonable accuracy even with such bad parametrization.

Some common characteristics were observed in the fittest individuals of the last generation: The most important features were actions, business objects and discourse markers. On the other hand, word-based features and actors had substantially lower weights. Finally, for features that distinguish the main word, such as *contains_actor_word* and *actor_main_word*, the latter type was found to consistently have a higher weight.

4.4.2. Effect of Predictors and Thresholds

In order to study the effect of predictors, we conducted an experiment using only the first test collection. The goal was to observe the effect of the predictor type (*p-sim* or *p-rel-S*), as well as the threshold value used to detect a missing step, on the overall accuracy.

Figure 9 shows the performance of the tool when using both predictors and varying their thresholds from 0 to 1, with a step length of 0.05. The value at 0 indicates the performance of the tool when not using predictors, while the value at 1 represents the maximum achievable improvement of using predictors for the detection of missing elements.

As shown, all predictor types have an initial peak region where performing the predictor-based refinement offers some benefit. After some point, the approach becomes too strict and considers too many elements as missing, resulting in a drop in the overall accuracy. This peak region is similar for the three approaches, but *p-rel-S* is more stable since it offers a less steep curve.

This exploration shows us the potential benefits of predictor-based refinement in our technique. We conclude a good range of values for the similarity threshold lays in the interval (0.05, 0.2). Finally, the increased stability of the *p-rel-S* predictor makes it more suited for a generic approach.

5. Conclusions

This paper presented a fully automated approach to align textual descriptions to process models. The proposed approach combines tailored NLP processing techniques, semantic matching, and predictors in order to establish optimal alignments between the nodes of a process model and the sentences of a textual description. Unlike existing approaches that address this task, our approach aligns a broad range of process model elements, including events and gateways, rather than just focusing on the alignment of activities. A quantitative evaluation performed on a collection 74 model-text pairs demonstrates that our approach achieves satisfactory results.

We foresee several research directions that can be followed. On the one hand, incorporating support to more element types and constructs may lead to a more precise analysis; among others, we may consider *subprocesses*. Another interesting direction is to improve the characterization and computation of the order for the sentences in the text, for instance learning a classifier tailored towards describing control flow in textual descriptions.

Acknowledgements

This work has been supported by funds from the Spanish Ministry for Economy and Competitiveness (MINECO), the European Union (FEDER funds) via grants GRAMM (TIN2017-86727-C2-1-R) and Graph-Med (ref. TIN2013-46181-C2-1-R, TIN2016-77820-C3-3-R), and the Alexander von Humboldt Foundation.

References

- [1] E. Andrade, H. van der Aa, H. Leopold, S. Alter, H. A. Reijers, Factors Leading to Business Process Noncompliance and its Positive and Negative Effects: Empirical Insights from a Case Study, in: 22nd Americas Conference on Information Systems, AMCIS, 2016.
- [2] T. Baier, J. Mendling, M. Weske, Bridging abstraction layers in process mining, *Information Systems* 46 (2014) 123–139.
- [3] J. Becker, P. Delfmann, S. Herwig, L. Lis, A. Stein, Formalizing Linguistic Conventions for Conceptual Models, in: *Conceptual Modeling - ER 2009*, LNCS, Springer Berlin Heidelberg, 70–83, 2009.
- [4] M. Born, F. Dörr, I. Weber, User-Friendly Semantic Annotation in Business Process Modeling, in: *WISE 2007 Workshops*, vol. 4832 of *LNCS*, Springer, 260–271, 2007.

- [5] T. R. Browning, On the alignment of the purposes and views of process models in project management, *Journal of Operations Management* 28 (4) (2010) 316–332.
- [6] J. L. Bruning, B. L. Kintz, *Computational handbook of statistics*, Scott, Foresman & Co, 1987.
- [7] S. Chakraborty, S. Sarker, S. Sarker, An exploration into the process of requirements elicitation: A grounded approach, *Journal of the association for information systems* 11 (4) (2010) 212–249.
- [8] J. C. de Gonçalves, F. M. Santoro, F. A. Baiao, Business process mining from group stories, in: *Computer Supported Cooperative Work in Design, 2009. CSCWD 2009. 13th International Conference on*, IEEE, 161–166, 2009.
- [9] R. M. Dijkman, M. Dumas, L. García-Bañuelos, Graph matching algorithms for business process model similarity search, in: *International Conference on Business Process Management*, Springer, 48–63, 2009.
- [10] R.-H. Eid-Sabbagh, M. Kunze, A. Meyer, M. Weske, A Platform for Research on Process Model Collections, in: J. Mendling, M. Weidlich (Eds.), *Business Process Model and Notation*, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-642-33155-8, 8–22, 2012.
- [11] E. V. Epure, P. Martín-Rodilla, C. Hug, R. Deneckère, C. Salinesi, Automatic process model discovery from textual methodologies, in: *Research Challenges in Information Science (RCIS), 2015 IEEE 9th International Conference on*, IEEE, 19–30, 2015.
- [12] C. Francescomarino, 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 of *LNBIP*, Springer, 211–223, 2009.

- [13] F. Friedrich, J. Mendling, F. Puhlmann, Process model generation from natural language text, in: *International Conference on Advanced Information Systems Engineering*, Springer, 482–496, 2011.
- [14] A. Gal, Uncertain schema matching, *Synthesis Lectures on Data Management* 3 (1) (2011) 1–97.
- [15] A. Ghose, G. Koliadis, A. Chueng, Process discovery from model and text artefacts, in: *Services, 2007 IEEE Congress on*, IEEE, 167–174, 2007.
- [16] V. Gruhn, R. Laue, Detecting Common Errors in Event-Driven Process Chains by Label Analysis, *Enterprise Modelling and Information Systems Architectures* 6 (1) (2011) 3–15.
- [17] J.-Y. Jung, J. Bae, Workflow clustering method based on process similarity, in: *International Conference on Computational Science and Its Applications*, Springer, 379–389, 2006.
- [18] A. Koschmider, E. Blanchard, User Assistance for Business Process Model Decomposition, in: *Proceedings of the 1st IEEE International Conference on Research Challenges in Information Science*, 445–454, 2007.
- [19] H. Leopold, *Natural language in business process models*, Springer, 2013.
- [20] H. Leopold, R.-H. Eid-Sabbagh, J. Mendling, L. G. Azevedo, F. A. Baião, Detection of naming convention violations in process models for different languages, *Decision Support Systems* 56 (2013) 310–325.
- [21] H. Leopold, C. Meilicke, M. Fellmann, F. Pittke, H. Stuckenschmidt, J. Mendling, Towards the automated annotation of process models, in: *International Conference on Advanced Information Systems Engineering*, Springer, 401–416, 2015.
- [22] H. Leopold, J. Mendling, A. Polyvyanyy, Supporting Process Model Validation through Natural Language Generation, *IEEE Transactions on Software Engineering* 40 (8) (2014) 818–840.

- [23] H. Leopold, M. Niepert, M. Weidlich, J. Mendling, R. M. Dijkman, H. Stuckenschmidt, Probabilistic optimization of semantic process model matching, in: *International Conference on Business Process Management*, Springer, 319–334, 2012.
- [24] H. Leopold, S. Smirnov, J. Mendling, Refactoring of process model activity labels, in: *Natural Language Processing and Information Systems*, Springer, 268–276, 2010.
- [25] H. Leopold, H. van der Aa, F. Pittke, M. Raffel, J. Mendling, H. A. Reijers, Searching textual and model-based process descriptions based on a unified data format, *Software & Systems Modeling* (2017) 1–16.
- [26] J. Ling, L. Zhang, Q. Feng, An Improved Structure-based Approach to Measure Similarity of Business Process Models., in: *SEKE*, 377–380, 2014.
- [27] K. Liu, Z. Yan, Y. Wang, L. Wen, J. Wang, Efficient syntactic process difference detection using flexible feature matching, in: *Asia-Pacific Conference on Business Process Management*, Springer, 103–116, 2014.
- [28] C. Meilicke, H. Leopold, E. Kuss, H. Stuckenschmidt, H. A. Reijers, Overcoming individual process model matcher weaknesses using ensemble matching, *Decision Support Systems* 100 (2017) 15–26.
- [29] J. Mendling, Metrics for process models: empirical foundations of verification, error prediction, and guidelines for correctness, vol. 6, Springer Science & Business Media, 2008.
- [30] J. Mendling, B. Baesens, A. Bernstein, M. Fellmann, Challenges of smart business process management: An introduction to the special issue, *Decision Support Systems* 100 (2017) 1–5, URL <https://doi.org/10.1016/j.dss.2017.06.009>.
- [31] J. Mendling, H. A. Reijers, J. Recker, Activity labeling in process modeling: Empirical insights and recommendations, *Information Systems* 35 (4) (2010) 467–482.

- [32] G. A. Miller, WordNet: a lexical database for English, *Communications of the ACM* 38 (11) (1995) 39–41.
- [33] A. Ottensooser, A. Fekete, H. A. Reijers, J. Mendling, C. Menictas, Making sense of business process descriptions: An experimental comparison of graphical and textual notations, *Journal of Systems and Software* 85 (3) (2012) 596–606.
- [34] L. Padró, E. 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*, 2473–2479, URL <http://www.lrec-conf.org/proceedings/lrec2012/summaries/430.html>, 2012.
- [35] F. Pittke, H. Leopold, J. Mendling, G. Tamm, Enabling reuse of process models through the detection of similar process parts, in: *International Conference on Business Process Management*, Springer, 586–597, 2012.
- [36] M. Rosemann, Potential Pitfalls of Process Modeling: Part A, *Business Process Management Journal* 12 (2) (2006) 249–254.
- [37] J. Sánchez-Ferreres, J. Carmona, L. Padró, Aligning Textual and Graphical Descriptions of Processes Through ILP Techniques, in: *International Conference on Advanced Information Systems Engineering (In press)*, Springer, 2017.
- [38] M. L. Sebu, H. Ciocârlie, Similarity of business process models in a modular design, in: *Applied Computational Intelligence and Informatics (SACI), 2016 IEEE 11th International Symposium on*, IEEE, 31–36, 2016.
- [39] M. Selway, G. Grossmann, W. Mayer, M. Stumptner, Formalising natural language specifications using a cognitive linguistic/configuration based approach, *Information Systems* 54 (2015) 191–208.
- [40] A. Senderovich, A. Rogge-Solti, A. Gal, J. Mendling, A. Mandelbaum, The ROAD from Sensor Data to Process Instances via Interaction Mining, in:

- International Conference on Advanced Information Systems Engineering, Springer, 257–273, 2016.
- [41] A. Sinha, A. Paradkar, Use Cases to Process Specifications in Business Process Modeling Notation, in: IEEE International Conference on Web Services, 473–480, 2010.
- [42] P.-N. Tan, et al., Introduction to data mining, Pearson Education India, 2006.
- [43] H. van der Aa, J. Carmona, H. Leopold, J. Mendling, L. Padró, Challenges and Opportunities of Applying Natural Language Processing in Business Process Management, in: International Conference on Computational Linguistics, 2018.
- [44] H. Van der Aa, A. Gal, H. Leopold, H. A. Reijers, T. Sagi, R. Shraga, Instance-Based Process Matching using Event-Log Information, in: International Conference on Advanced Information Systems Engineering (In press), Springer, 2017.
- [45] H. Van der Aa, H. Leopold, A. del Rio-Ortega, M. Resinas, H. A. Reijers, Transforming Unstructured Natural Language Descriptions into Measurable Process Performance Indicators Using Hidden Markov Models, *Information Systems* 71 (2017) 27–39.
- [46] H. Van der Aa, H. Leopold, F. Mannhardt, H. A. Reijers, On the Fragmentation of Process Information: Challenges, Solutions, and Outlook, in: International Conference on Enterprise, Business-Process and Information Systems Modeling, Springer, 3–18, 2015.
- [47] H. Van der Aa, H. Leopold, H. A. Reijers, Checking Process Compliance on the Basis of Uncertain Event-to-Activity Mappings, in: International Conference on Advanced Information Systems Engineering (In press), Springer, 2017.

- [48] H. Van der Aa, H. Leopold, H. A. Reijers, Comparing Textual Descriptions to Process Models: The Automatic Detection of Inconsistencies, *Information Systems* 64 (2017) 447–460.
- [49] H. van der Aa, H. Leopold, H. A. Reijers, Checking Process Compliance against Natural Language Specifications using Behavioral Spaces, *Information Systems* .
- [50] H. Van der Aa, H. Leopold, I. van de Weerd, H. A. Reijers, Causes and Consequences of Fragmented Process Information: Insights from a Case Study, in: *23rd Americas Conference on Information Systems, AMCIS, 2017*.
- [51] W. M. Van der Aalst, The application of Petri nets to workflow management, *Journal of circuits, systems, and computers* 8 (01) (1998) 21–66.
- [52] W. M. P. Van der Aalst, A. H. Ter Hofstede, M. Weske, Business process management: A survey, in: *International conference on business process management*, Springer, 1–12, 2003.
- [53] B. Van der Vos, J. A. Gulla, R. van de Riet, Verification of conceptual models based on linguistic knowledge, *Data & Knowledge Engineering* 21 (2) (1997) 147 – 163.
- [54] J. Vanhatalo, H. Völzer, J. Koehler, The refined process structure tree, *Data Knowl. Eng.* 68 (9) (2009) 793–818.
- [55] M. Weidlich, J. Mendling, M. Weske, Propagating changes between aligned process models, *Journal of Systems and Software* 85 (8) (2012) 1885–1898.
- [56] M. Weidlich, T. Sagi, H. Leopold, A. Gal, J. Mendling, Predicting the quality of process model matching, in: *International Conference on Business Process Management*, Springer, 203–210, 2013.
- [57] M. Weidlich, E. Sheetrit, M. C. Branco, A. Gal, Matching business process models using positional passage-based language models, in: *International Conference on Conceptual Modeling*, Springer, 130–137, 2013.

- [58] M. Weidlich, M. Weske, J. Mendling, Change propagation in process models using behavioural profiles, in: *Services Computing, 2009. SCC'09. IEEE International Conference on*, IEEE, 33–40, 2009.

Appendix II

Supporting the Process of Learning and Teaching Process Models

- **Publication Type:** Journal paper
- **Published In:** Transactions on Learning Technologies (TLT)
- **Impact Factor:** 2.315, Q2
- **Full Reference:** Josep Sànchez-Ferreres, Luis Delicado, Amine Abbad Andaloussi, Andrea Burattin, Guillermo Calderón-Ruiz, Barbara Weber, Josep Carmona and Lluís Padró. “Supporting the Process of Learning and Teaching Process Modeling”. In: *Transactions on Learning Technologies (TLT)*. 2019.

Supporting the Process of Learning and Teaching Process Models

Josep Sànchez-Ferreres, Luis Delicado, Amine Abbad Andaloussi, Andrea Burattin, Guillermo Calderón-Ruiz, Barbara Weber, Josep Carmona, and Lluís Padró

Abstract—The creation of a process model is primarily a formalization task that faces the challenge of constructing a syntactically correct entity which accurately reflects the semantics of reality, and is understandable to the model reader. This paper proposes a framework called *Model Judge*, focused towards the two main actors in the process of learning process model creation: novice modelers and instructors. For modelers, the platform enables the automatic validation of the process models created from a textual description, providing explanations about quality issues in the model. *Model Judge* can provide diagnostics regarding model structure, writing style, and semantics by aligning annotated textual descriptions to models. For instructors, the platform facilitates the creation of modeling exercises by providing an editor to annotate the main parts of a textual description, that is empowered with natural language processing (NLP) capabilities so that the annotation effort is minimized. So far around 300 students, in process modeling courses of five different universities around the world have used the platform. The feedback gathered from some of these courses shows good potential in helping students to improve their learning experience, which might, in turn, impact process model quality and understandability. Moreover, our results show that instructors can benefit from getting insights into the evolution of modeling processes including arising quality issues of single students, but also discovering tendencies in groups of students. Although the framework has been applied to process model creation, it could be extrapolated to other contexts where the creation of models based on a textual description plays an important role.

Index Terms—Natural language processing (NLP), process of process modeling, textual annotation, Business Process Model and Notation (BPMN).

I. INTRODUCTION AND MOTIVATION

PROCESS models play an important role in the analysis and improvement of business processes [1]. Moreover, they are often used as basis for process execution. Due to their

wide usage in organizations their quality is paramount: Process models should be syntactically correct, and follow the rules provided by the modeling grammar (i.e., syntactic quality), they should be complete in terms of requirements and only contain correct and relevant statements of the domain (i.e., semantic quality), and be understandable to the human model reader (i.e., pragmatic quality) [2], [3].

Research has shown that industrial process models often contain errors [4]. While some of these errors can be automatically identified by the verification support of state-of-the-art modeling environments [5], others—especially those concerning the semantics of a model—are not sufficiently supported up to now. Moreover, a recently conducted exploratory study by Haisjackl *et al.* investigated how humans inspect process models [6], and came to the conclusion that a systematic support for model inspection would be fundamental, either in form of test-driven development, or in form of (automated) checklists.

This paper picks up this challenge, and proposes a framework called *Model Judge* for the automatic validation of process models and providing explanations about quality issues in therein. The framework proposed is inspired by similar practices in other educational contexts, e.g., the success of judges in supporting learning to program [7]–[11]. Remarkably, we use natural language processing (NLP), together with optimization techniques [12], to validate process models with respect to textual descriptions of processes.

Model Judge provides diagnostics regarding issues on syntactic, pragmatic and semantic quality. The framework underlying *Model Judge* is grounded on aligning annotated textual descriptions with the models as basis for providing diagnostics [12]. Using *Model Judge*, novice modelers can be guided through continuous feedback. Instructors, in turn, can create new exercises, and obtain insights into modeling behaviors and better understand the evolution of error types and their lifetime in the model. Currently, we have applied our tool it in the context of five modeling courses totaling around 300 students. To demonstrate the applicability of our framework we monitored and analysed student data from one of the courses with 26 students and the results show that *Model Judge* has potential to support process modeling learners in improving the quality of their models.

In this paper we apply our framework to the creation of process models. However, a similar approach could be taken for other types of models, like Universal Modeling Language (UML) class diagrams or UML activity diagrams, in settings where learners need to create a model starting from a textual

Manuscript received September 19, 2019; revised March 18, 2020; accepted MM DD, 2020. Date of publication MM DD, 2020; date of current version MM DD, 2020. This work was supported in part by the Spanish Ministry of Science, Innovation and Universities, under the grant TIN2017-86727-C2-1-R. (Corresponding author: Josep Carmona).

J. Sànchez-Ferreres, L. Delicado, J. Carmona, and L. Padró are with the Department of Computer Science, Universitat Politècnica de Catalunya, Spain (e-mail: jsanchezf@cs.upc.edu; ldelicado@cs.upc.edu; jcarmona@cs.upc.edu; padro@cs.upc.edu).

A. Abad Andaloussi and A. Burattin are with the Department of Applied Mathematics and Computer Science, Technical University of Denmark, Denmark (e-mail: amab@dtu.dk; andbur@dtu.dk).

B. Weber is with the Institute for Computer Science, University of St. Gallen, Switzerland (e-mail: barbara.weber@unisg.ch).

G. Calderón-Ruiz is with the Department of Physics and Formal Sciences and Engineering, Universidad Católica de Santa María, Peru (e-mail: gcalderonr@gmail.com).

Digital Object Identifier: Add doi here when available

description.

The remainder of this paper is structured as follows: Section II gives an overall description of the *Model Judge* platform and its features. Section III explores the related work in the existing literature. Section IV provides the preliminaries for our work. Subsequently, Section V presents the specific modeling guidelines that were used in the design of *Model Judge* as well as the details behind the implementation of the platform. Finally, Section VI evaluates *Model Judge*'s performance in a modeling course before Section VII concludes the paper.

II. A TOUR THROUGH THE MODEL JUDGE

This section describes the main features that users can find in the *Model Judge*. The *Model Judge* is a web-based, platform, that can be accessed through any web browser at <https://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 exercises in an agile way.

To use *Model Judge*, a user (being student or instructor) needs to create an account, with its associated storage space. In this space, a student can save models, which will be always accessible once she logs in. On the other hand, instructors have the ability to create courses and exercises. A course is designed as a set of exercises the students must solve, which can be taken from the existing list of examples, or created by the instructors.

Registered students can attempt to solve any exercise for the courses they are enrolled to. This enrolling process is performed by introducing a course code provided by the instructor. Upon selecting an exercise, the main working zone will be displayed (see Fig. 1): on the left part, a textual description of the process is provided, so that the student understands the process to model. Next to the textual description, a model editor allows the students to create the process model. The students enrolled in a particular course may enable the platform to record every few minutes a copy of their work-in-progress. This can be used for analyzing their behavior, which can be then reported back to both the student and the instructor.

During the modeling session, two important checks are available to help novice modelers:

- **Validation:** returns an aggregated diagnostic that reflects if the model has some errors (more details on the types of errors detected by *Model Judge* can be seen in Section V-C), however, the exact source of the error is omitted from this message. The motivation for this check is to allow for a mild test to guide the students. This is shown at the right part of Fig. 1, using a traditional color code (green meaning positive facts, red meaning negative facts, and yellow meaning warnings).
- **Complete Validation:** This check provides a more thorough list of all the problems detected, which individually addresses each of the issues with the model. The motivation for this check is to allow an assessment similar to the one obtained if a teacher was correcting the model.

Two important monitoring features are included in *Model Judge* for instructors to get further insights from the modeling

sessions. On the one hand, instructors have access to a real-time dashboard during the modeling session that can be used to monitor the students' progress. This dashboard allows inspecting useful statistics like how much are students using the validation functionality, their overall progress measured in the number of error diagnostics, and the actual process models. This dashboard is shown in Fig. 2.

On the other hand, instructors have the option to access the data stored from students of the course once their modeling task is finished. This data contains the final models, which can be used for assessment, but also the intermediate models and their diagnostics in order to perform a more detailed analysis with the progression of the students during the exercise, similar to what is shown in Section VI.

Furthermore, an exercise editor is available for instructors, in order to create new exercises. For this, instructors must provide both the problem statement text and its corresponding annotations, which are used to describe the exercise solution to *Model Judge* (see Section V). An on-line exercise editor is provided which handles the generation of partial annotations and allows its further refinement, as seen in Fig. 3. Typically, the necessary work to create a new exercise consists of removing the automatically highlighted elements that are not relevant from the process perspective, as well as manually annotating any activities that were implicitly described. From our experience with the tool, we have estimated that a trained instructor can create a new exercise for a well-known process in less than an hour.

In summary, the *Model Judge* platform consists of three inter-related tools, each focused on a different aspect of an exercise's lifecycle. We next detail the ecosystem behind our platform, with the main functionalities of every tool.

1) *MJ-Core:* *MJ-Core* is the main tool in the *Model Judge* platform. Its main purpose is to present a space for students to solve exercises in, and instructors to create and manage their courses. Additionally, the core platform provides the login and registration capabilities for all other applications in the platform. The source code for this tool is available inside the *modeljudge* project at our source code repository [13].

2) *MJ-Editor:* *MJ-Editor* is a tool aimed at instructors that allows the creation of new exercises for the core platform. In order to create a new exercise, a natural language description of a process must be annotated. This editor, shown in Fig. 3, is a standalone web application which can also be used offline. This editor is built using the React framework for Javascript, and its source code is also available online inside the *atd-editor* project at our source code repository [13].

3) *MJ-Dashboard:* *MJ-Dashboard* is a tool that helps instructors to follow their existing courses by summarizing student and exercise data. The tool is built using a custom framework on top of the *Compojure* library. Currently it is only available to instructors, due to the advanced nature of the information displayed.

III. RELATED WORK

There exist very few works in the literature that jointly propose a learning framework for process modeling, and

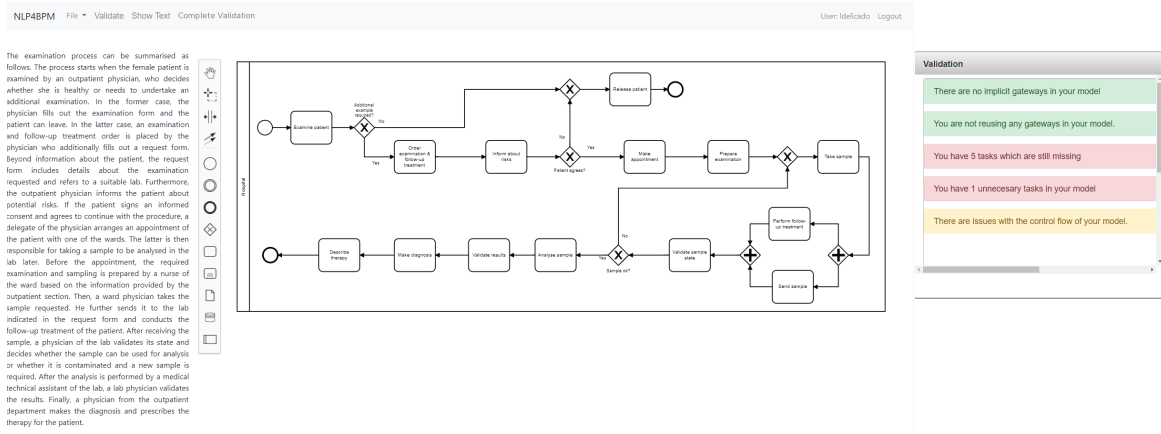


Fig. 1. The Model Judge: (Left) The textual description, (Center) the created model, (Right) The result of a validation.

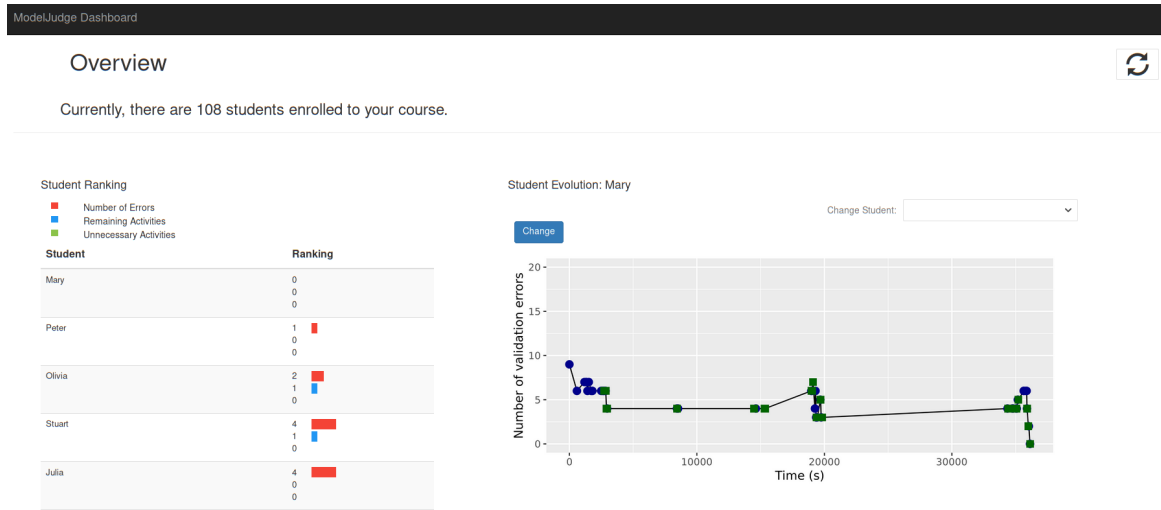


Fig. 2. The Model Judge Dashboard: (Left) ranking of students according to the gathered diagnostics (left), (Right) selected student evolution.

exploit NLP capabilities for automating the validation of created process models. Still, we can find existing examples of learning frameworks designed to aid students in conceptual modeling, such as the one presented in Pimentel *et. al*, where a gamification setup is defined to engage students in the task of creating *i** models [14]. Another are the works presented in Bogandova *et. al* and Buchmann *et. al*, which aim to create a taxonomy of common student errors when learning conceptual modeling [15], [16].

There have also been prior efforts to design e-learning systems for modeling notations, such as UML [17]. In this system, a very shallow textual validation for the main model components is done, and it is required to decide a priori all the possible solutions for a given exercise. Our framework is much more flexible, so that the space of solutions is merely defined by the problem statement itself. NLP applied to both text and model enables to implicitly account for all possible correct solutions without enumerating them explicitly.

There is related work that explores the human behavior in modeling, an aspect that influences the work of this paper. In Störrle *et al.*, a study is conducted to evaluate how human modelers visually parse UML diagrams using eye-tracking technology [18]. A more recent study follows a similar approach for the case of reading process models during a modeling session [19]. Incorporating insights arising from these studies, as well as well-known principles and guidelines, can be used to automatically improve the readability of a model [20].

Finally, the interaction between NLP and Business Process Management has some prior studies outside the scope of education. There has been work addressing the problem of the translation from textual to graphical representations of models (e.g., UML diagrams [21], [22], or Business Process Model and Notation (BPMN) [23], [24]) as well as schema [25] or process model [26] matching. The alignment technique presented in this paper is an adapted version of a prior work

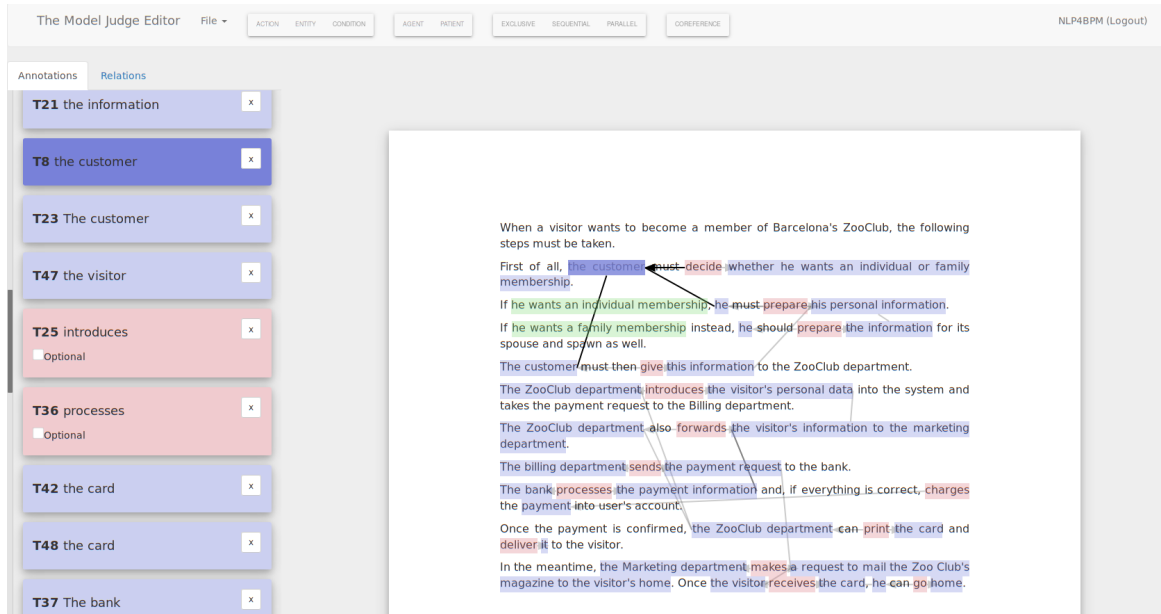


Fig. 3. The *Model Judge* Exercise Editor: the text describing the process (right) can be annotated (left) and relations between annotations reported (arcs in the text).

[12].

To the best of our knowledge, the proposal of this paper is the first one to tackle the challenges in the development of an automated system to support novices in the creation of business process models. Commercial tools (e.g., Signavio Process Manager) are already capable of providing feedback concerning some predefined modeling conventions. This feedback, however, is typically quite elementary and completely domain agnostic. Instead, the framework proposed is inspired on code judges for programming, which are extremely successful on guiding novice programmers towards obtaining solutions to programming exercises [7]–[11] by providing suggestions specifically tailored to the context of the exercise. As it is done in some of the aforementioned learning environments (e.g., Petit *et al.* [11]), in our framework we allow for a detailed diagnostic list, enabling the modeler to easily locate errors.

IV. PRELIMINARIES

A. Contextual Example: Modeling from Textual Descriptions

To illustrate the kind of modeling task we address in our work, let us consider the process of patient examination based on the description provided by the Women's Hospital of Ulm. Both the text and the model are extracted from the paper by Cabanillas *et al.* [27]. A part of the textual description of this process is provided in Fig. 4. A novice modeler would be confronted to this text in order to perform the creation of the corresponding process model. Fig. 5 shows a possible solution to this modeling task, i.e., a process model in BPMN [28].

For readers not familiar with the BPMN language's features, a short description is provided in Section IV-B.

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 latter case, an examination and follow-up treatment order is placed by the physician. (...) 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 analyzed in the lab later. (...) 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. (...) Finally, a physician from the outpatient department makes the diagnosis and prescribes the therapy for the patient.

Fig. 4. Fragment of a textual description for a patient examination process.

B. Process Modeling Notations

Process models can be created using a variety of modeling languages, such as Petri nets, Event-Driven Process Chains (EPCs), and BPMN. Although we focus in BPMN, the contributions of this paper are independent of the specific notation used to define a process model.

In particular, we focus on BPMN 2.0, a notation created as standard for business process modeling. BPMN has three different kinds of elements. First, the main elements are the nodes in the diagram, which may belong to three different types: *Activities* (Fig. 6, a), which represents some task that is performed; *Events* (Fig. 6, b), which represent that something happens; and *Gateways* (Fig. 6, c), which split or join flow control according to their type: parallel tasks are defined

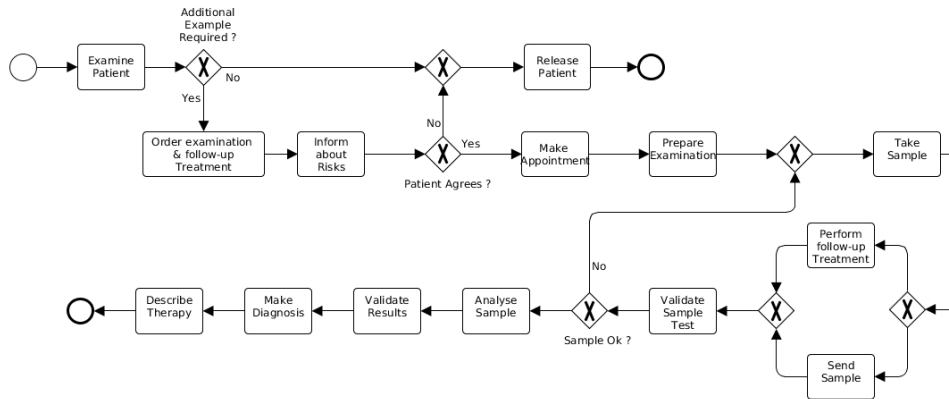


Fig. 5. Process model for the patient examination process.

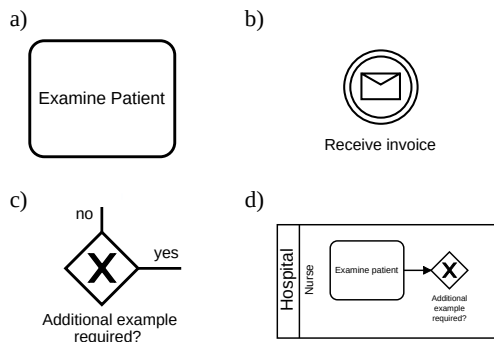


Fig. 6. Different constructs in the BPMN notation. Activity (a), Event (b), Gateway (c), Pool and Lane (d).

through the + gateway, whilst alternative paths are defined through the \times gateway. Second, the notation has different edges to connect nodes, so that control-flow dependencies can be defined. A solid line indicates the process workflow, while dashed lines represent messages sent between process participants. Finally, there are organization elements such as *lanes* that contain activities performed by the same participant, and *pools* (Fig. 6, d), that group several related lanes. Fig. 5 shows an example of the BPMN model corresponding to the patient examination process described in Fig. 4. Note that, for simplicity, in the process model of Fig. 5 we do not show pools or lanes.

C. Natural Language Processing and Annotation

In order to compare textual descriptions (like the one shown in Fig. 4) with a formal model, the text must be analyzed and converted to a structured representation that conveys the semantics of the text, and can be compared to the formal model.

To perform this step, we rely on existing tools produced by the NLP community, such as FreeLing [29]. The NLP analyzers are used to obtain a semantic representation of the

text. Particularly, we extract the *actions* being described in the text, the *agents* who perform each action, and the *patients* upon which each action is performed. Note that agent and patient terms are borrowed from linguistics, and the latter has no relation to the medical patients described in our running example. For instance, in Fig. 4 the sentence *an examination and follow-up treatment order is placed by the physician*, the agent is *physician*, the action is *place*, and the patient is *order*.

Additionally, we also extract per-word information such as the part of speech (PoS) tag and its ontological sense. By collecting this lower-level information, we allow our technique to fall back to the word level whenever the predicate level description is ambiguous or incomplete (e.g., detecting that word *place* occurs as a *verb* in a sentence is useful to discover that such action is being described, even if the information about the agent or the patient could not be extracted by the NLP tools).

More specifically, we apply a NLP pipeline consisting of tokenization, sentence splitting, morphological analysis, part-of-speech tagging, named entity recognition, word sense disambiguation, dependency parsing, semantic role labelling, and coreference resolution.

The steps up to word sense disambiguation provide word-level information such as the PoS (e.g., *placed* \rightarrow *verb past participle*), the lemma (e.g. *placed* \rightarrow *place*), or the sense (an entry in an ontology that will link to synonyms—e.g., *arrange for*— or other semantically close words—e.g., *request*, *order*, *ask for*).

The three last steps are of special relevance, since they allow the top-level predicate construction, and the identification of actors throughout the whole text: Dependency parsing identifies syntactic subjects and objects (which may vary depending, e.g., on whether the sentence is active or passive), while semantic role labeling identifies semantic relations (the *agent* of an action is the same regardless of whether the sentence is active or passive). Coreference resolution identifies several mentions of the same actor as referring to the same entity (e.g., in Fig. 4, *a delegate of the physician* and *the latter* refer to the same person, as well as the same object is mentioned

as *the sample requested* and *it*).

To compare formal models to textual descriptions, we will need annotated versions of those texts (i.e., enriched with all the linguistic information described above). Ideally, those annotations would be automatically performed by NLP tools, but given the limitations of the current NLP technology, we will resort to a certain amount of human annotation to improve the quality of our semantic representations. More details are provided in Section V-A.

Creating annotated versions of texts is usual in the NLP field, where many approaches are based on machine learning and require annotated text corpora both for training and for evaluating the performance of the developed systems. This need for annotated text has led the NLP community to develop several general-purpose annotation tools (e.g., Brat [30]). However, inspired on those resources, we have developed our own annotation tool tailored to *Model Judge*'s use case in order to offer a better user experience. This tool is also able to automatically infer partial annotations, as seen later in Section V-A.

V. FRAMEWORK UNDERLYING THE MODEL JUDGE

By observing the grading process of several process modeling courses, we have established a set of diagnostics that are suitable for being computed automatically. As done in the literature [2], we have split these diagnostics in three different categories: *Syntactic* diagnostics consider the control flow of the model. *Pragmatic* diagnostics consider the interpretability aspects of the model. Finally, *semantic* diagnostics check if there is no missing information from the underlying process and all the information provided is relevant and valid. Note that we use the terms *syntactic*, *semantic*, and *pragmatic* as it is usually done in the Process Modeling field, which differs from the meaning these terms have in linguistics and NLP.

This section describes our proposed framework for the automatic computation of diagnostics. The *Model Judge* produces a list of recommendations that the student can generate upon demand during the modeling process. Fig. 7 shows an overview diagram. The inputs to the system are the student's *Process Model* and the *Textual Description*, in natural language, that is being used as the exercise's statement.

The textual description is analyzed by the *Text Annotation* module, to produce a structured annotated description where the relevant concepts in the process model are identified. This annotation is then completed with human intervention to include domain expertise not available to NLP tools. After these steps—performed only once when the instructor creates a new exercise—students can work with the exercise.

In order to produce the diagnostics for a student-built model, the *Alignment* between the student process model elements—such as activities or gateways—and the annotations in the text is computed. Then, this alignment is used to compare the process model with the textual description in the *Diagnostics* module. In the remainder of this section we describe the aforementioned modules in detail.

A. Formalizing Annotated Textual Descriptions

Using natural language alone for the formal description of process models has some drawbacks. On one hand, the lack of structure in natural language text makes automatic analysis more difficult. On the other hand, the inherent ambiguity of written text makes it difficult to establish which parts of the text are relevant to describe the process.

NLP 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 Fig. 4, 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 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 [31], 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. This is why in Fig. 7 we incorporate a stage on human annotation, so that these ambiguities can be resolved.

In order to bridge the gap between the textual descriptions and the formal process models, we introduce the concept of *Annotated Textual Descriptions* (ATD). The goal of this representation is to use text annotations, typically found in the context of textual corpus labeling, to define a structured representation of text that still benefits from the flexibility and interpretability offered by natural language. Annotations are used as an intermediate language, and are generated in two steps: First, an automatic NLP tool generates a candidate annotation. Afterwards, the annotations are refined by a human (in our case, the instructor) to refine the annotation, e.g., discard unnecessary labels, or add missing or relevant details using domain expertise.

Even though obtaining accurate annotations requires some human intervention, our e-learning framework for self-assessment still enables for automation, since the annotations only need to be made once for an exercise, and from that moment on, they can be used to correct any number of proposed models for that exercise.

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.

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

Next, we describe each of the annotation and relation types

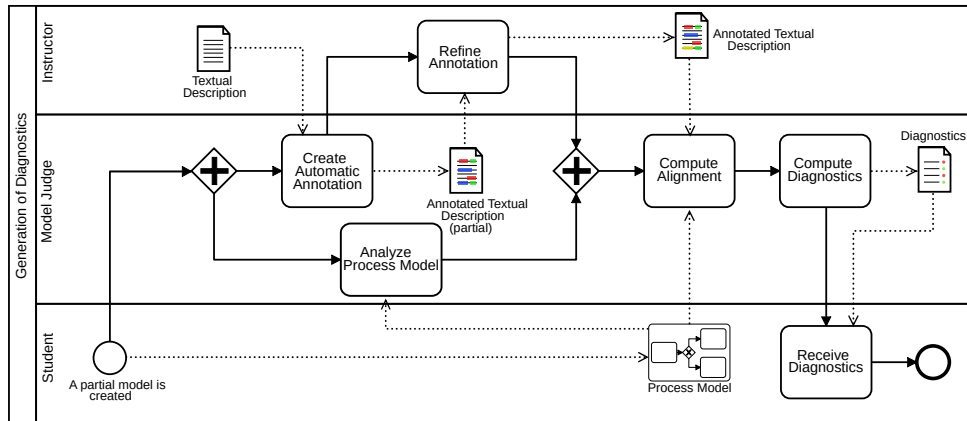


Fig. 7. Overview of the *Model Judge* framework to support the creation of process models.

that we consider for ATDs. Table I shows an example of an ATD which we will use for the remainder of this section.

1) *Annotations*: Annotations are used to mark an important region of the text. In the case of ATDs 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. We additionally allow to specify an action as optional. The *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* (α_8) 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 our framework, 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 can be considered correct.

Role. 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. Roles are associated with swimlanes in business processes (see Fig. 6 (d)).

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 the business process, 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 pre-requisite for some part of the process being executed, such as the patient *being healthy* (α_7). Conditions are typically associated with exclusive gateways, their labels and their surroundings.

2) *Relations*: Relations represent binary relationships between annotations. In ATDs, we consider the following relations. Note that the terms *agent* and *patient* have been chosen to highlight that the relations are consistent with the linguistic definition of the sentence semantic roles, as typically seen in linguistics (see Section IV).

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. A *role*, or *business object* 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* something.

Control Flow. The *sequential*, *exclusive* and *parallel* binary relationships, which are borrowed from *behavioral profiles* [32], are used to indicate the order between *actions* 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) to—i.e., happens before—*validates* (α_3)

Coreferences. A *role* is a coreference of another *role* when they refer to the same 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.

B. Alignment of Models and Annotated Texts

The goal of aligning a process model and a textual process description is to identify correspondences between the elements of the process model and parts of the textual process description [12], [33], [34]. In particular, all elements that

	Type	Text Fragment
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 I
EXAMPLE FRAGMENT OF ANNOTATED TEXTUAL DESCRIPTION WITH ANNOTATIONS (LEFT) AND RELATIONS (RIGHT)

describe the flow in a process model, i.e., *activities*, *events*, and *gateways*, are aligned to (parts of) the sentences of a textual process description. For instance, the sentence containing the text “... a delegate of the physician arranges an appointment of the patient with one of the wards.” would be aligned with the task “Make Appointment” of the BPMN model. Similarly, the gateway with the decision “Patient Agrees ?” must be mapped to the sentence *If the patient signs an informed consent ...*. This previous example clearly shows the challenge of computing alignments when textual descriptions are not annotated. Fig. 8 shows an example alignment for an initial fragment of our running example.

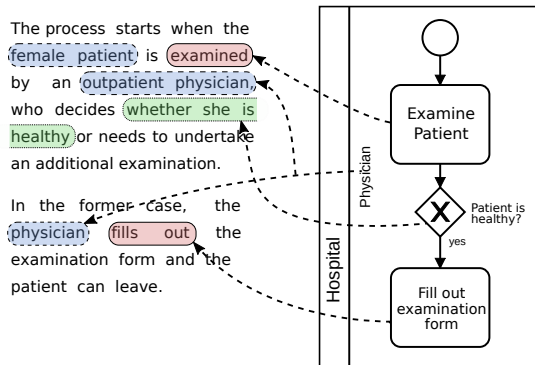


Fig. 8. Alignment between annotated textual description and BPMN process model.

In *Model Judge*, establishing the correspondence between the annotations in the annotated 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 this paper we extend our alignment framework presented in earlier works [12], [34] by assuming that textual descriptions are annotated. This assumption allows strengthening some of the constraints in the algorithm by having the certainty that information that comes from annotated texts is precise and

complete.

An overview of the details behind of our alignment technique can be seen in Fig. 9: (i) First, linguistic features from the textual description and the process model are extracted to map both representations into a canonical form in a *Feature Extraction* step. During this step we extract high-level information such as word senses (using WordNet) and semantic roles. In order to do this, we apply the full NLP pipeline described in Section IV-C. (ii) We can then use well-known similarity functions such as the Jaccard Index or Cosine Similarity to compare the feature vectors in the *Similarity Computation* phase. (iii) After that, the similarity information is encoded in an optimization algorithm, implemented as an Integer Linear Program, to find a valid alignment which maximizes similarity. (iv) Finally, in order to detect unnecessary activities, i.e., activities that are present in model but not in the text, we use *predictors*, adapted from Van der Aa *et al.* [33].

We refer the interested reader to the aforementioned work for a detailed description of the techniques.

C. Automatic Generation of Diagnostics

The core of our approach consists on the automatic generation of diagnostics. This computation uses the information from the *annotated textual description*, the *process model*, and the *alignment* between them. In this section we detail which diagnostics are currently generated by our tool and how they are computed.

1) *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. Illustrative examples for the presented syntactic errors can be found in Fig. 10.

Gateway Reuse and Implicit Gateways.

(Fig. 10, a and b) 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

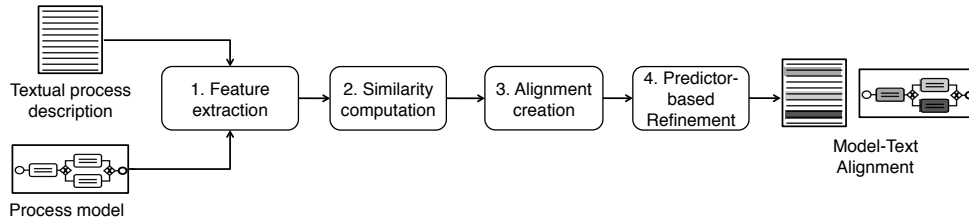


Fig. 9. Overview of alignment between textual descriptions and process models.

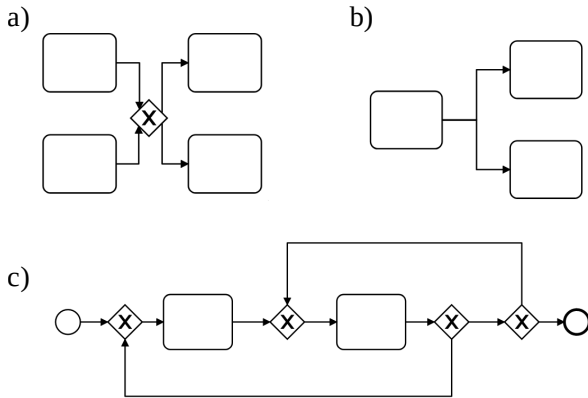


Fig. 10. Examples of several syntactic errors: Gateway Reuse (a), Implicit Gateway (b), Non-natural Loop (c).

and implicit gateways is a well-known best practice in business process models [6], [20], [35].

Non-Natural Loops. (Fig. 10, c) 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 *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* [36], which guarantees the process model is free from livelocks, deadlocks, and other anomalies that can be detected without domain knowledge.

Notice that the last two syntactic diagnostics can also be applied to other languages, like Petri nets. The computation of syntactic diagnostics only requires the process model, and they can be 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* [37].

2) *Pragmatic Diagnostics:* Pragmatics in process modeling can be described as the interpretability of the model, which can be impacted by factors such as complexity, modularity, secondary notation or activity labelling style [38]. It is that last aspect of pragmatics that *Model Judge* is focused on. 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 Mendling *et al.* [20], G6), in order to avoid ambiguous phrasing [39]. 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, and not contain details about the control flow of the process. A simpler model label using the *verb-object* rule for the aforementioned sentence would be “*Analyse sample*”.

Previous studies [39]–[42] 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.

The approach we present for checking adherence to a writing style is based on using a custom NLP parser for labels. The core of this technique consists of parsing the labels with a custom-made context-free grammar. This grammar is able to recognise most label syntactic structures defined by Pittke *et al.* and Leopold *et al.* [41], [42], and is built in such a way that the root node of the parse tree will be labeled with the identified writing style. For example, a label written in the *noun-action* style will have this information reflected at the root node of the tree. Parsing is then done as follows:

- 1) The n most likely part-of-speech (PoS) sequences of the label are computed.
- 2) For each of the part-of-speech sequences, the label is parsed using a *chart parser* with the aforementioned context-free grammar.
- 3) If all the parse tree roots are found to have an undesired style or none of the parse trees could be parsed by the grammar, the label writing style is not correct.

For instance, let us consider the activity label *sample patient*. The possible PoS sequences are $\langle noun, noun \rangle$ (i.e., a sample patient) and $\langle verb, noun \rangle$ (i.e., to sample a patient). After trying to parse the text considering each sequence, we see that the most likely sequence, $\langle noun, noun \rangle$, does not match any valid label pattern. Thus, the second sequence is chosen, which matches one of the valid *action* patterns. We can then say that the label’s writing style is correct, while if no valid pattern is found in any of the possible sequences, the label is considered to have an incorrect style.

In our implementation, patterns of labels describing double actions are also detected (e.g. *Close Ticket and Inform the Manager*) and, in this case, the system suggests the student to create two separate activities. It is worth noticing that this technique can be adapted so any instructor can enforce a label writing style in a flexible way, just defining a simple grammar to detect label patterns, and accept or reject the labels depending on the matched pattern.

Furthermore, this analysis helps avoiding common issues with general-purpose NLP tools when faced with the kind of text in business process models. For example, a general parser lacks the necessary context to infer that activity labels should always describe an action. As we can see with *sample patient*, adding knowledge about the context of a sentence into the parser solves the issue.

3) *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. For example, if the text states that an activity A must happen before an activity B can be executed, it is incorrect to model them as two separate branches of a choice. If a temporal relation 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.

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 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 aligned to some text action. If the similarity between them is low enough the activity is considered unnecessary, as there is no good match for the activity in the text.

Coverage of roles is performed similarly to activities. In this case, the similarity function is used to assess the similarity between *role* annotations and the process model's swimlanes.

Control flow validations have a preliminary implementation using the predictor technique described in the aforementioned prior work [12], [33]. However, there is still room for substantial improvements in this regard.

VI. EVALUATION

The goal of this section is to provide an evaluation of the *Model Judge*. More specifically, we aim at investigating the following questions:

RQ1. Is the *Model Judge* perceived useful and accurate by the students? This question is answered in Section VI-A;

RQ2. How are the *Model Judge* diagnostics and validations associated with actual modeling errors? This question is answered in Section VI-B1;

RQ3. How does the process of process modeling evolve over time, when the *Model Judge* is available? This question is answered in Section VI-B2.

To answer the above research questions, data from a modeling session at the Technical University of Denmark (DTU) was analyzed. During the modeling session 26 students were asked to individually create a process model given a textual description using the *Model Judge* (cf. Section II). There was no time limit imposed. Students had a validation functionality available during model creation. The diagnostics generated by the validation functionality provided some indication about the type of error (e.g., missing activity), but did not reveal the exact source of the error. Moreover, *Model Judge* provides a complete validation functionality offering more detailed diagnostic feedback. Students were instructed to use the complete validation only at the end of the modeling process, once they have completed the modeling. This way, during the modeling students were incentivated to find out the reasons why certain types of errors arised, without actually knowing the exact errors. We believe this facilitates the learning process. It is important to note that in our experimental setting the conditions for having a control group were not met.

A. Subjective Perception (RQ1)

To answer **RQ1** a survey was conducted at the end of the modeling session. Students were asked to assess the validation and complete validation functionalities of *Model Judge* in terms of perceived *accuracy* and *usefulness*. Moreover, students were asked to write down any complaints and/or improvement suggestions concerning *Model Judge*. Overall, 18 out of 26 students participated in the survey.

Fig. 11 shows the results of the survey: Our results demonstrate that 67% (12 out of 18) found the validation functionality useful (i.e., strongly agree or agree). Similarly, 67% (12 out of 18) perceived the complete validation useful.

In terms of accuracy, 44% (8 out of 18) perceived the validation functionality of the *Model Judge* as accurate (i.e., strongly agree or agree). For the complete validation the agreement was with 56% (10 out of 18) slightly higher.

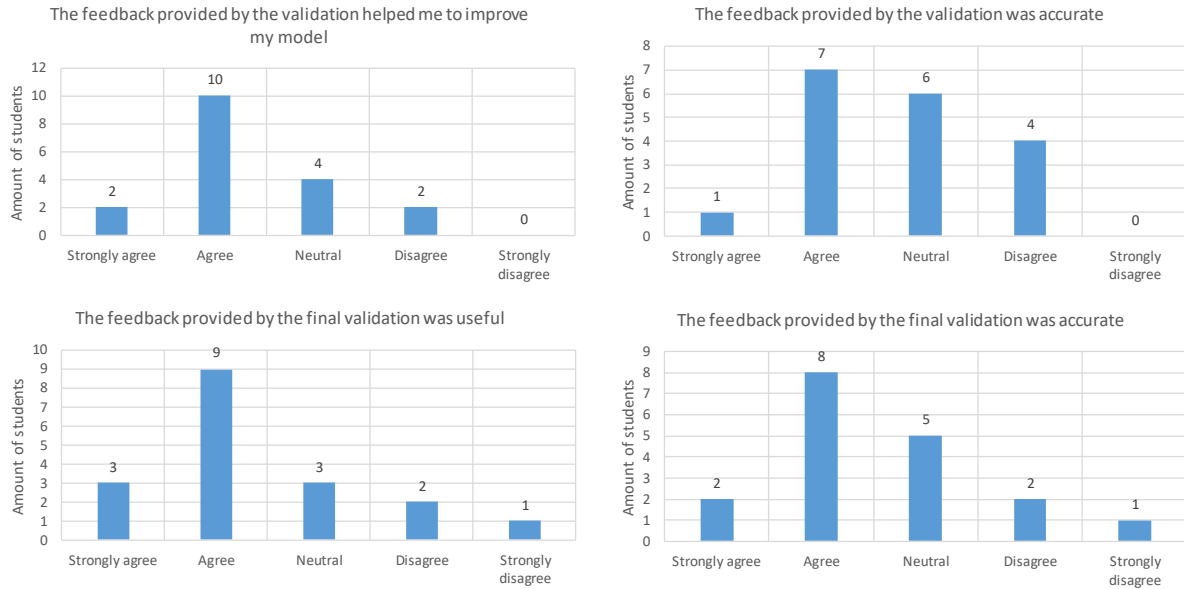


Fig. 11. Results of the student survey for the modeling courses. Each chart reports the results referring to a different question asked.

Additionally, it can be noted that the perceptions concerning usefulness were better than for accuracy. This is supported by the written feedback provided by the students. While in general, students appreciated the support provided by *Model Judge*, some students pointed out that the tool in did not recognize all their process model labels. In addition, some students stated that a more precise feedback would have been useful (e.g., “*I disliked the feedback [provided by the validation functionality]. I could not know what was wrong, just that something was.*”).

Moreover, it can be noted that the complete validation functionality is perceived as slightly more useful and accurate when compared to the validation functionality. This is not surprising, since the complete validation functionality provided a more detailed feedback to the students.

In general, we can answer research question **RQ1** partially positively, i.e., *Model Judge* is perceived by the majority of the students as useful but, concerning accuracy, the majority agrees that just the complete validation is accurate.

B. Analysis of the Modeling Session Data (RQ2, RQ3)

To answer research question **RQ2** and **RQ3** we analyzed the recordings of the modeling session.

Data Collection. For every student, we stored periodically (every minute) information for the whole modeling session. Additionally, information was also saved each time the user performed a simple or complete validation. In particular, we recorded a total of 1584 intermediate models for 26 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; and (v) The validation results of our tool for the particular process model. Note that

the validation results were computed for all snapshots, despite the fact that the students only saw the ones they explicitly requested.

The dataset used to answer RQ2 and RQ3 is available online at the PADS-UPC research group website [43].

1) *Association of the Tool on Diagnostics with Modeling Errors (RQ2):* In this section, we provide an answer to **RQ2**: we analyze the association between diagnostics of the *Model Judge* and actual modeling errors.

The actual modeling errors were derived based on the evaluation criteria agreed by two researchers, a Ph.D student and an associate professor affiliated to different institutions. Both assessors have teaching experience in business process management: The Ph.D student has contributed in the teaching of the course for 2 consecutive years, while the associate professor has been teaching the course for the past 7 years. Additionally, both researchers are familiar with evaluating process models derived by students. The criteria were set based on the guidelines provided by the SEQUAL [44] and 7PMG [20] frameworks. All the covered criteria have been discussed by both assessors before proceeding with the evaluation of the models.

We analyzed to what extent the provided diagnostics (identified by the *Model Judge*) are associated with modeling errors (identified by manual inspections). In terms of model diagnostics we considered: D_{avg} = “Average number of bad diagnostics during the modeling session,” D_{end} = “Number of bad diagnostics at the end of the exercise” and in term of modeling errors we considered: E_m = “Errors as missing activities,” E_c = “Errors in the organization of the control-flow,” and E_t = “Errors in the model semantics.”

Results, expressed as Spearman’s correlation tests, between diagnostics and actual errors are reported in Table II. The

		D_{avg}	D_{end}
E_m	Spearman's ρ	.502	.576
	p -value	.009	.002
E_c	Spearman's ρ	.476	.569
	p -value	.014	.002
E_t	Spearman's ρ	.445	.520
	p -value	.023	.007

TABLE II
SPEARMAN'S CORRELATION TEST RESULTS BETWEEN DIAGNOSTICS
AND ACTUAL ERRORS

		D_{avg}	D_{end}
V_s	Spearman's ρ	-.724	-.763
	p -value	< .001	< .001
V_c	Spearman's ρ	-.607	-.687
	p -value	< .001	< .001

TABLE III
SPEARMAN'S CORRELATION TEST RESULTS BETWEEN DIAGNOSTICS
AND NUMBER OF VALIDATIONS

correlation between the number of bad diagnostics and the different errors are moderate and all significant. This suggests that the *Model Judge*'s diagnostics is capable of approximating to a certain extent the actual modeling errors.

Starting from the previous results, we additionally studied the presence of correlations between the number of validations performed by the students and the number of bad diagnostics obtained. To that end, we computed a Spearman's correlations between V_s = "Number of validations," V_c = "Number of complete validations," D_{avg} (average bad diagnostics), D_{end} (bad diagnostics by the end). Besides the obvious correlations $V_c \sim V_s$ and $D_{avg} \sim D_{end}$, we also found significant and strong negative correlations between the two pairs of variables, as seen in Table III. This means that, as the number of validations grows, the number of bad diagnostics decreases.

In another analysis, we correlated V_s (i.e., number of validations) and V_c (i.e., number of complete validations) with the number of modeling errors observed by the end: E_m (errors as missing activities), E_c (errors in the organization of the control-flow), E_t (total semantics mistakes). The results are reported in Table IV and, as for the previous cases, all correlations are significant as well as negative. This means that as the number of validations grows the number of actual errors decreases. Based on previous results, this should not surprise: we already observed that diagnostics inversely correlate with

		V_s	V_c
E_m	Spearman's ρ	-.487	-.430
	p -value	.012	.028
E_c	Spearman's ρ	-.576	-.547
	p -value	.002	.004
E_t	Spearman's ρ	-.457	-.441
	p -value	.019	.024

TABLE IV
SPEARMAN'S CORRELATION TEST RESULTS BETWEEN ERRORS AND
NUMBER OF VALIDATIONS

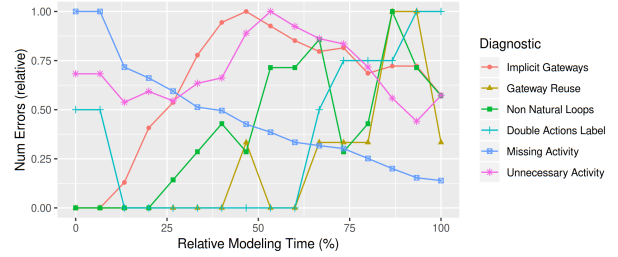


Fig. 12. Evolution of the diagnostic types for the modeling session.

the number of validations in (see Table III) and that diagnostics and errors positively correlate (see Table II).

Finally, to answer research question **RQ2**, we can conclude that both the diagnostics and the validation capabilities of the *Model Judge* are associated with actual modeling errors. In particular, there are correlations between *Model Judge*'s diagnostics and actual errors. We also observed negative correlations between the number of validations and diagnostics and between number of validation and actual errors (which mean that an increase in the number of validations is associated with less errors).

2) *Evolution of the Modeling Sessions (RQ3)*: The final investigation we performed aimed at understanding how the process modeling evolves when the *Model Judge* is available (**RQ3**). For this, we analyzed after the modeling course ended the collected snapshots with the goal to observe the evolution of the number of validation errors during the modeling sessions. Note that, for this analysis, we considered individual modeling sessions, with some students having performed more than one session.

In a first investigation, we analyzed how the frequency of different diagnostics varies during the sessions. Fig. 12 shows the evolution of the average amount of diagnostics for all students, per diagnostic type. To better observe the relative behavior 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. "Missing Activity" diagnostics decrease as the session advances, since less activities will be missing as the modeling session progresses. "Unnecessary Activity" and "Implicit Gateway" 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 behavior, but still increase for the duration of the session. This can be explained by the fact that, as the modeling 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 these errors until the end of the modeling session. This is in line with existing research that modelers differ in

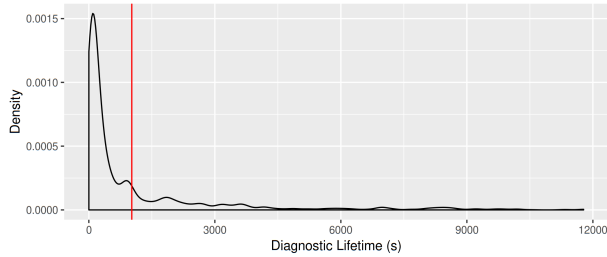


Fig. 13. Density plot of the diagnostic lifetimes variable.

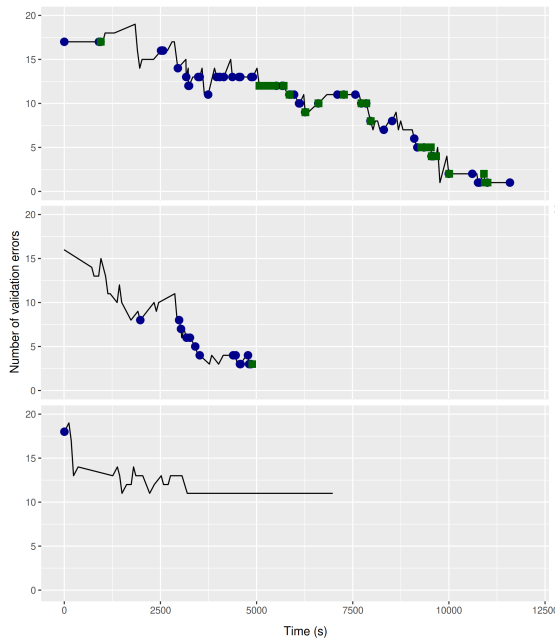


Fig. 14. Three characteristic behaviors observed in the modeling sessions. The blue circles represent simple validations, while green squares denote complete validations.

term of their validation behavior [45].

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 (Fig. 13). 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.

Finally, by manual inspecting the session data, we identified some “modeling profiles,” i.e., typical approaches followed by students while solving their modeling task. Fig. 14 shows a representative for each of the identified profiles, when plotting number of validation errors vs. time (in seconds) together with

simple and complete validations. Note that there are a few outlier sessions that do not match any of the three profiles shown. An instructor can obtain a good overview of the students evolution by looking at these student’s plots. These are the three profiles we identified: (i) The first group (on top of Fig. 14) is composed of modeling sessions where students frequently use the intermediate and complete validation functions, and ended up with almost no bad diagnostics. This group corresponds to 26.0% of the sessions; (ii) The modeling sessions from the second group (in the middle of Fig. 14) correspond to students who frequently use the validation, however, they 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 59.3% of the students; (iii) The modeling sessions from the third group (at the bottom of Fig. 14) correspond to students who 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 14.8% of the students.

In this section we answered **RQ3** by examining how the modeling sessions evolve when the *Model Judge* is used: we observed the distribution of the diagnostics as well as the density plot of the diagnostic lifetimes. Additionally, we identified three typical modeling profiles that can be used by instructors to gather some initial understanding on the modeling approach followed by students.

VII. CONCLUSION

In this paper we provide both the framework and an evaluation of the *Model Judge*. The framework is grounded in the use of NLP techniques together with an algorithm to align textual descriptions and graphical process models notations such as BPMN.

Our experience of applying *Model Judge* in different universities shows that it can be easily incorporated in a modeling course, where novice modelers can benefit from an environment that produces continuous support in the task of creating a process model. As for instructors, they are able to better support students by monitoring modeling sessions and easily create new exercises to fit their needs.

As future work we plan to extend the capabilities of the framework (multilingual support, expand the type and quality of diagnostics, among others), and apply it in more courses so that more conclusions can be drawn on the data gathered.

ACKNOWLEDGMENT

This work has been partially supported by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R.

REFERENCES

- [1] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management, Second Edition*. Springer, 2018, doi: 10.1007/978-3-662-56509-4.
- [2] J. Krogstie, G. Sindre, and H. D. Jørgensen, “Process models representing knowledge for action: a revised quality framework,” *EJIS*, vol. 15, no. 1, pp. 91–102, Dec. 2006, doi: 10.1057/palgrave.ejis.3000598.

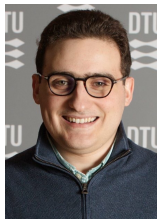
- [3] O. I. Lindland, G. Sindre, and A. Solvberg, "Understanding quality in conceptual modeling," *IEEE Softw.*, vol. 11, no. 2, pp. 42–49, March 1994.
- [4] J. Mendling, *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*, ser. Lecture Notes Business Inf. Process. Springer, 2008, vol. 6, doi: 10.1007/978-3-540-89224-3.
- [5] M. Wynn, H. Verbeek, W. van der Aalst, A. ter Hofstede, and D. Edmond, "Business process verification – finally a reality!" *Bus. Process Manage. J.*, vol. 15, no. 1, pp. 74–92, Feb. 2009, doi: 10.1108/14637150910931479.
- [6] C. Haisjackl, P. Soffer, S. Y. Lim, and B. Weber, "How do humans inspect BPMN models: an exploratory study," *Softw. & Syst. Model.*, vol. 17, no. 2, pp. 655–673, May 2018, doi: 10.1007/s10270-016-0563-8.
- [7] A. Kurnia, A. Lim, and B. Cheang, "Online judge," *Comput. & Educ.*, vol. 36, no. 4, pp. 299–315, May 2001, doi: 10.1016/S0360-1315(01)00018-5.
- [8] M. Joy, N. Griffiths, and R. Boyatt, "The boss online submission and assessment system," *ACM J. Educational Resour. Comput.*, vol. 5, no. 3, p. 2, Sep. 2005, doi: 10.1145/1163405.1163407.
- [9] A. Kosowski, M. Malafiejski, and T. Noinski, "Application of an online judge & contest system in academic tuition," in *Advances Web Based Learning - ICWL 2007, 6th Int. Conf., Edinburgh, UK, August 15-17, 2007*, pp. 343–354.
- [10] R. Singh, S. Gulwani, and A. Solar-Lezama, "Automated feedback generation for introductory programming assignments," in *ACM SIG-PLAN Conf. Programming Language Design Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*, pp. 15–26.
- [11] J. Petit, S. Roura, J. Carmona, J. Cortadella, A. Duch, O. Giménez, A. Mani, J. Mas, E. Rodríguez-Carbonell, A. Rubio, J. de San Pedro, and D. Venkataramani, "Judge.org: Characteristics and experiences," *IEEE Trans. on Learn. Technol.*, 2017.
- [12] J. Sánchez-Ferreres, H. van der Aa, J. Carmona, and L. Padró, "Aligning textual and model-based process descriptions," *Data Knowl. Eng.*, vol. 118, pp. 25–40, Nov. 2018, doi: 10.1016/j.datak.2018.09.001.
- [13] PADS-UPC Research group. PADS-UPC source code repository on github. [Online]. Available: <http://github.com/PADS-UPC>
- [14] J. Pimentel, E. Santos, T. Pereira, D. Ferreira, and J. Castro, "A gamified requirements inspection process for goal models," in *Proc. 33rd Annu. ACM Symp. Applied Computing, Pau, France, April 09-13, 2018*. ACM, pp. 1300–1307.
- [15] D. Bogdanova and M. Snoeck, "Camelot: An educational framework for conceptual data modelling," *Inf. Softw. Technol.*, Feb. 2019.
- [16] D. Bogdanova, e. R. A. Snoeck, Monique, D. Karagiannis, and M. Kirikova, "Learning from errors: Error-based exercises in domain modelling pedagogy," in *Practice Enterprise Modeling, Vienna, Austria, October 31 - November 2, 2018*. Cham: Springer Int. Publishing, pp. 321–334.
- [17] B. Demuth and D. Weigel, "Web based software modeling exercises in large-scale software eng. courses," in *2009 22nd Conf. Software Engineering Education Training, Hyderabad, India, 17-20 Feb. 2009*, pp. 138–141.
- [18] H. Störrle, N. Baltzen, H. Christoffersen, and A. M. Maier, "On the impact of diagram layout: How are models actually read?" in *PSRC@ MoDELS, Valencia, Spain, September 28 - October 3, 2014*, pp. 31–35.
- [19] A. Burattin, M. Kaiser, M. Neuraüter, and B. Weber, "Eye tracking meets the process of process modeling: a visual analytic approach," in *Int. Conf. Business Process Management*. Springer, pp. 461–473.
- [20] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst, "Seven process modeling guidelines (7PMG)," *Inf. & Softw. Technol.*, vol. 52, no. 2, pp. 127–136, Feb. 2010, doi: 10.1016/j.infsof.2009.08.004.
- [21] F. Meziane, N. Athanasakis, and S. Ananiadou, "Generating natural language specifications from UML class diagrams," *Requirements Eng.*, vol. 13, no. 1, pp. 1–18, Sep. 2008.
- [22] I. S. Bajwa and M. A. Choudhary, "From natural language software specifications to UML class models," in *Enterprise Information Systems - 13th Int. Conf., ICEIS, Beijing, China, June 2011*, pp. 224–237.
- [23] H. Leopold, J. Mendling, and A. Polyvyanyy, "Supporting process model validation through natural language generation," *IEEE Trans. Softw. Eng.*, vol. 40, no. 8, pp. 818–840, May 2014.
- [24] F. Friedrich, J. Mendling, and F. Puhlmann, "Process model generation from natural language text," in *Adv. Information Systems Engineering - 23rd Int. Conf., CAISE 2011, Beijing, China, June 8-11, 2011*, London, UK, pp. 482–496.
- [25] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB J.*, vol. 10, no. 4, pp. 334–350, Dec. 2001. [Online]. Available: <http://dx.doi.org/10.1007/s007780100057>
- [26] U. Cayoglu, R. Dijkman, M. Dumas, P. Fettke, L. García-Bañuelos, P. Hake, C. Klinkmüller, H. Leopold, A. Ludwig, P. Loos, J. Mendling, A. Oberweis, A. Schoknecht, E. Sheerit, T. Thaler, M. Ullrich, I. Weber, and M. Weidlich, "Report: The process model matching contest 2013," in *Business Process Management Workshops - BPM Int. Workshops, Beijing, China, August 26, 2013*, Beijing, China, pp. 442–463. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-06257-0_35
- [27] C. Cabanillas, D. Knuplesch, M. Resinas, M. Reichert, J. Mendling, and A. R. Cortés, "Ralph: A graphical notation for resource assignments in business processes," in *Adv. Information Systems Engineering - 27th Int. Conf., CAiSE 2015, Stockholm, Sweden, June 8-12, 2015*, pp. 53–68.
- [28] T. Allweyer, *BPMN 2.0: introduction to the standard for business process modeling*. BoD–Books on Demand, 2010.
- [29] L. Padró and E. Stanilovsky, "Freeling 3.0: Towards wider multilinguality," in *Proc. 8th Int. Conf. Language Resources Evaluation, LREC 2012, Istanbul, Turkey, May 23-25, 2012*, pp. 2473–2479. [Online]. Available: <http://www.lrec-conf.org/proceedings/lrec2012/summaries/430.html>
- [30] P. Stenetorp, S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. Tsujii, "Brat: a web-based tool for nlp-assisted text annotation," in *Proc. Demonstrations 13th Conf. European Chapter Association Computing Linguistics, Avignon, France, April 23-27, 2012*. Association for Comput. Linguistics, pp. 102–107.
- [31] P. Mirza, "Extracting temporal and causal relations between events," Ph.D. dissertation, Int. Doctorate School Inf. Communication Technol., Univ. of Trento, Via Calepina, 14, 38122 Trento TN, Italy, 2016.
- [32] S. Smirnov, M. Weidlich, and J. Mendling, "Business process model abstraction based on behavioral profiles," in *Service-Oriented Computing*. Springer, 2010, pp. 1–16.
- [33] H. Van der Aa, H. Leopold, and H. A. Reijers, "Comparing textual descriptions to process models: The automatic detection of inconsistencies," *Inf. Syst.*, vol. 64, pp. 447–460, Mar. 2017.
- [34] J. Sánchez-Ferreres, J. Carmona, and L. Padró, "Aligning textual and graphical descriptions of processes through ILP techniques," in *Adv. Information Systems Engineering - 29th Int. Conf., CAiSE 2017, Essen, Germany, June 12-16, 2017*, pp. 413–427.
- [35] V. Bernstein and P. Soffer, "Identifying and quantifying visual layout features of business process models," in *Enterprise, Business-Process Information Systems Modeling - 16th Int. Conf., BPMDS 2015, 20th Int. Conf., EMMSAD 2015, Held at CAiSE 2015, Stockholm, Sweden, June 8-9, 2015*, pp. 200–213.
- [36] W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn, "Soundness of workflow nets: classification, decidability, and analysis," *Formal Asp. Comput.*, vol. 23, no. 3, pp. 333–363, Aug. 2011, doi: 10.1007/s00165-010-0161-4.
- [37] T. Lengauer and R. E. Tarjan, "A fast algorithm for finding dominators in a flowgraph," *ACM Trans. Program. Lang. Syst.*, vol. 1, no. 1, pp. 121–141, Jan. 1979, doi: 10.1145/357062.357071.
- [38] C. Haisjackl, P. Soffer, S. Lim, and B. Weber, "How do humans inspect BPMN models: an exploratory study," *Softw. & Syst. Model.*, vol. 17, Oct. 2016.
- [39] H. Leopold, S. Smirnov, and J. Mendling, "On the refactoring of activity labels in business process models," *Inf. Syst.*, vol. 37, no. 5, pp. 443–459, Jul. 2012, doi: 10.1016/j.is.2012.01.004.
- [40] J. Mendling, H. A. Reijers, and J. Recker, "Activity labeling in process modeling: Empirical insights and recommendations," *Inf. Syst.*, vol. 35, no. 4, Jun. 2010.
- [41] H. Leopold, R.-H. Eid-Sabbagh, J. Mendling, L. Azevedo, and F. Baião, "Detection of naming convention violations in process models for different languages," *Decis. Support Syst.*, no. 56, pp. 310–325, Jun. 2013.
- [42] F. Pittke, H. Leopold, and J. Mendling, "Automatic detection and resolution of lexical ambiguity in process models," *IEEE Trans. Softw. Eng.*, vol. 41, no. 6, pp. 526–544, 2015.
- [43] PADS-UPC Research group. ModelJudge modeling session dataset. [Online]. Available: <http://www.cs.upc.edu/~pads-upc/ModelJudgeData.zip>
- [44] J. Krogstie, *Model-based development and evolution of Inf. systems: A Quality Approach*. Springer Sci. & Business Media, 2012.
- [45] J. Pinggera, "The Process of Process Modeling," Ph.D. dissertation, Univ. of Innsbruck, Dept. of Comput. Sci., Innrain 52, 6020 Innsbruck, Austria, 2014.



Josep Sánchez-Ferreres is a Ph.D. student at Universitat Politècnica de Catalunya, studying the relation between natural language and formal business process representations. His research interests include natural language processing, business process management and process mining but also Machine Learning and artificial intelligence. He has published papers in international conferences in the field (BPM, CAiSE).



Luis Delicado Luis Delicado received the M.S. degree in Computer Science mention in Computer Graphics at Universitat Politècnica de Catalunya. His research background combines two different fields: Business Process and Crowd animations, with his role of Software developer inside and outside of his research. He has different papers published at international conferences in both fields (BPM, MIG).



Amine Abbad Andaloussi is a Ph.D. candidate at the Technical University of Denmark. He received his M.Sc. degree from the same university in 2018. His main research is about the understandability of business process models. With an academic background in computer science and strong research interest in cognitive psychology, Amine investigates the way end-users engage with business process models in order to improve the understandability of process modeling notations and enhance the existing modeling tool-support.



Andrea Burattin is Associate Professor at the Technical University of Denmark, Denmark. Previously, he worked as Assistant Professor at the same university, and as postdoctoral researcher at the University Innsbruck (Austria) and at the University of Padua (Italy). In 2013 he obtained his Ph.D. degree from University of Bologna and Padua (Italy). The IEEE Task Force on process mining awarded to his Ph.D. thesis the Best process mining Dissertation Award 2012–2013. His Ph.D. thesis has then been published as Springer Monograph in the LNBIP series. He served as organizer of BPI workshop since 2015, and special sessions on process mining at IEEE CIDM since 2013. He is also in the program committee of several conferences. His research interests include process mining techniques and, in particular, online approaches on event streams.



Guillermo Calderón Ruiz receives his Ph.D. in Engineering Sciences in the field of Computer Science from the Pontificia Universidad Católica de Chile in 2011. He is a principal professor in Universidad Católica de Santa María in Arequipa-Peru, he is also the Director of the School of Systems Engineering since 2017 in the same university. He also runs a software development center in the same university developing projects for the organizations in the southern of Peru. His research interest include process mining, data science, business process management, business intelligence and information technology. He is member of the IEEE Task Force on Process Mining, and he is also in the program committee of several conferences.



Barbara Weber Barbara Weber is Professor for Software Systems Programming and Development at the University of St. Gallen, Switzerland. She is Chair for Software Systems Programming and Development and Director of the Institute of Computer Science. Barbara's research interests include process model understandability, process of process modeling, process flexibility, and user support in flexible process-aware systems as well as neuro-adaptive information systems. Barbara has published more than 150 refereed papers, for example, in Nature Scientific Reports, Information and Software Technology, Information Systems, Data and Knowledge Engineering, Software and System Modeling, and Journal of Management Information systems and is co-author of the book "Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies" by Springer.



Josep Carmona received the M.S. and Ph.D. degrees in Computer Science from the Technical University of Catalonia, in 1999 and 2004, respectively. He is an associate professor in the Computer Science Department of the same university. His research interests include formal methods and concurrent systems, data and process science, business intelligence and business process management, and natural language processing. He has co-authored around 100 research papers in conferences and journals. He received best paper awards at the Int. Conf. on Application of Concurrency to System Design (ACSD 2009), at the Int. Conf. on business process management (BPM 2013), and at the International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2016). He is the author of the book "Conformance Checking: Relating Processes and Models", by Springer.



Lluís Padró is Associate Professor at Universitat Politècnica de Catalunya. His research area is framed in artificial intelligence, specifically in natural language processing, and more particularly in the development of language analysers (morphological, syntactic, dependency, sense disambiguators, semantic role labelers, named entity recognisers, etc). He has published more than 100 papers in the main conferences (ACL, ANLP, NAACL, EACL, COLING, EMNLP, etc) and journals (Computational Linguistics, Machine Learning, Journal on Language Resources and Evaluation, etc) in the area. He has served as program committee for lead conferences and journals (ACL, EACL, EMNLP, ICML, JLRE, Neurocomputing, ...) He is the founder, administrator and main developer of project FreeLing (<http://nlp.lsi.upc.edu/freeling>), an open-source suite offering language analysis services in a variety of languages widely used both in academy and industry.

Appendix III

Aligning Textual and Graphical Descriptions of Processes Through ILP Techniques

- **Publication Type:** Conference paper
- **Published In:** International Conference on Advanced Information Systems Engineering (CAiSE 2017)
- **GGIS Rating:** A
- **Full Reference:** Josep Sànchez-Ferreres, Josep Carmona and Lluís Padró. “Aligning Textual and Graphical Descriptions of Processes Through ILP Techniques”. In: *Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CAiSE)*. 2017. pp. 413-427.

Aligning Textual and Graphical Descriptions of Processes Through ILP Techniques

Josep Sànchez-Ferreres, Josep Carmona, and Lluís Padró

Universitat Politècnica de Catalunya, Barcelona, Spain
{jsanchezf, jcarmona, padro}@cs.upc.edu

Abstract. With the aim of having individuals from different backgrounds and expertise levels examine the operations in an organization, different representations of business processes are maintained. To have these different representations aligned is not only a desired feature, but also a real challenge due to the contrasting nature of each process representation. In this paper we present an efficient technique for aligning a textual description and a graphical model of a process. The technique is grounded on using natural language processing techniques to extract linguistic features of each representation, and encode the search as a mathematical optimization encoded using Integer Linear Programming (ILP) whose resolution ensures an optimal alignment between both descriptions. The technique has been implemented and the experiments witness the significance of the approach with respect to the state-of-the-art technique for the same task.

Keywords: Process Models, Natural Language Processing, Integer Linear Programming

1 Introduction

Nowadays organizations store processes descriptions in various representations. The reason for this is the different nature stakeholders have: while textual descriptions of processes are well-suited for non-technical users, they are less appropriate for describing precise aspects of the underlying process [1]. In contrast, formal and graphical process notations (e.g., BPMN) are unambiguous representations which can be the basis for automating the corresponding processes within the organization [2], but they are oriented to specialized users. In this context, due to the evolving nature of processes, there is a high risk of having deviations between the different representations, a problem that may have serious consequences for any organization [3].

In the last decade, the field of *Natural Language Processing* (NLP) has grown up to a mature enough level, where the algorithmic support to analyze any text is high. Currently, there are several powerful open-source libraries that can be integrated easily to any software project, thus making linguistic analysis a reality in many contexts [4–7]. In this paper we exploit state-of-the-art NLP algorithms

to extract advanced linguistic features for the text found in both representations, so that the corresponding linguistic footprint can be mapped to a canonical form. Several similarity metrics can be defined on top of this canonical form, including weighted versions which may favor particular characteristics of process descriptions such as the action performed.

Once the similarity metric is chosen, the problem is casted as an optimization, whose solution(s) represent an assignment between tasks and sentences such that the accumulated sum of similarity is maximum. In particular, we encode the problem as an Integer Linear Programming (ILP) model whose resolution provides the optimal alignment between the text and the model.

The work of this paper is inspired by and shares the motivation of the seminal work [8, 9] (see Section 3 for an accurate comparison of both approaches). Remarkably, although the core algorithm for searching solutions of the techniques is very different from our approach's, the quality of both approaches is similar. However, due to the simplicity of the encoding proposed, the method proposed is much faster and can deal with model-text pairs of medium/large size in a feasible time, a crucial distinctive feature of our approach with respect to [8, 9]. By mapping the problem as an ILP, we clearly separate problem encoding from computation, thus allowing to easily incorporate new dimensions to consider (as we have done in this paper by incorporating actors). Notably, the technique provides a result very fast, thus widening the application scope from post mortem or batch analysis to real-time analysis.

The research method followed in this work is *Design Science* [10], which “creates and evaluates IT artifacts intended to solve identified organizational problems”.

The remainder of the paper is organized as follows: we provide a motivating example in the next section. Then, in Section 3 a detailed comparison with related work is reported. Preliminaries are then provided in Section 4, and the main contribution of the paper is presented in Section 5. Experiments on reference benchmarks are presented in Section 6. Finally, Section 7 concludes the paper and provides future lines for research.

2 Motivating Example

To give some intuition let us consider the example represented by the textual description and its corresponding BPMN model in Figure 1. The technique we present derives the correct alignment between sentences 1-13 and tasks A - P , except for task J , as this task is not mentioned in the text.

In the simpler cases, the correct alignment between a task and a sentence can be obtained just by comparing the words in the sentences and the task labels. In this work, we aim to expand on previous techniques by considering more information about the tasks in the form of features. To better illustrate this, let us consider sentences 6 and 8, which correspond to tasks G and L respectively. When performing the comparison only by looking at the task labels, there is no clear way to distinguish G from L since they have the same label. Because of

(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 enters the visitor's personal data into the system and takes the payment request to the Billing department. (7) The ZooClub department also forwards the visitor's information to the marketing department. (8) On receiving the request, the billing department also enters the visitor's personal data into their local database. (9) After that, the billing department sends the payment request to the bank. (10) The bank processes the payment information and, if everything is correct, charges the payment into user's account. (11) Once the payment is confirmed, the ZooClub department can print the card and deliver it to the visitor. (12) In the meantime, the Marketing department makes a request to mail the Zoo Club's magazine to the visitor's home. (13) Once the visitor receives the card, he can go home.

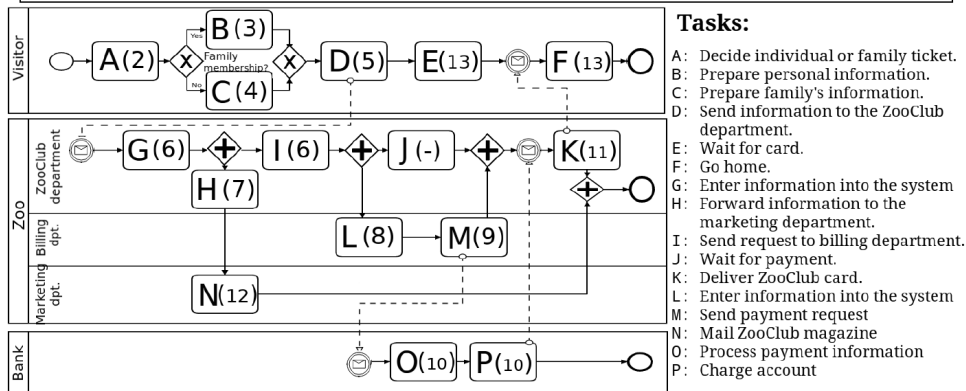


Fig. 1. Textual description and BPMN model of the Zoo business process. The correct alignment is displayed in parenthesis on the task labels.

that, there is nothing preventing both tasks from getting mapped to the same sentence. To correctly solve cases like this, semantic information is required, such as the fact that G and L are performed by different actors, so they should be assigned to sentences where the right actor is performing the task. Another helpful linguistic information can be obtained from gateways surrounding a task: Tasks following choice gateways are more likely to match sentences containing conditional statements.

3 Related Work

The contributions of this paper intersect with various works in the literature. In general, previous work can be categorized into transformations between models and text (e.g., [11, 12] for UML diagrams, or [1, 13] for BPMN), and schema [14]

or process model [15] matching. Also, there has been work on generating process models from group stories [16] and from use-cases [17], which are less related to this work since they restrict the form of the textual description used to describe the process. For the problem considered in this paper, the transformation approaches can only be applied when the source process description is unambiguous, and the transformation used does not modify the underlying process. Hence, the rest of the section considers only the work that computes alignments without requiring a transformation between process descriptions.

The seminal work [8, 9] proposed an algorithm for aligning textual descriptions and process models, with the particular aim of detecting inconsistencies between both representations. Their approach consists on using a linguistic analysis that derives a bag-of-words summary (i.e., resolving anaphoric references, extracting relevant clauses or removing prepositions) of the main elements in each representation. Then, a similarity computation between these elements is applied, and finally an optimal alignment which globally maximizes the similarity is computed, using a *best-first search* technique.

In our case, we extend the linguistic analysis with semantic role labeling, coreference resolution, and the computation of the semantic graph. Moreover, we encode the problem of computing an alignment as the resolution of an ILP model. As we will see in the experiments, this algebraic representation of the alignment problem represents a significant reduction (of several orders of magnitude) in the time requirements for computing an alignment. Finally, we map text sentences to feature vectors with a rich unbounded set of features, which do not depend on an apriori assumption on the importance of certain constructions. This rich feature representation allows to differentiate semantic roles such as *actor* or *object*, and also allows to include other process information besides the task labels.

Table 1 shows the derived alignment for the example in Section 2, by both our tool and the one introduced in [9]. We want to stress that in spite of our better performance for this particular example, the quality of our approach and the one in [9] is similar. We believe both contributions can be naturally combined to boost the quality of the alignments derived.

Task	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Groundtruth	2	3	4	5	13	13	6	7	6	–	11	8	9	12	10	10
[9]’s approach	✓	✓	3 ✗	6 ✗	✓	✓	✓	12 ✗	✓	6 ✗	✓	6 ✗	✓	✓	✓	✓
Our approach	✓	✓	✓	✓	✓	✓	✓	✓	✓	6 ✗	✓	✓	✓	✓	✓	✓

Table 1. Errors in task-to-sentence alignments produced by our approach and by [9]’s for the example in Section 2.

4 Preliminaries on Process Models and NLP

4.1 Graphical Process Notations

There exist a plethora of graphical notations to model processes. A full description of them is beyond the scope of this paper. In this paper we focus on BPMN, a notation that has become one of the most widely used to model business processes. However, the techniques presented can be adapted to other notations like EPCs, Petri Nets, YAWL, among others.

BPMN models are composed by three types of nodes: events, activities and gateways. *Events* (represented as circles) denote something that happens (e.g., time, messages, ...), rather than *Activities* (rounded-corner rectangles) which are something that is done. Finally, *gateways* (diamond shapes) are used to describe the control flow. These elements can be partitioned into pools or lanes, to group activities performed by the same actor (person, department, institution, etc). An example of BPMN is shown in Figure 1.

4.2 Natural Language Processing

Natural Language Processing (NLP) is a wide research area inside Artificial Intelligence that includes any kind of technique or application related to the automatic processing of human language. NLP goals range from simple basic processing such as determining in which language a text is written, to high-level complex applications such as Machine Translation, Dialogue Systems, or Intelligent Assistants.

However, linguistic analysis tools can be used as a means to structure information contained in texts for its later processing in applications less related to language itself. This is our case, where we use NLP analyzers to convert a textual description of a BPM into a structured representation that can be compared, mapped, or analyzed using more conventional tools.

The NLP processing software used in this work is FreeLing¹ [5], 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.

¹ <http://nlp.cs.upc.edu/freeling>

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 [18] 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.

Semantic role labeling: 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.)

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.

5 Aligning Model and Text with ILP

5.1 Overview

A general description of the approach is shown in Figure 2. The overall process can be separated into three categories. The modules handling the text in natural language (white), the ones treating the process model (light gray) and finally, those working on feature vectors (dark gray).

As a first stage, the model task labels and the textual process description are analyzed using FreeLing to obtain a structured representation of the text. After that, a phase of feature extraction follows where the model tasks and the text sentences are both converted into a canonical feature vector representation. These vectors can then be compared by means of standard distance metrics.

Parallel to that, a chronological partial order of both the sentences in the text and the tasks in the model is computed. To find an optimal alignment between model and text, these ingredients are encoded as an ILP model, whose solution denotes an optimal alignment between tasks and sentences. That assignment is used afterwards to both present the results to the end user and compute a numerical similarity score.

5.2 Linguistic Analysis of Text and Model

We perform a full NLP analysis on the text body corresponding to the input text, as listed in Section 4.2. This is a distinctive aspect of our approach with respect to [8,9], since we gather the linguistic information from the semantic graph, which contains a structured semantic representation of the text.

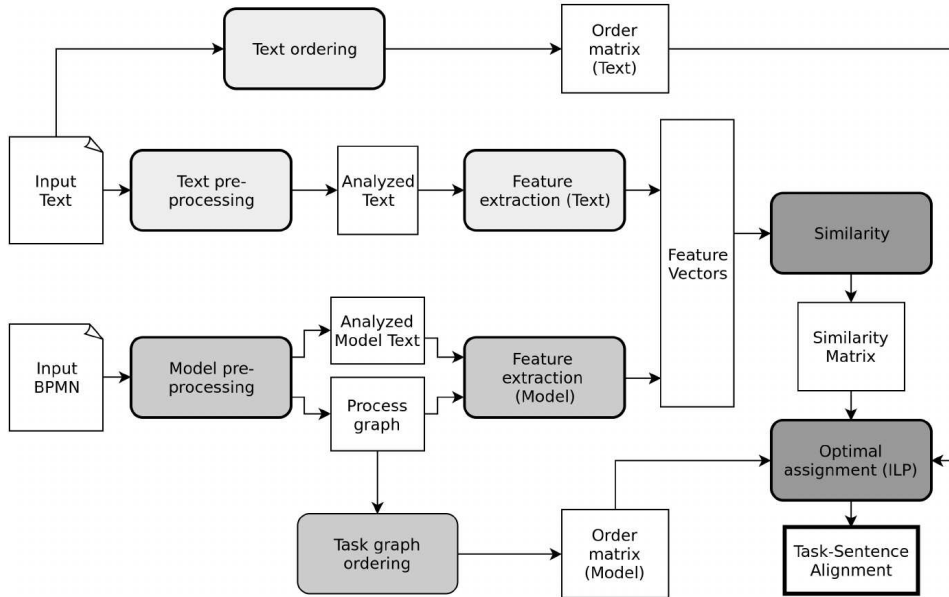


Fig. 2. Diagram illustrating the approach taken.

For the process model, we extract the natural language parts contained on it (from the activities, events, gateways, arcs, lanes and pools), and then a simple linguistic analysis (up to the word sense disambiguation step) is performed.²

All the information gathered is then used in the feature extraction phase, explained in the next section.

5.3 Feature Extraction

Up to this point, the model and text are represented using different structures and handled separately. At this step they are both converted into an identical representation of feature vectors by the means of a feature extraction step. The purpose of this transformation is to enable the comparison by deriving a canonical representation. The motivation of using feature vectors is to aim for an *open* description of the text in both representations, i.e., to consider features as assignments to linguistic characteristics extracted from the text. We have considered the following linguistic characteristics in our approach (unless stated otherwise, *target text* in descriptions below refers to either a sentence in the textual description or to the label of a model task):

contains_lemma(l, pos) This feature is extracted from the target text if it contains a word with the lemma l and part-of-speech pos .

² In order improve the performance of the word sense disambiguator on model sentences, the sentences from the text are provided as additional context to the analyzer. This greatly improves the disambiguation step when the model and the text are sharing a common semantic domain.

- contains_action(a)* This is extracted from a target text where the action a (typically a verb) is found.
- agent_contains(l, v)* This feature encodes who is performing the text or task action. It is extracted for text sentences whenever l is found as the agent of verb v , or for model tasks containing verb v and belonging to a swimlane/pool containing l .
- contains_synset(s)* This feature is extracted whenever the WordNet synset s appears in the target text.
- contains_hyponym(s)* This feature is extracted from a target text containing a word for which s is an hyponym³ at distance HL or less. HL is a parameter of the algorithm.
- object_contains(l, v)* This feature is extracted when l is found as the direct object of verb v in the target text.
- follows_conditional_containing(l)* This feature is extracted when l is found in a clause after a conditional statement (i.e. then, or else) in the text sentence, or when l is found in a task following an exclusive gateway with a question.

Table 2 shows some of the features extracted for sentence 3 in the example from Section 2, “*If he wants an individual membership, he must prepare his personal information*”. Note that the features include information such as lemma *customer* being mentioned, when it does not appear in the sentence. This is because the coreference resolution module detected that pronoun *he* in this sentence is referring to the actor *customer* mentioned somewhere else in the text. Also, output of the Semantic Role Labeler is also encoded in features stating that *customer* is the agent of action *want* and *prepare*, and that *information* is the object of *prepare*.

contains_lemma(customer, noun)	contains_lemma(want, verb)
contains_lemma(individual, adj.)	contains_action(want)
contains_action(prepare)	contains_synset(09984659-n::client)
contains_hyponym(10741590-n::user)	agent_contains(customer, wants)
agent_contains(customer, prepare)	object_contains(prepare, information)

Table 2. Set of features (some omitted for brevity) for example sentence 3.

Clearly, because the features are instantiated by words, this generates an open space of potentially infinite dimensions. For instance, the feature *contains_action* is instantiated twice for the sentence *The crane caught a fish and flew away*: *contains_action(catch)* and *contains_action(fly)*. In practice, this is handled by using a sparse representation of vectors.

The set of features proposed encodes high-level semantic information such as: who is the agent of the action, what is the action, or under what conditions is the task executed. That context is sometimes crucial in detecting whether a task

³ A word w_1 is a *hyponym* of w_2 iff w_1 describes a subclass of w_2 (e.g. *mammal* is a hyponym of *cat*, and *document* is a hyponym of *letter*). Hyponymy is obtained from WordNet.

is describing an action, referring to it or just using similar terms. The model and the text should generate similar feature vectors whenever the similarity between a sentence and a task is high, and vice-versa. This means the chosen features must represent properties that can be found both in the BPMN model and the textual description.

5.4 Similarity Metrics

After the feature vector transformation defined in 5.3 it suffices to compare similarities between feature vectors in order to compute the similarity between a task and a sentence. In order to adjust the relevance of a feature f , we associate a scalar weight w_f to it as the product of two values: a constant value w_f^c that is particular for each feature class (e.g: *contains_lemma*), and a variable value w_f^v whose magnitude depends on certain conditions. The set of constant weights of each feature class is a parameter of the algorithm. As an example, the particular weight of a feature such as *contains_lemma(customer, noun)* can be defined as the product of a constant factor for all instances of *contains_lemma* and the *tf-idf*⁴ score of this lemma in the sentence.

Three similarity metrics are available as parameters: The *Cosine similarity* and the weighted versions of the *Jaccard index* and the *Overlapping index*. We have evaluated all three metrics and have chosen the last one as the default for our tool, since it gives more intuitive numerical values and the performance between all three does not differ significantly:

Weighted overlapping index This metric expands the Overlapping index by considering weighted elements in the set, such as in our case:

$$WeightedOverlapping(A, B) = \frac{\sum_{f \in A \cap B} w_f}{\sum_{g \in smallest(A, B)} w_g}$$

This metric returns a bounded value between 0 and 1.

5.5 Text and Model Ordering

When only considering the similarity metrics defined in the previous section, a task and a sentence might be very similar but may appear at very different parts of the corresponding representations [8]. For example, the action described by the sentence might occur in the last part of the text, while the task could be amongst the first tasks to execute in the process model. This means the chronological order of the events must be taken into account when trying to determine whether a task and a sentence refer to the same action.

Consequently, we seek to find the partial order relation \preceq between the elements of both representations, text and model, such that $e \preceq e'$ means: “Element

⁴ The *tf-idf* of a token t is the product of $tf := (\text{Number of appearances of } t \text{ in its sentence} / \text{Number of tokens in that sentence})$ and $idf := \log_e(\text{Total number of sentences} / \text{Sentences containing } t)$

e happens before, or at the same time as e' ⁵. This allows us to define the strict order relation \rightsquigarrow as

$$e \rightsquigarrow e' \iff e \preceq e' \wedge e' \not\preceq e$$

In the process model, the computation of the strict ordering relation goes beyond the mere structure of the model, and instead should be computed from the underlying behavior. Fortunately, there are efficient techniques to determine the strict order relation [19, 20] of a process model. In this paper, the relation \rightsquigarrow corresponds to that same relation in the *behavioral profile* of the model as explained in [19].

Using the example in Section 2, a full *behavioral profile* would be extracted containing relations such as $G \rightsquigarrow I$ (happens before), $B + C$ (exclusive) or $H || L$ (parallel).

For the case of the text, it has been shown that the ambiguities present in textual descriptions make it impossible to determine the order of the tasks in them with total certainty [21]. This makes it hard to precisely extract the order unless techniques for extracting temporal relations are applied [22, 23]. In our case, we have chosen to simplify the problem assuming a sequential order of the events depicted in the text. This assumption fails whenever the text deliberately reports events in reverse order such as in: “Task A is performed. But before A, Task B must have been executed.”. In practice we hardly found such reverse ordering constructions in the texts describing process models.

5.6 Optimal Alignment Computation

This final step aims to find the optimal alignment between sentences in the textual description and tasks in the model. That information is then used in order to compute the global similarity between the model and the text as a numeric score. The information found in the optimal alignment can also aid in finding the actual inconsistencies between both representations as seen in [8]: i) Tasks describing actions not appearing in the text, ii) Sentences describing actions which are not in the model, and iii) Different orderings of tasks.

For a formal definition of the problem, let the task set be T , the sentence set be S and $sim(s, t)$ be the computed similarity between $s \in S$ and $t \in T$ ⁵ (cf Section 5.4). We assume for all pairs of elements both in S and T the order relation \rightsquigarrow has been computed.

We define an alignment as a partial function $f_A : T \rightarrow S$ of tasks to sentences such that $f_A(t) = s$, meaning that task t is describing the same actions as sentence s . In a fashion similar to that of [8], we define the optimal alignment f_A^* to be the alignment fulfilling the following properties:

Partial assignment The domain of f_A^* , denoted by $Dom(f_A^*)$ is a subset of the whole set of tasks, i.e. all $t \in T'$, for $T' \subseteq T$.

⁵ In this case, $sim(s, t)$ corresponds to $WeightedOverlapping(v_s, v_t)$ where v_s and v_t are the feature vectors of s and t respectively.

Order consistency Let $s = f_A^*(t)$ and $s' = f_A^*(t')$ for some pair of different tasks (t, t') . Then, the following restriction must hold: $t \rightsquigarrow t' \implies s \preceq s'$

Optimality The value of $\sum_{t \in \text{Dom}(f_A^*)} \text{sim}(t, f_A^*(t))$ is the maximum value such that the two other properties hold.

In order to obtain a solution, the aforementioned properties can be encoded in the following ILP:

$$\text{maximize: } \sum_{s \in S} \sum_{t \in T} a_{t,s} \cdot \text{sim}(t, s)$$

subject to:

$$\forall t \in T : \sum_{s \in S} a_{t,s} = 1$$

$$\forall (s, s') \in S \times S, (t, t') \in T \times T, t \rightsquigarrow t' \wedge s' \rightsquigarrow s : a_{t,s} + a_{t',s'} \leq 1$$

variables:

$$\forall s \in S, t \in T : a_{t,s} \in \{0, 1\}$$

The variables $a_{t,s}$ can be interpreted as: “Task t is assigned to sentence s ”, i.e: $a_{s,t} \iff s = f_A^*(t)$. The first family of constraints limits the number of sentences per task to exactly one⁶; this contradicts the requirement for function f_A^* to be partial, since a solution to the ILP model will have domain T . In practice, however, a threshold is used on the value $\text{sim}(t, s)$, and hence, assignments between tasks and sentences below this threshold are discarded. Finally, the second family of constraints encodes the *Order consistency* property by discarding the cases in which the order restriction would be violated.

Theorem 1. *The ILP model for aligning textual descriptions and process models is feasible and computes an optimal alignment f_A^* for a similarity metric sim .*

6 Experiments and Tool Support

The techniques of this paper have been implemented and are available as a web application⁷. The tool uses FreeLing for linguistic analysis, and Gurobi [24] as ILP solver. As similarity metric, we used the *weighted overlapping index*. Below we provide the two main experiments performed, devoted to analyze the positioning of the tool with respect to the state-of-the-art tool for the same task, (Section 6.1) and to test the tool capabilities in handling large instances (Section 6.2). The experiments for both tools have been performed on the same machine.

⁶ Note that these equations can also be encoded using the *Special Ordered Sets (SOS)* constraint $\forall t : a_{t,1}, \dots, a_{t,|S|}$, which denotes exactly the same constraint, and yields better performance in the ILP solvers that implement it.

⁷ The web application is available at: <http://xorrai.cs.upc.edu:8080/bpmninterface/>. The tool we present in this paper corresponds to the *BPMN vs Text* tab.

6.1 Comparison with the Technique from [9]

To validate the quality of the results provided by our tool, we compare them with the ones generated by the approach in [9], on a gold standard from [13] that was later extended by the authors of [9]. We also expanded the gold standard with the last group of models, taken from [13]. The models in this benchmark were manually analyzed in [13, 9] to obtain the correct assignment between tasks and sentences, so that the quality of a tool can be assessed.

Table 3 reports the results. For each model, we provide the number of tasks and sentences. Moreover, for each approach we report the accuracy (ratio of tasks correctly assigned to its matching sentence) and the execution time (average time per task) for each tool. To obtain a global perspective of the results, we provide a micro-average (total computation time over total number of tasks in all models), a macro-average (total computation time over number of models) and a median. The huge differences between both methods are caused by a small subset of models that are more difficult to solve than the rest. Although our approach produces slightly better accuracies than [9], the difference is not significant. However, our approach can obtain the same accuracy in the alignment with a remarkable reduction of computation time.

Model	T	S	[9] approach		Our proposal	
			Acc.	ms/task	Acc.	ms/task
Model1-2	8	6	100.0%	98	100.0%	29
Model1-4	7	11	100.0%	256	100.0%	45
Model10-1	4	3	75.0%	94	75.0%	27
Model10-10	10	8	70.0%	92	80.0%	26
Model10-11	9	7	77.8%	53	66.7%	23
Model10-12	5	4	80.0%	14	80.0%	23
Model10-13	4	3	100.0%	15	100.0%	25
Model10-14	10	5	50.0%	595	60.0%	29
Model10-3	12	11	91.7%	807	75.0%	28
Model10-4	11	9	90.9%	1,221	90.9%	28
Model10-5	4	4	100.0%	338	100.0%	24
Model10-6	4	3	75.0%	89	75.0%	20
Model10-7	8	7	100.0%	555	100.0%	19
Model10-8	5	7	80.0%	374	60.0%	27
Model10-9	8	5	100.0%	388	75.0%	23
Model2-1	26	38	76.9%	7,532	76.9%	134
Model2-2	19	30	63.2%	7,706	73.7%	84
Model3-1	6	7	100.0%	97	83.3%	28
Model3-2	6	4	100.0%	72	100.0%	20
Model3-3	4	5	100.0%	228	100.0%	29
Model3-4	2	4	50.0%	153	50.0%	56
Model3-5	11	9	81.8%	214	72.7%	31
Model3-6	6	8	83.3%	515	83.3%	28
Model4-1	18	40	33.3%	30,757	55.6%	173
Model5-1	2	6	0.0%	341	0.0%	73
Model5-2	5	5	60.0%	572	80.0%	30
Model5-3	9	10	55.6%	1,015	55.6%	34
Model6-2	4	5	75.0%	255	75.0%	33
Model6-3	5	9	80.0%	985	80.0%	1,880
Model6-4	9	14	66.7%	1,204	44.4%	47
Model7-1	4	7	100.0%	442	100.0%	30
Model8-1	5	3	100.0%	167	80.0%	18
Model8-2	5	6	40.0%	461	60.0%	26
Model8-3	5	5	100.0%	445	80.0%	26
Model9-1	7	8	71.4%	151	85.7%	33
Model9-3	6	4	100.0%	83	66.7%	25
Model9-4	7	5	28.6%	89	71.4%	26
Model9-5	8	7	62.5%	579	62.5%	24
Model9-6	8	13	37.5%	1,275	25.0%	45
BicycleManuf.	9	12	100.0%	772	66.7%	41
ClaimsCreation	6	5	83.3%	467	100.0%	30
HotelService	12	11	91.7%	196	83.3%	33
Dispatch-of-g.	7	7	71.4%	709	100.0%	35
Hospital	14	14	28.6%	25,109	71.4%	35
Hotel	12	11	83.3%	198	83.3%	30
Self-service	18	13	83.3%	1,002	88.9%	38
Underwriter	7	11	85.7%	274	100.0%	43
Zoo	15	12	46.7%	214	73.3%	31
Micro average			73.7%	3,517	76.3%	76
Macro average			75.6%	1,860	76.4%	79
Median			80.0%	357.5	80.0%	29.5

Table 3. Accuracy and solving time of our proposal and the one in [9].

6.2 Experiments on Large Instances

In the previous experiment validated the quality of the results provided by our technique. In this second experiment we focus on the time performance, using models of increasing size. This will allow to extrapolate the capabilities of our approach for larger instances. For the sake of comparison, we also include the execution times for the current implementation of the tool described in [9].

Due to the small number of available model–text pairs, and to reduced range of model sizes in existing data, we opted for generating a synthetic dataset of model-text pairs. The model generation consists of two steps: first we use the PGL2 tool [25] to generate the structure of a BPMN model. The second step consists of enriching the generated model by replacing model labels with randomly generated task descriptions. Once a process model is generated, a text is also generated with a random number of sentences $|S| = |T| \pm k$, where $|T|$ is the number of tasks in the model, and k was set to three in the experiments. Both the text sentences and the task descriptions in the model are generated with a simple word–bigram Markov model built using [26], trained with all the textual descriptions from the benchmark in Section 6.1. The generated synthetic benchmark has 400 model–text pairs ranging from 1 to 115 tasks.

Figure 3 shows the execution time of both tools for all model sizes⁸. The plots show that our approach has an asymptotic behavior with a complexity much lower than the methods in [9]. Remarkably, there is a correlation between the variance in the execution time and the input size: from size 50 upwards in the plot of the right of Figure 3, one can see that the execution time for models of similar size varies significantly. This suggests that other factors, apart from the model size, influence the execution time.

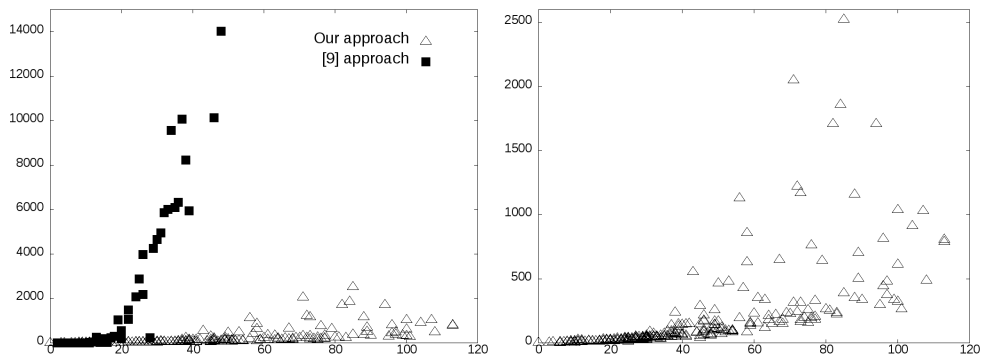


Fig. 3. Left: Execution times (in seconds) for [9] and our approach. Right: Zoom-in for the execution times of our approach.

⁸ We could not include all the executions for the approach from [9] since instances bigger than 46 tasks hit the imposed 4 hours time limit.

7 Conclusions and Future Work

In this paper we have proposed a novel approach for aligning textual descriptions and graphical models of processes. By applying a full linguistic analysis that results in an extensive set of features, and casting the problem as a mathematical optimization, we were able to align instances of unprecedented size. Moreover, in terms of quality the technique performs similar to the state of the art approach.

As a future work, we plan to expand the capabilities of the tool in different dimensions. First, we plan to incorporate the analysis of temporal relations in the text so that the control flow is better described. Second, a full exploration of the parameters of the technique (e.g., the weights for the similarity metric) will be done to boost the quality of the results. Finally, we plan to evaluate the tool in more realistic scenarios.

Acknowledgements. We would like to thank Han van der Aa and Henrik Leopold for their help and support to this work, and for sharing their software and part of the data used in the experiments of the paper. This work is funded by the Spanish Ministry for Economy and Competitiveness (MINECO), the European Union (FEDER funds) under grants COMMAS and Graph-Med (ref. TIN2013-46181-C2-1-R, TIN2016-77820-C3-3-R).

References

1. Leopold, H., Mendling, J., Polyvyanyy, A.: Supporting process model validation through natural language generation. *IEEE Trans. Software Eng.* **40**(8) (2014) 818–840
2. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: *Fundamentals of Business Process Management*. Springer (2013)
3. van der Aa, H., Leopold, H., Mannhardt, F., Reijers, H.A.: On the fragmentation of process information: Challenges, solutions, and outlook. In: *Enterprise, Business-Process and Information Systems Modeling - 16th International Conference, BPMDS 2015, 20th International Conference, EMMSAD 2015, Held at CAiSE 2015, Stockholm, Sweden (June 2015)* 3–18
4. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S.J., McClosky, D.: The Stanford CoreNLP natural language processing toolkit. In: *Association for Computational Linguistics (ACL) System Demonstrations*. (2014) 55–60
5. Padró, L., Stanilovsky, E.: Freeling 3.0: Towards wider multilinguality. In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC, Istanbul, Turkey (May 2012)* 2473–2479
6. Bird, S., Loper, E., Ewan, K.: *Natural Language Processing with Python*. O’Reilly Media Inc (2009)
7. Apache Software Foundation: *Apache OpenNLP*. opennlp.apache.org (2010)
8. van der Aa, H., Leopold, H., Reijers, H.A.: Detecting inconsistencies between process models and textual descriptions. In: *Business Process Management - 13th International Conference, BPM, Innsbruck, Austria (August 2015)* 90–105
9. van der Aa, H., Leopold, H., Reijers, H.A.: Comparing textual descriptions to process models – the automatic detection of inconsistencies. *Information Systems* (2016) In Press.

10. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. *MIS Quarterly* **28**(1) (2004) 75–105
11. Meziane, F., Athanasakis, N., Ananiadou, S.: Generating natural language specifications from UML class diagrams. *Requir. Eng.* **13**(1) (2008) 1–18
12. Bajwa, I.S., Choudhary, M.A.: From natural language software specifications to UML class models. In: *Enterprise Information Systems - 13th International Conference, ICEIS, Revised Selected Papers, Beijing, China (June 2011)* 224–237
13. Friedrich, F., Mendling, J., Puhmann, F.: Process model generation from natural language text. In: *Advanced Information Systems Engineering - 23rd Intl. Conference, CAiSE 2011, London, UK (June 2011)* 482–496
14. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. *VLDB J.* **10**(4) (2001) 334–350
15. Cayoglu, U., Dijkman, R., Dumas, M., Fettke, P., García-Bañuelos, L., Hake, P., Klinkmüller, C., Leopold, H., Ludwig, A., Loos, P., Mendling, J., Oberweis, A., Schoknecht, A., Sheerit, E., Thaler, T., Ullrich, M., Weber, I., Weidlich, M.: Report: The process model matching contest 2013. In: *Business Process Management Workshops - BPM International Workshops, Beijing, China (August 2013)* 442–463
16. de A. R. Gonçalves, J.C., Santoro, F.M., Baião, F.A.: Business process mining from group stories. In: *Proceedings of the 13th Intl. Conf. on Computers Supported Cooperative Work in Design, CSCWD, Santiago, Chile (April 2009)* 161–166
17. Sinha, A., Paradkar, A.M.: Use cases to process specifications in business process modeling notation. In: *IEEE International Conference on Web Services, ICWS, Miami, Florida (July 2010)* 473–480
18. Fellbaum, C.: *WordNet. An Electronic Lexical Database. Language, Speech, and Communication.* The MIT Press (1998)
19. Weidlich, M.: *Behavioural profiles: a relational approach to behaviour consistency.* PhD thesis, University of Potsdam (2011)
20. Polyvyanyy, A., Weidlich, M., Conforti, R., Rosa, M.L., ter Hofstede, A.H.M.: The 4c spectrum of fundamental behavioral relations for concurrent systems. In: *Application and Theory of Petri Nets and Concurrency - 35th International Conference, PETRI NETS 2014, Tunis, Tunisia, June 23–27, 2014. Proceedings.* (2014) 210–232
21. van der Aa, H., Leopold, H., Reijers, H.A.: Dealing with behavioral ambiguity in textual process descriptions. In: *Business Process Management - 14th International Conference, BPM, Rio de Janeiro, Brazil (September 2016)* 271–288
22. Mirza, P.: *Extracting Temporal and Causal Relations between Events.* PhD thesis, International Doctorate School in Information and Communication Technologies, University of Trento, Italy (2016)
23. UzZaman, N., Llorens, H., Derczynski, L., Allen, J., Verhagen, M., Pustejovsky, J.: Semeval-2013 task 1: Tempeval-3: Evaluating time expressions, events, and temporal relations. In: *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013), Atlanta, Georgia, USA, Association for Computational Linguistics (June 2013)* 1–9
24. Gurobi Optimization Inc: *Gurobi optimizer reference manual (2016)* Available at: <https://www.gurobi.com/documentation/6.5/refman.pdf>.
25. Burattin, A.: *PLG2: Multiperspective process randomization with online and offline simulations.* In: *Online Proceedings of the BPM Demo Track 2016, Rio de Janeiro, Brasil (September 2016)*
26. Schwartz, H.R.: *Markov sentence generator (2010)* Available at <https://github.com/hrs/markov-sentence-generator>.

Appendix IV

Formal Reasoning on Natural Language Descriptions of Processes

- **Publication Type:** Conference paper
- **Published In:** International Conference in Business Process Management (BPM 2019)
- **GGIS Rating:** A
- **Full Reference:** Josep Sànchez-Ferreres, Andrea Burattin, Josep Carmona, Marco Montali and Lluís Padró. “Formal Reasoning on Natural Language Descriptions of Processes”. In: *Proceeding of the 17th International Conference on Business Process Management (BPM). 2019.*

Formal Reasoning on Natural Language Descriptions of Processes

Josep Sànchez-Ferreres¹, Andrea Burattin², Josep Carmona¹, Marco Montali³,
and Lluís Padró¹

¹ Universitat Politècnica de Catalunya
{jsanchezf, jcarmona, padro}@cs.upc.edu

² Technical University of Denmark
andbur@dtu.dk

³ Free University of Bolzano
montali@inf.unibz.it

Abstract. The existence of unstructured information that describes processes represents a challenge in organizations, mainly because this data cannot be directly referred into process-aware ecosystems due to ambiguities. Still, this information is important, since it encompasses aspects of a process that are left out when formalizing it on a particular modelling notation. This paper picks up this challenge and faces the problem of ambiguities by acknowledging its existence and mitigating it. Specifically, we propose a framework to partially automate the elicitation of a formal representation of a textual process description, via text annotation techniques on top of natural language processing. The result is the ATDP language, whose syntax and semantics are described in this paper. ATDP allows to explicitly cope with several interpretations of the same textual description of a process model. Moreover, we link the ATDP language to a formal reasoning engine and show several use cases. A prototype tool enabling the complete methodology has been implemented, and several examples using the tool are provided.

1 Introduction

Organizing business processes in an efficient and effective manner is the overarching objective of *Business Process Management* (BPM). Classically, BPM has been mainly concerned with the quantitative analysis of key performance dimensions such as time, cost, quality, and flexibility [10] without considering in depth the analysis of textual data that talks about processes.

Hence, textual descriptions of processes in organizations are a vast and rather unexploited resource. Not neglecting the information that is present in natural language texts in a organization brings opportunities to complement or correct process information in conceptual models. In spite of this, only very recently *Natural Language Processing* (NLP)-based analysis has been proposed in the BPM context, as reported in [12, 15, 14, 22].

This paper is a first step towards the challenge of unleashing formal reasoning on top of textual descriptions of processes. By relying on *textual annotations*, we

propose ATDP, a multi-perspective language that can be connected to a reasoner so that a formal analysis is possible. From a raw textual description, annotations can be introduced manually, or selected from those inferred by NLP analysis (e.g., from libraries like [16]), thus alleviating considerably the annotation effort. Remarkably, our perspective differs from the usual trend in conceptual modelling, i.e., ATDP specifications can contain several interpretations, so ambiguity is not forced to be ruled out when modelling, for those cases when the process is under-specified, or when several interpretations are equally valid.

We formalize ATDP, and describe its semantics using linear temporal logic (LTL), with relations defined at two different levels, thanks to the notion of *scopes*. Then we show how to cast reasoning on such a specification as a model checking instance, and provide use cases for BPM, such as model consistency, compliance checking and conformance checking. Notably, such reasoning tasks can be carried out by adopting the standard infinite-trace semantics of LTL, or by considering instead finite traces only, in line with the semantics adopted in declarative process modeling notations like Declare [17]. Finally, a tool to convert ATDP specifications into a model checking instance is reported.

The paper is organized as follows: in the next section we provide the work related to the contributions of this paper. Then Section 3 contains the preliminaries needed for the understanding of the paper content. Section 4 describes a methodology to use ATDP in organizations. In Section 5 we provide intuition, syntax and semantics behind the ATDP language. Then in Section 6 it is shown how reasoning on ATDP specification can be done through model checking and finally Section 7 concludes the paper.

2 Related Work

In order to automatically reason over a natural language process description, it is necessary to construct a formal representation of the actual process. Such generation of a formal process model starting from a natural language description of a process has been investigated from several angles in the literature. We can project these techniques into a spectrum of support possibilities to automation: from fully manual to automatic.

The first available option consists in converting a textual description into a process model by manually modeling the process. This approach, widely discussed (e.g., [10, 9]), has been thoroughly studied also from a psychological point of view, in order to understand which are the challenges involved in such process of process modeling [18, 4]. These techniques, however, do not provide any automatic support and the possibility for automatic reasoning is completely depending on the result of the manual modeling. Therefore, ambiguities in the textual description are subjectively resolved.

On the opposite side of the spectrum, there are approaches that autonomously convert a textual description of a process model into a formal representation [13]. Such representation can be a final process model (e.g., as BPMN) [7] and, in this case, it might be possible to automatically extract in-

formation. The limit of these techniques, however, is that they need to resolve ambiguities in the textual description, resulting in “hard-coded” interpretations.

In the middle of the spectrum, we have approaches that automatically process the natural language text but they generate an intermediate artifact, useful to support the manual modeling by providing intermediate diagnostics [8, 20]. The problem of having a single interpretation for ambiguities is a bit mitigated in this case since a human modeler is still in charge of the actual modeling. However, it is important to note that the system is biasing the modeler towards a single interpretation.

The approach presented in this paper drops the assumption of resolving all ambiguities in natural language texts. Therefore, if the text is clear and no ambiguities are manifested, then the precise process can be modeled. However, if this is not the case, instead of selecting one possible ambiguity resolution, our solution copes with the presence of several interpretations for the same textual description.

3 A Recap on Linear Temporal Logics

In this paper, we use Linear Temporal Logic (LTL) [19] to define the semantics of the ATDP language. In particular, we use the standard interpretation of temporal logic formulae over *infinite traces*.

LTL formulae are built from a set \mathcal{P} of propositional symbols and are closed under the boolean connectives, the unary temporal operator \bigcirc (*next-time*) and the binary temporal operator U (*until*):

$$\varphi ::= a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 U \varphi_2 \quad \text{with } a \in \mathcal{P}$$

Intuitively, $\bigcirc\varphi$ says that φ holds at the *next* instant, $\varphi_1 U \varphi_2$ says that at some future instant φ_2 will hold and *until* that point φ_1 always holds. Common abbreviations used in LTL include the ones listed below:

- Standard boolean abbreviations, such as \top , \perp , \vee , \rightarrow .
- $\diamond\varphi = \top U \varphi$ says that φ will *eventually* hold at some future instant.
- $\square\varphi = \neg\diamond\neg\varphi$ says that from the current instant φ will *always* hold.
- $\varphi_1 W \varphi_2 = (\varphi_1 U \varphi_2 \vee \square\varphi_1)$ is interpreted as a *weak until*, and means that either φ_1 holds until φ_2 or forever.

Recall that the same syntax can also be used to construct formulae of LTL interpreted over *finite traces* [6]. Later on in the paper we show how our approach can also accommodate this interpretation. Recall however that the intended meaning of an LTL formula may radically change when moving from infinite to finite traces [5].

4 A Framework for Semantic Reasoning of Natural Language Descriptions of Processes

We briefly describe our envisioned framework for process modelling and management based on natural language. Figure 1 overviews the framework. Given a

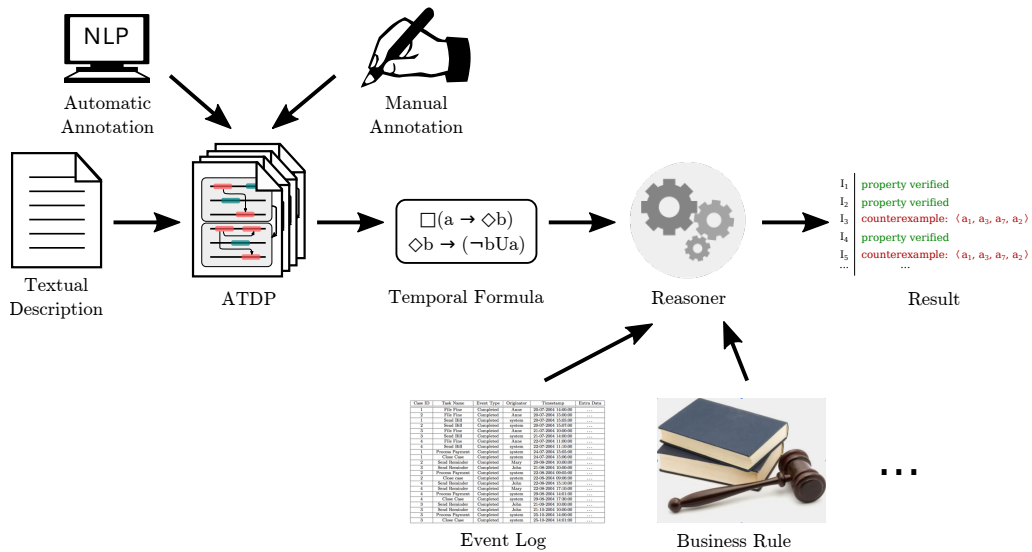


Fig. 1. Annotation framework overview

textual description of a process, automatic or manual annotation (or a combination of both) is used to obtain an Annotated Textual Description of a Process (ATDP), which contains all the interpretations of the original text. This specification can then be automatically transformed into temporal formula that encompasses the semantics of the process. The temporal formula can then be queried with the help of a reasoner (e.g., a model checker). Typical use cases may require the encoding of additional inputs, e.g., traces of an event log, compliance rules, among others. The result of the reasoner is the satisfaction or rebuttal (with the corresponding counterexample) of the query. Notice that query results may not hold in all possible interpretations of the text.

5 Processes as Annotated Textual Descriptions

We now propose ATDP, a language for annotated textual descriptions of processes starting with a gentle introduction relying on a real-world example. Specifically, we use the textual description of the examination process of a Hospital extracted from [21]. Figure 2 shows the full text, while Figure 3 contains a a fragment of the visualization for an ATDP specification of the description.

One of the key features of the ATDP approach is the ability to capture *ambiguity*. In our example, we can see this at the topmost level: the text is associated to three different interpretations I_1 , I_2 and I_3 , providing three different process-oriented semantic views on the text. Each interpretation is a completely unambiguous specification of the process, which fixes a specific way for understanding ambiguous/unclear parts. Such parts could be understood differently in another interpretation. A specification in ATDP then consists of the union of

all the valid interpretations of the process, which may partially overlap but also contradict each other.

Each interpretation consists of a hierarchy of *scopes*, providing a recursive mechanism to isolate parts of text that correspond to “phases” in the process. Each scope is thus a conceptual block inside the process, which is in turn decomposed as a set of lower-level scopes. Each scope dictates how its inner scopes are linked via control-flow relations expressing the allowed orderings of execution of such inner scopes. In our example, I_1 contains two scopes. A sequential relation indicates that the second scope is always executed when the first is completed, thus reconstructing the classical flow relation of conventional process modeling notation. All in all, the scope hierarchy resembles that of a process tree, following the variant used in [1].

Inside leaf scopes, *text fragments* are highlighted. There are different types of fragments, distinguished by color in our visual front-end. Some fragments (shown in red) describe the atomic units of behavior in the text, that is, activities and events, while others (shown in blue) provide additional perspectives beyond control flow. For example, `outpatient physician` is labelled as a *role* at the beginning of the text, while `informs` is labelled as an *activity*. Depending on their types, fragments can be linked by means of *fragment relations*. Among such relations, we find:

- Fragment relations that capture background knowledge induced from the text, such as for example the fact that the `outpatient physician` is the role responsible for performing (i.e., is the **Agent** of) the `informs` activity.
- Temporal constraints linking activities so as to declaratively capture the acceptable courses of execution in the resulting process, such as for example the fact that `informs` and `signs an informed consent` are in *succession* (i.e., `informs` is executed if and only if `signs an informed consent` is executed afterwards).

As for temporal relations, we consider a relevant subset of the well-known patterns supported by the Declare declarative process modeling language [17]. In this light, ATDP can be seen as a multi-perspective variant of a process tree where the control-flow of leaf scopes is specified using declarative constraints over the activities and events contained therein. Depending on the adopted constraints, this allows the modeler to cope with a variety of texts, ranging from loosely specified to more procedural ones. At one extreme, the modeler can choose to nest scopes in a fine-grained way, so that each leaf scope just contains a single activity fragment; with this approach, a pure process tree is obtained. At the other extreme, the modeler can choose to introduce a single scope containing all activity fragments of the text, and then add temporal constraints relating arbitrary activity fragments from all the text; with this approach, a pure declarative process model is obtained.

5.1 ATDP Models

ATDP models are defined starting from an input text, which is separated into *typed text fragments*. We now go step by step through the different components

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. 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. 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.

Fig. 2. Textual description of a patient examination process.

of our approach, finally combining them into a coherent model. We then move into the semantics of the model, focusing on its temporal/dynamic parts and formalizing them using LTL.

Fragment types. Fragments have no formal semantics associated by themselves. They are used as basic building blocks for defining ATDP models. We distinguish fragments through the following types.

Activity. This fragment type is used to represent the atomic units of work within the business process described by the text. Usually, these fragments are associated with verbs. An example activity fragment would be `validates` (from `validates the sample state`). Activity fragments may also be used to annotate other occurrences in the process that are relevant from the point of view of the control flow, but are exogenous to the organization responsible for the execution of the process. For instance, `(the sample) is contaminated` is also an activity fragment in our running example.

Role. The role fragment type is used to represent types of autonomous actors involved in the process, and consequently responsible for the execution of activities contained therein. An example is `outpatient physician`.

Business Object. This type is used to mark all the relevant elements of the process that do not take an active part in it, but that are used/manipulated by activities contained in the process. An example is the (medical) `sample` obtained and analyzed by physicians within the patient examination process.

When the distinction is not relevant, we may refer to fragments as the entities they represent (e.g. *activity* instead of *activity fragment*).

Given a set F of text fragments, we assume that the set is partitioned into three subsets that reflect the types defined above. We also use the following dot

notation to refer to such subsets: (i) F .activities for activities; (ii) F .roles for roles; (iii) F .objects for business objects.

Fragment relations. Text fragments can be related to each other by means of different non-temporal relations, used to express multi-perspective properties of the process emerging from the text. We consider the following relations over a set F of fragments.

Agent. An *agent relation* over F is a partial function

$$agent_F : F.activities \rightarrow F.roles$$

indicating the role responsible for the execution of an activity. For instance, in our running example we have $agent(\text{informs}) = \text{physician}$, witnessing that informing someone is under the responsibility of a physician.

Patient. A *patient relation* over F is a partial function

$$patient_F : F.activities \rightarrow F.roles \cup F.objects$$

indicating the role or business object constituting the main recipient of an activity. For instance, in our running example we have $patient(\text{prepare}) = \text{sample}$, witnessing that the prepare activity operates over a sample.

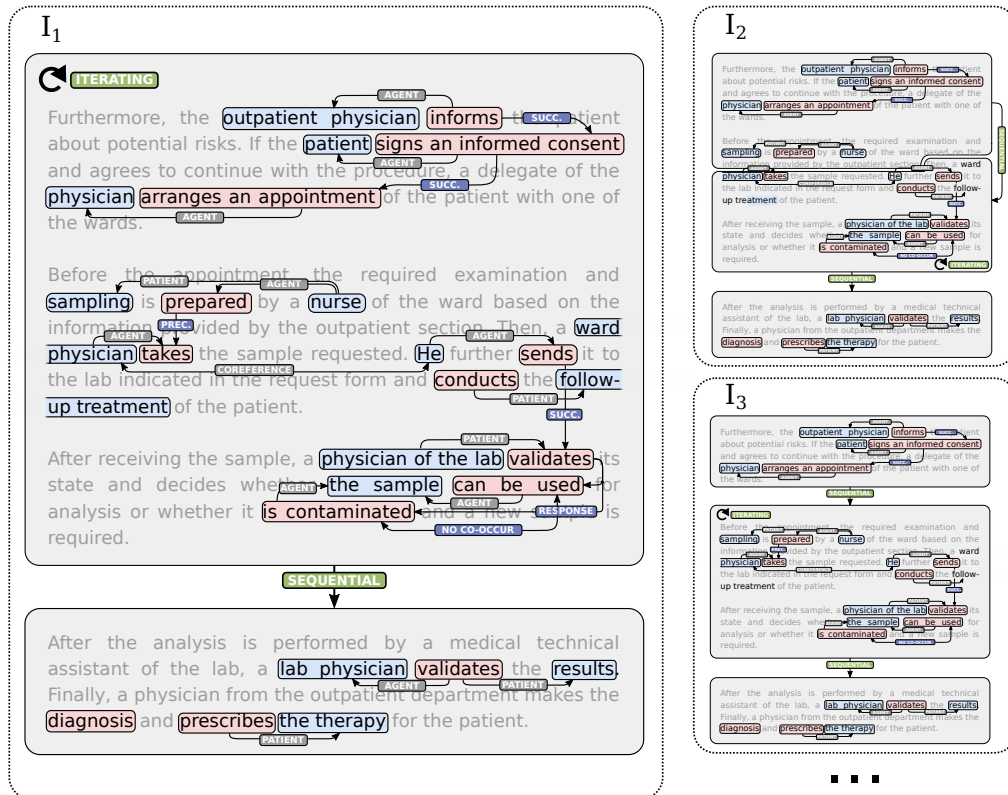


Fig. 3. Example annotation of a textual process description with multiple ambiguous interpretations. Some relations are omitted for brevity.

Coreference. A coreference relation over F is a (symmetric) relation

$$\text{coref}_F \subseteq F.\text{roles} \times F.\text{roles} \cup F.\text{objects} \times F.\text{objects}$$

that connects pairs of roles and pairs of business objects when they represent different ways to refer to the same entity. It consequently induces a coreference graph where each connected component denotes a distinct process entity. In our running example, all text fragments pointing to the `patient` role corefer to the same entity, whereas there are three different physicians involved in the text: the `outpatient physician`, the `ward physician` and the `physician of the lab`. These form disconnected coreference subgraphs.

Text scopes. To map the text into a process structure, we suitably adjust the notion of process tree used in [1]. In our approach, the blocks of the process tree are actually *text scopes*, where each scope is either a *leaf scope*, or a branching scope containing a one or an ordered pair⁴ of (leaf or branching) sub-scopes.

Each activity is associated to one and only one leaf scope, whereas each leaf scope contains one or more activities, so as to non-ambiguously link activities to their corresponding process phases.

Branching scopes, instead, are associated to a corresponding control-flow operator, which dictates how the sub-scopes are composed when executing the process. At execution time, each scope is enacted possibly multiple times, each time taking a certain amount of time (marked by a punctual scope start, and a later completion). We consider in particular the following scope relation types:

Sequential (\rightarrow) A sequential branching scope s with children $\langle s_1, s_2 \rangle$ indicates that each execution of s amounts to the sequential execution of its sub-scopes, in the order they appear in the tuple. Specifically: *(i)* when s is started then s_1 starts; *(ii)* whenever s_1 completes, s_2 starts; *(iii)* the completion of s_2 induces the completion of s .

Conflicting (\times) A conflicting branching scope s with children $\langle s_1, s_2 \rangle$ indicates that each execution of s amounts to the execution of one and only one of its children, thus capturing a choice. Specifically: *(i)* when s is started, then one among s_1 and s_2 starts; *(ii)* the completion of the selected sub-scope induces the completion of s .

Inclusive (\vee) An inclusive branching scope s with children $\langle s_1, s_2 \rangle$ indicates that each execution of s amounts to the execution of at least one of s_1 and s_2 , but possibly both.

Interleaving (\wedge) An interleaving branching scope s with children $\langle s_1, s_2 \rangle$ indicates that each execution of s amounts to the interleaved, parallel execution of its sub-scopes, without ordering constraints among them. Specifically: *(i)* when s is started, then s_1 and s_2 start; *(ii)* the latest, consequent completion of s_1 and s_2 induces the completion of s .

Iterating (\odot) An iterating branching scope s with child s_1 indicates that each execution of s amounts to the iterative execution of s_1 , with one or more

⁴ We keep a pair for simplicity of presentation, but all definitions carry over to n -ary tuples of sub-blocks.

iterations. Specifically: (i) when s is started, then s_1 starts; (ii) upon the consequent completion of s_1 , then there is a non-deterministic choice on whether s completes, or s_1 is started again.

All in all, a *scope tree* T_F over the set F of fragments is a binary tree whose leaf nodes S_l are called *leaf scopes* and whose intermediate/root nodes S_b are called *branch nodes*, and which comes with two functions:

- a total *scope assignment* function $parent : F.activities \rightarrow S_l$ mapping each activity in F to a corresponding leaf scope, such that each leaf scope in S_l has at least one activity associated to it;
- a total *branching type* function $btype : S_b \rightarrow \{\rightarrow, \times, \vee, \wedge, \cup\}$ mapping each branching scope in S_b to its control-flow operator.

Temporal constraints among activities. Activities belonging to the same leaf scope can be linked to each other by means of temporal relations, inspired by the Declare notation [17]. These can be used to declaratively specify constraints on the execution of different activities within the same leaf scope. Due to the interaction between scopes and such constraints, we follow here the approach in [11], where, differently from [17], constraints are in fact *scoped*.⁵

We consider in particular the following constraints:

Scoped Precedence Given activities a_1, \dots, a_n, b , $Precedence(\{a_1, \dots, a_n\}, b)$ indicates that b can be executed only if, within the same instance of its parent scope, at least one among a_1, \dots, a_n have been executed *before*.

Scoped Response Given activities a, b_1, \dots, b_n , $Response(a, \{b_1, \dots, b_n\})$ indicates that whenever a is executed within an instance of its parent scope, then at least one among b_1, \dots, b_n has to be executed *afterwards*, within the same scope instance.

Scoped Non-Co-Occurrence Given activities a, b , $NonCoOccurrence(a, b)$ indicates that whenever a is executed within an instance of its parent scope, then b *cannot* be executed within the same scope instance (and vice-versa).

Scoped Alternate Response Given activities a, b_1, \dots, b_n , $AlternateResponse(a, \{b_1, \dots, b_n\})$ indicates that whenever a is executed within an instance of its parent scope, then a *cannot be executed again* until, within the same scope, at least one among b_1, \dots, b_n is *eventually* executed.

Terminating Given activity a , $Terminating(a)$ indicates that the execution of a within an instance of its parent scope terminates that instance.

Mandatory Given activity a , $Mandatory(a)$ indicates that the execution of a must occur at least once for each execution of its scope.

Interpretations and models. We are now ready to combine the components defined before into an integrated notion of text interpretation. An *ATDP interpretation* I_X over text X is a tuple $\langle F, agent_F, patient_F, coref_F, T_F, C_F, \rangle$, where: (i) F is a set of *text fragments* over X ; (ii) $agent_F$ is an *agent function* over F ; (iii) $patient_F$ is a *patient function* over F ; (iv) $coref_F$ is a *coreference relation* over F ; (v) T_F is a *scope tree* over the activities in F ; (vi) C_F is a set of *temporal*

⁵ It is interesting to notice that Declare itself was defined by relying on the patterns originally introduced in [11].

constraints over the activities in F , such that if two activities are related by a constraint in, then they have to belong to the same leaf scope according to T_F .

An *ATDP model* M_X over text X is then simply a finite set of ATDP interpretations over X .

5.2 ATDP Semantics

We now describe the execution semantics of ATDP interpretations, in particular formalizing the three key notions of scopes, scope types (depending on their corresponding control-flow operators), and temporal constraints over activities. This is done by using LTL, consequently declaratively characterizing those execution traces that conform to what is prescribed by an ATDP interpretation. We consider execution traces as finite sequences of atomic activity executions over interleaving semantics.

Scope Semantics.

To define the notion of scope execution, for each scope s , we introduce a pair of artificial activities st_s and en_s which do not belong to F .activities. The execution of s starts with the execution of st_s , and ends with the execution of en_s . The next three axioms define the semantics of scopes:

A1. An activity a inside a scope s can only be executed between st_s and en_s :

$$\neg a W st_s \wedge \Box(en_s \rightarrow \neg a W st_s)$$

A2. A scope s can only be started and ended inside of its parent s' :

$$\neg(st_s \vee en_s) W st_{s'} \wedge \Box(en_{s'} \rightarrow \neg(st_s \vee en_s) W st_{s'})$$

A3. Executions of the same scope cannot overlap in time. That is, for each execution of a scope s 's start there is a unique corresponding end:

$$\begin{aligned} & \Diamond en_s \rightarrow (\neg en_s U st_s) \wedge \Box(st_s \rightarrow \Diamond en_s) \wedge \\ & \Box(st_s \rightarrow \bigcirc(\Diamond st_s \rightarrow (\neg st_s U en_s))) \wedge \\ & \Box(en_s \rightarrow \bigcirc(\Diamond en_s \rightarrow (\neg en_s U st_s))) \end{aligned}$$

Temporal Constraint Semantics. In this section, we define the semantics of temporal constraints between activities. Note that, in all definitions we will use

the subindex s to refer to the scope of the constraint.

$$\begin{aligned}
\text{Precedence}_s(\{a_1, \dots, a_K\}, b) &:= \bigvee_{i=1}^N \Box(st_s \rightarrow (\neg b U (a_i \vee en_s))) \\
\text{Response}_s(a, \{b_1, \dots, b_N\}) &:= \bigvee_{i=1}^N \Box(st_s \rightarrow (a \rightarrow (\neg en_s U b_i)) U en_s) \\
\text{NonCoOccurrence}_p(a, b) &:= \Box(st_s \rightarrow (a \rightarrow (\neg b U en_s)) U en_s) \wedge \\
&\quad \Box(st_s \rightarrow (b \rightarrow (\neg a U en_s)) U en_s) \\
\text{AlternateResponse}_p(a, b) &:= \text{Response}_p(a, b) \wedge \\
&\quad \Box(st_s \rightarrow (a \rightarrow \bigcirc(\neg a U (b \vee en_s)))) U en_s) \\
\text{Terminating}_p(a) &:= \Box(a \rightarrow \bigcirc en_s) \\
\text{Mandatory}_p(a) &:= \Box(st_s \rightarrow (\neg en_s U a))
\end{aligned}$$

Scope Relation Semantics. In all our definitions, let $\langle s_1, s_2 \rangle$ denote the children of a branching scope s , associated to the control-flow operator being defined. Note that by $\text{Sequence}(a, b)$ we refer to the formula $\text{Precedence}(\{a\}, b) \wedge \text{Response}(a, \{b\})$.

Sequential (\rightarrow) : $\text{Sequence}_s(en_{s_1}, st_{s_2}) \wedge \text{Mandatory}_s(st_{s_1}) \wedge \text{Mandatory}_s(st_{s_2})$

Conflicting (\times) : $\text{Mandatory}_s(st_{s_1}) \oplus \text{Mandatory}_s(st_{s_2})$

Inclusive (\vee) : $\text{Mandatory}_s(st_{s_1}) \vee \text{Mandatory}_s(st_{s_2})$

Interleaving (\wedge) : $\text{Mandatory}_s(st_{s_1}) \wedge \text{Mandatory}_s(st_{s_2})$

Iterating (\odot) : This relation is defined by negation, with any non-iterating scope s , child of s' , fulfilling the property:

$$(st_{s'} \rightarrow (\neg en_{s'} U st_s \wedge (st_s \rightarrow \bigcirc(\neg st_s U en_{s'}))) U en_{s'})$$

Additionally, iterating scopes may be affected by the presence of terminating activities, as defined by the following property: A terminating activity a_t inside an iterating scope s , child of s' , stops the iteration. That is, its execution cannot be repeated anymore inside its parent:

A4. $\Box(st_{s'} \rightarrow ((a_t \rightarrow (\neg st_s U en_{s'})) U en_{s'}))$

6 Reasoning on ATDP Specifications

A specification in ATDP is the starting point for reasoning over the described process. This section shows how to encode the reasoning as a model checking instance, so that a formal analysis can be applied on the set of interpretations of the model. Furthermore, we present three use cases in the scope of business process management: *checking model consistency*, *compliance checking* and *conformance checking*.

6.1 Casting Reasoning as Model Checking

Reasoning on ATDP specifications can be encoded as an instance of model checking, which allows performing arbitrary queries on the model. The overall system can be defined by the following formula

$$(A \wedge \mathcal{C}_F \wedge \mathcal{C}_{T_F}) \implies Q \quad (1)$$

where A is the conjunction of all LTL formulas defined by the axioms, \mathcal{C}_F is the conjunction of the activity temporal constraints, \mathcal{C}_{T_F} is the conjunction of all LTL formulas defined by the semantics of the process tree, and Q is an arbitrary query expressed in LTL (cf. Section 5.2).

In this paper, we present an encoding of the ATDP’s semantics into NuSMV, a well-known software for model-checking [3]. First, the notion of process execution is defined using an *activity* variable, with a domain of all the activities in the ATDP. At any given step, the system may choose a single value for this variable, meaning that this activity has been executed. This ensures that simultaneous execution of activities will not happen.

The system definition for an ATDP is then split into two parts: a transition system and an LTL property specification. The transition system is a graph defining the next possible values for the *activity* variable given its current value. In our proposed encoding the transition system is specified as a complete graph, since all the behavioural constraints are specified in A , \mathcal{C}_F and \mathcal{C}_{T_F} as parts of the property specification, as seen in Eq. (1). This property specification is directly encoded as a single LTL formula.

We can adapt NuSMV, which performs model checking on infinite traces, to check properties on finite traces when necessary. In order to do that, we add a special activity value **STOP**. In the transition system, an edge is added from any possible activity to **STOP**. Additionally, the constraint $\Diamond \text{STOP}$ is added to the antecedent of the LTL property specification. This enforces that all traces accepted by the model end in an infinite loop repeating the (only) execution of the **STOP** activity, which is equivalent to terminating execution.

Non-temporal information can be introduced in the queries without increasing the problem complexity, since the information is statically defined. For example, when the text mentions that several activities are performed by a certain role, this information remains invariant during the whole model-checking phase. Thus, queries concerning roles can be translated directly into queries about the

set of activities performed by that role. A possible encoding of this into a model checker consists of adding additional variables during the system definition.

When dealing with multiple interpretations, the above framework is extended with two types of queries:

Existential: Is the proposition true in any interpretation of the process?

$$\exists I \in \text{ATDP} : (A_I \wedge \mathcal{C}_{F_I} \wedge \mathcal{C}_{T_{F_I}}) \implies Q$$

Complete: Is the proposition true in all interpretations of the process?

$$\forall I \in \text{ATDP} : (A_I \wedge \mathcal{C}_{F_I} \wedge \mathcal{C}_{T_{F_I}}) \implies Q$$

Existential and complete queries can be used to reason in uncertain or incomplete specifications of processes.

An application of complete queries would be finding invariant properties of the process. That is, a property that holds in all possible process interpretations. Existential queries, in turn, fulfill a similar role when the proposition being checked is an undesired property of the process. By proving invariant properties, it is possible to extract information from processes even if these are not completely specified or in case of contradictions. A negative result for this type of query would also contain the non-compliant interpretations of the process, which can help the process owner in gaining some insights about which are the assumptions needed to comply with some business rule.

Tool Support.

The encoding technique described in Section 6.1 has been implemented in a prototype tool, ATDP2NuSMV. The tool can be used to convert an ATDP specification into a NuSMVinstance.

ATDP2NuSMV is distributed as a standalone tool, that can be used in any system with a modern Java installation, and without further dependencies. A compiled version as well as the source code can be found in the following repository: <https://github.com/setzer22/atdp2nusmv>.

In the next subsection, we present use cases that have been tested with ATDP2NuSMV and NuSMV. The ATDP specifications as well as the exact query encodings can be found in the repository. The use case examples are based on a full version of the specification presented in Figure 3.

Use Case 1: Model Consistency. An ATDP specification can be checked for consistency using proof by contradiction. Specifically, if we set $Q = \perp$, the reasoner will try to prove that $A \wedge \mathcal{C}_F \wedge \mathcal{C}_{T_F} \rightarrow \perp$, that is, whether a false conclusion can be derived from the axioms and constraints describing our model. Since this implication only holds in the case $\perp \rightarrow \perp$, if the proof succeeds we will have proven that $A \wedge \mathcal{C}_F \wedge \mathcal{C}_{T_F} \equiv \perp$, i.e. that our model is not consistent. On the contrary, if the proof fails we can be sure that our model does not contain any contradiction.

To illustrate this use case, we use interpretations `hosp-1` and `hosp-1-bad`, available in our repository. The first interpretation consists of a complete version of the specification in Figure 3, where `F.activities` includes $a_1 = \text{takes (the sample)}$ and $a_2 = \text{validates (sample state)}$, and constraints in \mathcal{C}_F include: `Mandatory(a_1)`, `Precedence($\{a_1\}, a_2$)` and `Response($a_1, \{a_2\}$)`.

NuSMV falsifies the query in interpretation `hosp-1` with a counter-example. When the model is consistent, the property is false, and the resulting counter example can be any valid trace in the model.

The second specification, `hosp-1-bad` adds `Precedence({a2}, a1)` to the set of relations R . This relation contradicts the previously existing `Precedence({a1}, a2)`, thus resulting in an inconsistent model. Consequently, NuSMV cannot find a counter-example for the query in interpretation `hosp-1-bad`. This result can be interpreted as the model being impossible to fulfill by any possible trace, and thus inconsistent.

Use Case 2: Compliance Checking.

Business rules, as those arising from regulations or SLAs, impose further restrictions that any process model may need to satisfy. On this regard, compliance checking methods assess the adherence of a process specification to a particular set of predefined rules.

The presented reasoning framework can be used to perform compliance checking on ATDP specifications. An example rule for our running example might be: “An invalid sample can never be used for diagnosis”. The relevant activities for this property are annotated in the text: $a_3 = (\text{the sample}) \text{ can be used}$, $a_4 = (\text{the sample}) \text{ is contaminated}$, $a_5 = \text{makes the diagnosis}$, and the property can be written in LTL as: $Q = \Box(a_4 \rightarrow (\neg a_5 U a_3))$.

In the examples from our repository, interpretations `hosp-2-i`, with $i = \{1, 2, 3\}$, correspond to the three interpretations of the process shown in Figure 3. Particularly, the ambiguity between the three interpretations is the scope of the repetition when the taken sample is contaminated. The three returning points correspond to: `sign an informed consent`, `sampling is prepared` and `take the sample`. NuSMV finds the property true for all three interpretations, meaning that we can prove the property $\Box(a_4 \rightarrow (\neg a_5 U a_3))$ without resolving the main ambiguity in the text.

Use Case 3: Conformance Checking. Conformance checking techniques put process specifications next to event data, to detect and visualize deviations between modeled and observed behavior [2]. On its core, conformance checking relies on the ability to find out whereas an observed trace can be reproduced by a process model.

A decisional version of conformance checking can be performed, by encoding traces inside Q as an LTL formulation. Given a trace $t = \langle a_1, a_2, \dots, a_N \rangle$, we can test conformance against an ATDP interpretation with the following query⁶:

$$Q = \neg(a_1 \wedge \bigcirc(a_2 \wedge \bigcirc(\dots \wedge \bigcirc(a_N \wedge \bigcirc \text{STOP}))))$$

This query encodes the proposition “Trace t is not possible in this model”. This proposition will be false whenever the trace is accepted by the model.

⁶ The proposed query does not account for the start and end activities of scopes, which are not present in the original trace. A slightly more complex version can be crafted that accounts for any invisible activity to be present between the visible activities of the trace. We do not show it here for the sake of simplicity.

Other variants of this formulation allow for testing trace patterns: partial traces or projections of a trace to a set of activities. In this case, the counter-example produced will be a complete trace which fits the model and the queried pattern.

As an example of this use-case, we provide the example ATDP interpretation `hosp-3` in our repository. We project the set of relevant activities to the set $a_6 = \text{informs (the patient)}$, $a_7 = \text{signs (informed consent)}$ $a_8 = \text{arranges (an appointment)}$. Two trace patterns are tested, the first: $t_1 = \langle \dots a_6, a_7, a_8, \dots \rangle$ and $t_2 = \langle \dots a_7, a_6, a_8, \dots \rangle$. NuSMV finds the trace pattern t_1 fitting the model, and produces a full execution trace containing it. On the other hand, t_2 does not fit the model, which is successfully proven by NuSMV.

7 Conclusions and Future Work

This paper proposes ATDP, a novel multi-perspective language for the representation of processes based on textual annotation. On the control-flow dimension, ATDP is a mixture of imperative constructs at general level via scopes, and declarative constructs inside each scope. In a way, the language generalizes process trees, allowing declarative relations instead of atomic activities in the leaf nodes. The paper also shows how to translate ATDP specifications into temporal formulas that are amenable for reasoning. Three use cases in the context of BPM are shown, illustrating the potential of the ideas in this paper.

Several avenues for future work are under consideration. First, to explore alternatives or refinements of the encoding in Eq. (1) to make it more suitable in a model-checking context. Second, to validate the proposed language against more examples and use cases, specifically by testing how the ATDP primitives accommodate to different document styles. Finally, studying the connection between ATDP and other process model notations may serve as a bridge between textual descriptions and their operationalization within an organization.

Acknowledgments This work has been supported by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R and by Innovation Fund Denmark project EcoKnow.org (7050-00034A).

References

1. Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. A genetic algorithm for discovering process trees. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2012.
2. Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich. *Conformance Checking - Relating Processes and Models*. Springer, 2018.
3. Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A new symbolic model checker. *STTT*, 2(4):410–425, 2000.
4. Jan Claes, Irene Vanderfeesten, J. Pinggera, H.A. Reijers, B. Weber, and G. Poels. A visual analysis of the process of process modeling. *Information Systems and e-Business Management*, 13(1):147–190, 2015.
5. Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 1027–1033. AAAI Press, 2014.

6. Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 854–860. IJCAI/AAAI, 2013.
7. Luis Delicado, Josep Sanchez-Ferrerres, Josep Carmona, and Lluís Padró. NLP4BPM - Natural Language Processing Tools for Business Process Management. In *BPM Demo Track*, 2017.
8. Luis Delicado, Josep Sanchez-Ferrerres, Josep Carmona, and Lluís Pardo. The Model Judge - A Tool for Supporting Novices in Learning Process Modeling. In *BPM 2018 Demonstration Track*, 2018.
9. Remco Dijkman, Irene Vanderfeesten, and Hajo A. Reijers. Business process architectures: overview, comparison and framework. *Enterprise Information Systems*, 10(2):129–158, feb 2016.
10. Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.
11. Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of the 1999 International Conference on Software Engineering (ICSE)*, pages 411–420. ACM, 1999.
12. Henrik Leopold. *Natural language in business process models*. PhD thesis, Springer, 2013.
13. Bilal Maqbool, Farooque Azam, Muhammad Waseem Anwar, Wasi Haider Butt, Jahan Zeb, Iqra Zafar, Aiman Khan Nazir, and Zuneera Umair. A Comprehensive Investigation of BPMN Models Generation from Textual Requirements - Techniques, Tools and Trends. In *Information Science and Applications 2018 - ICISA 2018, Hong Kong, China, June 25-27th, 2018*, volume 514 of *Lecture Notes in Electrical Engineering*, pages 543–557. Springer, 2019.
14. Jan Mendling, Bart Baesens, Abraham Bernstein, and Michael Fellmann. Challenges of smart business process management: An introduction to the special issue. *Decision Support Systems*, 100:1–5, 2017.
15. Jan Mendling, Henrik Leopold, and Fabian Pittke. 25 challenges of semantic process modeling. *International Journal of Information Systems and Software Engineering for Big Companies*, 1(1):78–94, 2015.
16. Lluís Padró and Evgeny Stanilovsky. Freeling 3.0: Towards wider multilinguality. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC)*, pages 2473–2479, 2012.
17. M. Pesic, H. Schonenberg, and W. M. P. van der Aalst. DECLARE: Full support for loosely-structured processes. In *Proc. of the Eleventh IEEE Int. Enterprise Distributed Object Computing Conference (EDOC'07)*, pages 287–298. IEEE Computer Society, 2007.
18. J. Pinggera. *The Process of Process Modeling*. PhD thesis, University of Innsbruck, Department of Computer Science, 2014.
19. Amir Pnueli. The temporal logic of programs. pages 46–57. IEEE, 1977.
20. Josep Sànchez-Ferrerres, Josep Carmona, and Lluís Padró. Aligning textual and graphical descriptions of processes through ILP techniques. In *Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CAiSE)*, pages 413–427, 2017.
21. F Semmelrodt. Modellierung klinischer prozesse und compliance regeln mittels BPMN 2.0 und eCRG. Master’s thesis, University of Ulm, 2013.
22. 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 *Proceedings of the 27th International Conference on Computational Linguistics (COLING)*, pages 2791–2801, 2018.

Appendix V

From Process Models to Chatbots

- **Publication Type:** Conference paper
- **Published In:** International Conference on Advanced Information Systems Engineering (CAiSE 2019)
- **GGIS Rating:** A
- **Full Reference:** Anselmo López, Josep Sànchez-Ferreres, Josep Carmona and Lluís Padró. “From Process Models to Chatbots”. In: *Proceedings of the 31th International Conference on Advanced Information Systems Engineering (CAiSE)*. 2019. pp. 383-398. (GGIS Rating: A)

From Process Models to Chatbots

Anselmo López, Josep Sànchez-Ferrerres, Josep Carmona, and Lluís Padró

Process and Data Science Group, Computer Science Department.
Universitat Politècnica de Catalunya. Barcelona, Spain.
{anselmol, jsanchezf, padro, jcarmona}@cs.upc.edu
<http://www.cs.upc.edu/~pads-upc>

Abstract. The effect of digital transformation in organizations needs to go beyond automation, so that human capabilities are also augmented. A possibility in this direction is to make formal representations of processes more accessible for the actors involved. On this line, this paper presents a methodology to transform a formal process description into a conversational agent, which can guide a process actor through the required steps in a user-friendly conversation. The presented system relies on dialog systems and natural language processing and generation techniques, to automatically build a chatbot from a process model. A prototype tool –accessible online– has been developed to transform a process model in BPMN into a chatbot, defined in Artificial Intelligence Marking Language (AIML), which has been evaluated over academic and industrial professionals, showing potential into improving the gap between process understanding and execution.

1 Introduction

Formal process modeling notations are ubiquitous in organizations. They precisely describe a business process, using a graphical notation that has a formal execution semantics, amenable for automating certain tasks of the underlying process [3]. These notations, among which Business Process Model and Notation (BPMN) is a prominent example, are not always suitable or understandable by any actor involved in the process. A good example is a logistic processes, where several agents are required, ranging from agents to transport the goods, down to accountants that keep track of the finances of the whole process.

Hence, one cannot assume always that all the actors of a process would be able to understand a BPMN model, in order to know what they need to do for the successful execution of the process. The fact that digital transformation aims at a better maturity and elicitation of an organization’ processes [7], would only contribute to increasing the complexity and size of the process repositories in organizations, which in turn causes a pressure on process’ actors. The main goal of the work of this paper is to alleviate this pressure. A similar observation and motivation was presented in the seminal work to convert a BPMN model into a textual description [6], from which this paper shares some parts of the methodology proposed.

In this paper we are inspired by a trend seen in the last few years in online services. Often, these sites have a section called *Frequently Asked Questions*, where users can read some solutions to common problems. Sometimes, these pages also have guides to execute some complicated processes or tasks. The main problem is that users have to search for their solution through all the web content, which is often a tedious task. That is why companies are using alternatives to help their customers [15]. One of the most implemented options currently is the *conversational bot* or *chatbot*¹.

Chatbots allow a user to query a complex content, so that a more human interaction with it is enabled. Moreover, the user is relieved from the burden of searching for a solution, which is now a task carried out by the chatbot.

In this paper we present a methodology that takes as input a BPMN model, and generates a chatbot aimed to guide a process actor through the modeled business process. The actor can be guided step-by-step through the process, ask questions about who should perform certain task, or to whom should a document be sent, etc. In this way, a more flexible process interaction is envisioned at a very low cost, since using the methodology proposed some of the processes of a process repository can be transformed into chatbots. The methodology has been validated over 33 individuals, both from academy and industry.

The methodology proposed relies on *script-based dialog management* [16], in which the dialog state determines what is the system expecting at a given moment, and the user utterance will determine the system's answer and the transition to a new dialog state. We generate the finite state dialog automaton from the BPMN structure, and the system utterances from the textual components of the model (task labels, pool and swimlane names, ...), and we add additional states and transitions to deal with user questions about actors (e.g. who should do a task) and objects (e.g. to whom a document must be sent).

The organization of this paper is as follows: next section provides a simple example to illustrate the contributions of this paper. Then, in Section 3 we provide the necessary background to understand the methodology that will be presented in Section 4. In Section 5 we describe the prototype tool implementing the methodologies of this paper, which is then validated in Section 6. Finally, Section 7 summarizes the paper milestones and reports future challenges ahead.

2 Motivating Example

To illustrate the contribution of this paper, Figure 2 shows an interaction with the chatbot obtained by applying the methodology proposed on the simple process model shown in Figure 1². The interaction is shown in the following page.

By a careful look at the interaction, one can see the main ingredients of the methodology described in this paper. First, Natural Language Processing (NLP)

¹ One can see an example for the case of computer repair at <http://www.toshiba.co.uk/generic/yoko-home/>.

² This example was found in the Signavio Academic Initiative pool of BPMN models.

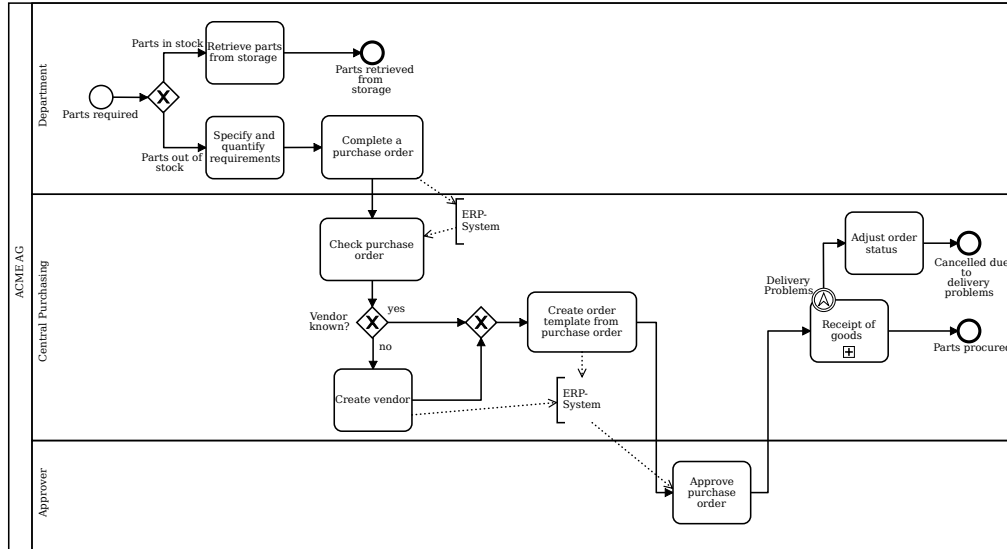


Fig. 1. BPMN representation of the ‘Procure parts’ business process

is required to analyze the text in the different elements of the BPMN model. Second, a tailored finite state automata representation of the underlying process model is used, so that the conversation state is unambiguous depending on the previous questions. Finally, natural language generation is used, so that sentences are created to describe the task to the user in a human-readable manner: see for instance the third-person form when explaining the tasks performed.

3 Preliminaries

3.1 Process Modeling

As it has been already acknowledged in the introduction, formal process notations are an important part of any digitalization initiative, since they open the door to an unambiguous and focused (process) automation. A recent article reports three different process management levels, ranging from multi-process management, to the classical process management level, down to process instance level [8].

Process models can be created using a variety of modeling languages, such as Petri nets, Event-Driven Process Chains (EPCs), and BPMN. Although we focus in BPMN, the contributions of this paper are independent of the specific notation used to define a process model. In particular, we focus on BPMN 2.0, notation created as a standard for business process modeling. BPMN has three different kinds of elements. First, the main elements are the nodes in the diagram, which may belong to three different types: Events, which represent that something happens; Activities, which represents some task that is performed;

Bot: Hi, which process do you want to see?
You: I'm interested in the procure parts process.
Bot: The process begins when the parts are required by the Department.
 What do you want to do now?
You: Who checks the purchase order?
Bot: The Central Purchasing checks the purchase order.
You: Follow process.
Bot: Parts out of stock/Parts in stock ?
You: Parts out of stock.
Bot: The Department specifies and quantifies the requirements.
You: Next.
Bot: The Department completes the purchase order.
You: Next.
Bot: The Central Purchasing checks the purchase order. Vendor known?
You: Yes.
Bot: The Central Purchasing creates the order template from purchase order.
You: Who did you say before that checks the purchase order?
Bot: The Central Purchasing checks the purchase order.
You: ok.
Bot: The Approver approves the purchase order.
You: Next.
Bot: After that, the Central Purchasing executes process called "receipt of goods".
You: Next.
Bot: Then, the process is finished.

Fig. 2. Example of dialog with the bot generated for the "procure parts" process.

and Gateways, which split or join flow control. Second, the notation has different edges to connect nodes. A solid line indicates the process workflow, while dashed lines represent messages sent between process participants³. Finally, there are organization elements such as lanes that contain activities performed by the same participant, and pools, that group several related lanes.

3.2 Dialog Systems

Dialog is the most natural way for humans to communicate, and since the dawn of computers, researchers have aimed to devise ways to communicate with machines as we do with people. From Eliza [19] –the first reactive chatbot– to modern assistants like Siri, Alexa, or Google Assistant, dialog systems construction still strongly rely on large amounts of human intervention, to establish which topics the chatbot should be aware of, and provide useful answers to.

Regardless of whether the dialog interface is oral or written, traditional dialog systems are tailored to a specific task (e.g. helping the user to buy a plane ticket, post a claim for a wrong product, etc.) since the system requires a precise definition of domain concepts and actions to execute depending on the user

³ Dashed lines can also represent data associations.

input. For this reason, they usually are expensive to develop, and not easily customizable to new application domains. This is also the case of modern personal assistants.

On the other hand, there are the so-called *recreational* (also known as *conversational*) chatbots which do not target a specific task, but only aim to entertain the user, or to win a Turing's Test competition [19, 18].

Dialog systems typically consist of four main components:

- User input processing and understanding: Takes care of processing the user input (which may be speech- or text-based, or even multimodal) and extracting the relevant information and intention.
- Dialog manager: Keeps track of the dialog state, and decides how to update it, and which tasks should be executed at each moment.
- Task Manager: Deals with the back-office operations required for the dialog goal (retrieving information from a database, purchasing tickets, booking reservations, etc).
- Output generator: Produces the appropriate answer or feedback (speech, text, or multimodal) to be sent to user.

Each of this components may be realized at different levels of complexity: Input processing may range from a simple keyword matching on the user text to an advanced Natural Language Processing system. Dialog managers can follow a simple stateless reactive pattern, be based on finite state automata or more complex state-keeping structures, or rely on advanced Machine Learning methods (which require lots of annotated data –actual dialogs– relative to the target domain to be trained). Task Manager –which is missing in non-task oriented dialog systems– is the most domain-dependent component, and must be tailored for each application. And finally, output generation can be approached with techniques ranging from basic pre-written fixed sentences or patterns, up to complete Natural Language Generation systems.

See [5, 1] for more details on dialog systems architectures and technologies.

3.3 Natural Language Processing and Generation

Apart from the internal logic or domain-related reasoning that a dialog system must carry out (e.g. access a database to extract available flights matching user's needs, decide which may be most useful, etc.), a crucial part of the dialog is understanding user utterances.

For that, NLP tools are required in order to convert the text spoken or written by the user into structured data that can be used by the system.

In our case, we are generating a chatbot from a BPMN model. For that, we need to extract information from the language components in the model – basically the task labels and pool and lanes descriptions– and for this we also resort to NLP tools to extract the actions being described in the labels, the agents who perform each action, and the objects upon which action is performed. The way we extract this information follows a similar strategy to the one presented in [6].

Another important component in any dialog system is that in charge of generating the system reply that will be sent to the user. Ideally, the system utterance should sound natural, avoid reiteration of already shared information, use a varied set of language structures and lexica, etc. This is addressed by a subfield of NLP known as Natural Language Generation (NLG), that given a semantic representation of the concepts to be expressed, generates the appropriate sentences. NLG is used not only to generate system replies in dialog systems, but also in automatic document generation, either to generate reports from raw data (*data-to-text* NLG) or from other texts (*text-to-text* NLG) (e.g. automatic summarization).

NLG can be approached at different complexity levels, depending on the task and on the expected results. Simple dialog systems often use pre-canned sentences (which may contain wildcards that are appropriately replaced). A varied set of pre-canned sentences for each situation, from which an answer is randomly chosen when needed, may be enough to avoid a too repetitive user experience.

However, for more advanced NLG applications, complex architectures may be needed. Main steps in a NLG system involve: Determining what to say, planning the structure of the generated text or document, choosing the words to be used, generating the sentences expressing each concept, aggregating or merging several sentences in one to avoid redundancy, introducing pronouns to refer to entities previously mentioned, and finally, realize all that in appropriate and grammatical sentences. More details on NLG techniques can be found in [14].

In our process model scenario, we can not resort to pre-canned text, since each input model may be different. Given that our generated dialog has one state for each model task (see Section 4) we apply the *realization* step to obtain a sentence describing the task, and then we use this generated text as a pre-canned pattern at execution time.

4 Chatbot Generation from BPMN

To achieve our goal of generating a dialog agent from a process model in BPMN, we first define which kind of interactions the user is expected to have with the system, namely:

- Ask who is the actor performing any task.
- Ask to who (from who) is a message or a data object sent (received).
- Be guided step-by-step through the process:
 - Find out what is the next task to be executed (or a list of possible tasks, if several are possible) either by a particular actor or in the general process
 - Be asked to provide information when exclusive gateways are reached and be guided into the appropriate branch
 - Be informed when the process ends for a particular actor, or as a whole.

The purpose of these interactions is the use cases that may arise from this work, i.e., helping users to perform tasks of a process model. This type of interactions was required in a short collaboration we had with a process modelling software company. Other types of interactions are left for future work.

Given the expected flows of the dialog, we build a finite state automaton (FSA) that encodes the interactions and conversation flows that we focus in this paper.

The utterances that the system will produce when reaching each state in the FSA are generated analyzing the meaning of the text instances in the BPMN model (task labels and pool/swimlane names), and then feeding this semantic representation into a NLG system.

Also, a variety of patterns to match and interpret user response at each state are generated from model text, plus some general expressions valid for any process (such as “what is the next task?” or “end this conversation”).

Once the conversation FSA has been generated, it is encoded into AIML [17], so it can be executed in any available AIML interpretation engine. Figure 3 shows the main steps in the generation process, detailed in the following sections.

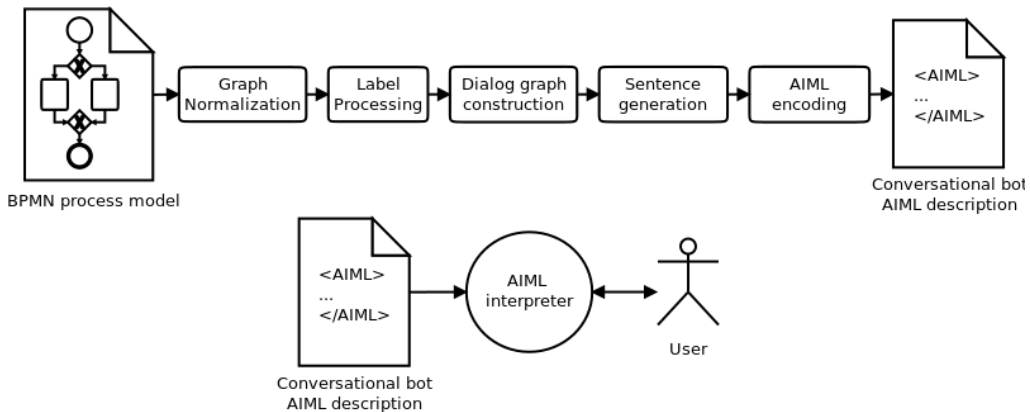


Fig. 3. Chatbot generation process stages (top). Once the chatbot description has been generated, it can be executed by AIML interpreter to interact with the user (bottom).

4.1 Graph Normalization

We start from the BPMN file, and we parse its XML format in order to load the process graph. This graph may require some normalization step, in order to ensure that all blocks in the graph are well-formed. In our case, we aim at having a BPMN that can be partitioned into *Single-Entry Single-Exit* (SESE) components [13]: for instance, the activities P1, P2 and P3 together with the two adjacent parallel gateways form a SESE in Figure 5(a). Several transformation techniques can be applied in case a process model is not well-formed (e.g., [12]). Hence, in the rest of this paper, we assume the process is well-formed.

4.2 Label Processing

Once the graph is normalized, we have to collect the linguistic information of model labels. We use FreeLing⁴ library [10] to desambiguate the part-of-speech of the label text, and to run it through a custom grammar that extracts the action, the object, as well as other complements. The subject is usually omitted in the task label, so it is retrieved from the pool or swimlane name.

For instance, the label *Retrieve parts from storage* in swimlane *Department* in Figure 1 would produce the semantic structure in Figure 4.

```
[ action: retrieve,
  subject: department,
  object: parts,
  complement: from storage ]
```

Fig. 4. Semantic structure produced by NLP analysis of the sentence *Retrieve parts from storage* in swimlane *Department* from Figure 1.

We use a custom grammar and not a general purpose natural language parser such as those provided by FreeLing or other similar library because of the particular structure of model task labels: Task labels are commonly written in simple patterns action-object (*retrieve parts*), or object-nominalized action (*parts retrieval*) with sometimes some additional complement(s) [11]. Also, the subject is usually omitted, which causes general purpose PoS taggers and parsers to fail more often. Having an ad-hoc grammar allows us to (1) control precisely which patterns should be detected, and (2) feed the parser with k most-likely PoS annotations from the tagger to find out if any of them matches the expected patterns, thus recovering from errors in the tagging step that would lead to wrong parsing results.

4.3 Dialog graph construction

Next step is generating the dialog graph, that is, the FSA that encodes all the possible dialog flows. This is a typical architecture followed by many simple chatbots, specially those based on AIML. The dialog graph consists of a set of states and transitions between them. Transition from one state to the next depends on the user utterance.

Definition 1. A dialog graph FSA is a tuple, $(Q, \mathcal{T}, \delta, A, \Omega)$, where:

- Q is a finite set of state nodes,
- \mathcal{T} is the set of all possible text utterances emitted by the user,
- $\delta : Q \times \mathcal{T} \rightarrow Q$ is a transition function that given the current state $q \in Q$ and a text utterance $t \in \mathcal{T}$ computes the destination state,

⁴ <http://nlp.cs.upc.edu/freeling>

$A \subseteq Q$ is the set of start state nodes, and
 $\Omega \subseteq Q$ is the set of final state nodes.

Note that the transition function δ does not work on a closed alphabet as in normal FSAs. Function δ may range from a simple set of regular expressions performing pattern matching on the user sentence, to a highly sophisticated language analysis system using the latest Artificial Intelligence techniques. In our case, since AIML supports only regular expression based transitions, we restrict ourselves to that approach, though with some extensions provided by the used interpreter (see Section 4.5).

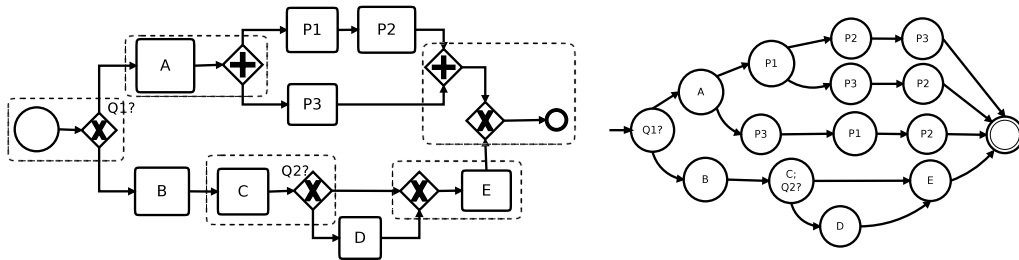


Fig. 5. Initial dialog state graph (right) corresponding to a BPMN model (left). Dotted lines show how split (join) gateways are fused with preceding (following) elements. Note the expansion of the parallel block into all its possible paths. Self-loops are added later to handle questions or commands valid in any state.

The created dialog graph has a structure that resembles the original BPMN graph, but with some differences to make it suitable for dialog control:

- **Join gateways:** In the BPMN semantics, join gateways describe the point where the branches of a previous split gateway are merged. This kind of node makes no sense in a dialog flow (it would be confusing that the system uttered “Now there is a join. what do you want to do?”). Thus, this kind of nodes are removed from the graph, and its entering edges are associated to the following element.
- **Parallel blocks:** A parallel block consists of the flow elements between a split and a join inclusive gateway. In BPMN, parallel block are interpreted as meaning that the involved tasks may be executed in any order. To account for this behavior in the dialog graph, we create a path in the dialog graph for each valid permutation of the tasks in the parallel branches. In this way, the user can choose the order in which she wants to perform the tasks⁵. Notice that as commented in Section 4.1, our strategy to transform parallel blocks (see below the formalization of the algorithm for this specific part) assumes

⁵ In case of a parallel block consisting on a large amount of parallel tasks, in principle it is not needed to offer all the permutations in the dialog graph, if that contributes to a state-space explosion: the artifact would be that the user only sees a subset of the possibilities offered by the process model, which in some cases can be acceptable.

that all parallel blocks in the BPMN are well-structured (a parallel block is *well-structured* when the number of branches going out the split gateway is equal to the number of branches entering the join gateway).

Figure 5 shows a simple example of the transformation of a BPMN model into a graph dialog that guides the user through the process.

The steps performed to recursively expand the parallel blocks and generate the corresponding dialog graph fragment are now overviewed. First of all, a depth-first search traversal is performed to detect split parallel gateways. When one is found, a new parallel block instance is pushed onto a general stack. The stack contains parallel blocks in depth-first order, because we need to guarantee the correct transformation of internal parallel blocks at every depth. Within a particular parallel scope, all the nodes encountered are added to the corresponding parallel branch. If the node is a join gateway, then all the active open branches of the containing parallel block instance are closed, and then the parallel block with the new expanded instance is replaced. For every parallel block detected, we check if there is any parallel block inside. If there is one, we call the function for that node. If there is no block, all the permutations of the branches of the selected block are created in the corresponding newly created FSM fragment. Then, these permutations are connected to the rest of the dialog graph.

Once the control-flow is completely transferred to the dialog graph, the last step of the construction is to also transfer the additional information contained in the BPMN model: messages, actors and data objects. These mainly correspond to self-loops on any conversation state, where information is reported to the user while retaining the conversation state (e.g. the user may ask *who did you say before that checks the purchase order?* even when the conversation is not in the state corresponding to this task).

4.4 Sentence generation

The dialog graph at this point has the definitive structure, but sentences that will be emitted by the system at each state have not been generated yet. To generate these sentences, we proceed in consecutive stages.

First, we create the syntactic specifications for each node. This step uses the semantic structures generated during label processing (Section 4.2). Using these annotations –and depending on the BPMN element type they correspond to– a syntactic structure is created with the appropriate characteristics (kind of sentence –affirmative, interrogative–, verb features –tense, person, ...–, modifiers, etc.) Note that some node types require a special treatment. For instance, at the process start node, the sentence will be headed by the text *The process starts when*, to give the user a better context information. Also, exclusive gateways will be generated as questions and not as affirmative sentences.

Each dialog node can have more than one syntactic structure. Also, the order of the structures can affect the way sentences are generated. The syntactic structures are provided to the *realization engine*, a module that applies syntactic, grammatical and morphological rules to produce a correct phrase with the requested features.

We use the realization engine provided by SimpleNLG⁶ library [4], an open-source project that uses basic English lexicon and grammar to transform the input into an appropriate sentence. One of the benefits of SimpleNLG is its potential to be adapted to other languages, using existing linguistic resources and performing some code adaptation.

SimpleNLG provides classes representing different kinds of phrases (verb phrase, noun phrase, prepositional phrase, etc). The calling application can instantiate any phrase specifying the desired features. SimpleNLG engine will build the sentences using grammar rules to properly combine the input phrase instances to form a valid syntactic tree. In our case, we build the phrases using the semantic structures previously created and we use them as required by the node type. Once the phrase instances are created, they are sent to the realization engine to obtain a full sentence.

The realization engine follows several steps to build the final sentence: First, the syntactic rules are applied to obtain the post-syntax tree. This decides, for example, the appropriate order for the words in the target language. Then, the morphological transformations –like selecting the correct determiners or the verb tense– are applied on the obtained tree. Finally, the last step is the orthography function, where sentence punctuation is revised and corrected. If there is some special format required, it is applied after these steps.

Once the sentences for each graph node are generated, we use them as basic information to create the message sentences and the questions:

When the model contains a message element, the user is asked to choose between continuing with the next task in the current lane, or to follow the message and see which task the message recipient will perform⁷. Message information is often encoded in the task originating it, and not in the message element itself, thus the generator needs to check both possibilities and decide which is the right text to use to generate sentences relative to message sending/receiving.

We also generate possible questions about the elements on the process. We resort to the same realization engine to produce questions about who is the responsible for each task, which is the object of an action, or who is the sender/recipient of a message. After some generalization to allow for variations, these questions are included in the set of regular expressions recognized by function δ . Also, this nodes are associated with the related task, so after asking, e.g. *who checks the purchase order* and getting the answer, the user may decide to follow the process from that point, or to remain in the current state.

4.5 AIML encoding

Once the dialog graph is complete and all the needed text has been generated, the dialog can be exported to the desired format to be interpreted by a chatbot engine.

⁶ <https://github.com/simplenlg>

⁷ This can only be done if the information is present in the original BPMN model.

We use *Artificial Intelligence Modeling Language* (AIML) standard because it is the conversational bot definition format most widely used. This XML-based format builds on the concepts of *topics*, which correspond to dialog states, and *categories* to represent the expected transitions from each state. Each category specifies a *pattern* –a regular expression to be matched with the user input–, and a *template* providing the answer the bot must emit and the new state to transition to. Since AIML basically describes extended FSAs, it is straightforward to convert our dialog graph into this format. AIML *patterns* allow for the use of wildcards that will match zero or more words in the user input, as well as *sets*, that allow specifying that a word in the user utterance may be any of a given list. We use both this mechanisms to add flexibility to user sentence interpretation, allowing for synonyms, or for extra words inserted in the user input. We pre-encode our *sets* in general synonym dictionaries extracted from WordNet [9].

AIML also supports some features over a pure FSA, such as the possibility of having internal variables to store any relevant information, that may be needed further along in the dialog (e.g. to store some user-provided information such as her name, or some other internal information not encoded in the state). In future versions of our bot generator this could be used, for instance, to ask the user which process role she wants to play, so when describing tasks executed by the selected role, the system would output e.g. *You check the purchase order* instead of *Central Purchasing checks the purchase order*.

4.6 AIML Interpretation

Once the AIML dialog definition file has been generated, it can be executed using any available AIML interpretation engine, so a user can actually interact with the bot.

Among the many open source available options, we use *ProgramY*⁸. It is maintained by AIML Foundation (who defines the evolution of the standard), and it is kept in sync with the latest standard updates. Also, it includes some useful additional features, like custom tags o full RegEx support, as well as a variety of front-ends to integrate the dialogs in different environments (standalone, web-based, Telegram, Twitter, Facebook, etc.).

5 Tool support

The methodology of this paper is available through the NLP4BPM platform [2], accessible at <https://bpm.cs.upc.edu/bpmninterface/>. Once logged in⁹, the user can go to the tab "BPMN to AIML" where a BPMN file can be uploaded and get as a result the AIML corresponding to the created chatbot, applying the

⁸ <https://github.com/keiffster/program-y>

⁹ An anonymous user is temporarily available for review purposes with the username "demo" and password "caisedemo".

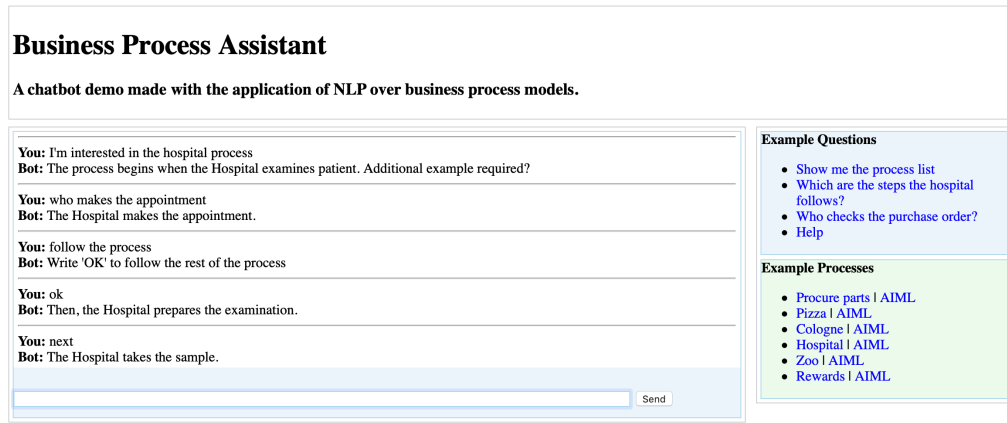


Fig. 6. Example of interpretation for several BPMN process models (available at <https://bpm.cs.upc.edu/chatbot>).

methodology of this paper. On the tab "Interpreting AIML" the user can upload the AIML generated to interact with the created chatbot.

For a fast insight on the contribution of this paper, we have set up an AIML interpreter demonstrating some generated chatbots for a collection of BPMN models, so that a user can interact with them. Figure 6 shows a screenshot of the environment, accessible at <https://bpm.cs.upc.edu/chatbot>.

6 Evaluation, Limitations and Use Cases

To evaluate the contribution of this paper, we collected feedback of 33 individuals from academia (27) or industry (6). After interacting with the chatbot for a couple of processes, the following questions were answered:

Q1: *How was your interaction with the chatbot ?* (1: not fluent – 4: very fluent)

Q2: *Did the Process Model Chatbot answer your questions about the process?* (1: it did not – 4: it did)

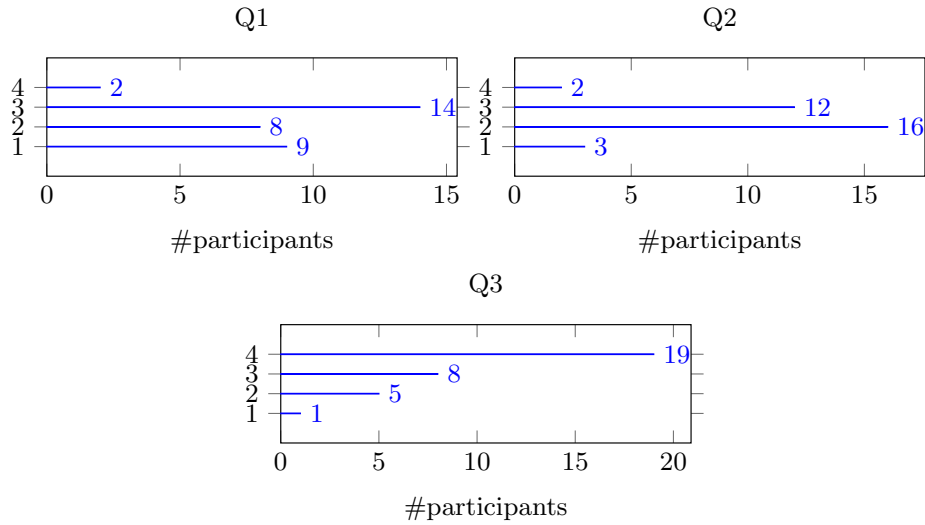
Q3: *Do you see potential for this kind of application in organizations?* (1: no potential – 4: large potential).

Two more informations were asked, were individuals could provide free text on the following two questions;

Q4: *What did you like / dislike about the tool ?*

Q5: *Do you have any suggestions in order to improve the Process Model Chatbot?*

From the answers to Q1-Q2, one can see that there is room for improvement in the implementation of our ideas: in both questions, more than half of the answers where on lowest scores. This is an artifact of the limited functionality of the current implementation, which lacks some flexibility and needs to be extended to be able to cover more parts of the process. In spite of this, through the answers to Q3 (81.8% agree on the huge potential of the ideas), we are confident that by improving theses weakness we will be able to come up with a solution that can be of practical use in organizations.



The answers to Q4-Q5 were an interesting source of ideas for improvement and encouragement, but confirmed the limited capabilities of the current implementation. Also, suggestions on use cases were provided, e.g., to help in the training of individuals, to help in the management of process changes, to have a state-aware dialogue between the actors and the process, among others.

7 Conclusions and Future Work

In this paper we have presented a fresh view on the interaction between processes and humans in organizations. By automating the translation between formal model notations like BPMN into conversational agents, a more flexible ecosystem is envisioned. This paper represents the first step towards the ambitious goal of empowering humans in organizations, so that decision-making is facilitated. We foresee multiple directions for future research, among which we highlight:

- Extend the capabilities of the interaction, either by extending the language (in our case, AIML), the types of BPMN constructs to consider, or the interpretation itself. Also, enable the interaction even when the user does not know the main activities of the process.
- Incorporate domain knowledge and/or other perspectives, e.g., data access rights, or security/privacy information.
- Create interactions at the level of process repositories.¹⁰

Acknowledgments We would like to thank Gero Decker for drawing our attention to the problem considered in this paper. This work has been supported by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R.

¹⁰ In <https://bpm.cs.upc.edu/chatbot> we provide very simple queries for detecting processes in a repository. One can think on a more elaborated setting where complex queries can be allowed.

References

1. Chen, H., Liu, X., Yin, D., Tang, J.: A survey on dialogue systems: Recent advances and new frontiers. *SIGKDD Explor. Newsl.* **19**(2), 25–35 (Nov 2017)
2. Delicado, L., Sànchez-Ferrerres, J., Carmona, J., Padró, L.: NLP4BPM - natural language processing tools for business process management. In: Proceedings of the BPM Demo Track co-located with 15th International Conference on Business Process Modeling (BPM 2017), Barcelona, Spain, September (2017)
3. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: *Fundamentals of Business Process Management*, Second Edition. Springer (2018)
4. Gatt, A., Reiter, E.: *SimpleNLG: a realisation engine for practical applications* (2009)
5. Jurafsky, D., Martin, J.H.: *Speech and Language Processing* (2Nd Edition). Prentice-Hall, Inc., Upper Saddle River, NJ, USA (2009)
6. Leopold, H., Mendling, J., Polyvyanyy, A.: Supporting process model validation through natural language generation. *IEEE Trans. Software Eng.* **40**(8), 818–840 (2014)
7. Matt, C., Hess, T., Benlian, A.: Digital transformation strategies. *Business & Information Systems Engineering* **57**(5), 339–343 (2015)
8. Mendling, J., Baesens, B., Bernstein, A., Fellmann, M.: Challenges of smart business process management: An introduction to the special issue. *Decision Support Systems* **100**, 1–5 (2017)
9. Miller, G.A.: Wordnet: A lexical database for english. *Commun. ACM* **38**(11), 39–41 (1995)
10. Padro, L., Stanilovsky, E.: Freeling 3.0: Towards wider multilinguality. In: International Conference on Language Resources and Evaluation. pp. 2473–2479 (May 2012)
11. Pittke, F., Leopold, H., Mendling, J.: Automatic detection and resolution of lexical ambiguity in process models. In: *Software Engineering 2016, Fachtagung des GI-Fachbereichs Softwaretechnik*, 23.-26. Februar 2016, Wien, Österreich. pp. 75–76 (2016)
12. Polyvyanyy, A., García-Bañuelos, L., Fahland, D., Weske, M.: Maximal structuring of acyclic process models. *Comput. J.* **57**(1), 12–35 (2014)
13. Polyvyanyy, A., Smirnov, S., Weske, M.: The triconnected abstraction of process models. In: *Business Process Management, 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009. Proceedings.* pp. 229–244 (2009)
14. Reiter, E., Dale, R.: *Building Natural Language Generation Systems*. Cambridge University Press, New York, NY, USA (2000)
15. Thorne, C.: Chatbots for troubleshooting: A survey. *Language and Linguistics Compass* **11**(10) (2017)
16. Traum, D.R., Andersen, C.F., Chong, W., Josyula, D.P., Okamoto, Y., Purang, K., O’Donovan-Anderson, M., Perlis, D.: Representations of dialogue state for domain and task independent meta-dialogue. *Electron. Trans. Artif. Intell.* **3**(D), 125–152 (1999)
17. Wallace, R.: *The Elements of AIML style*. ALICE AI Foundation (2003)
18. Wallace, R.S.: The anatomy of A.L.I.C.E. In: Epstein, R., Roberts, G., Beber, G. (eds.) *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer*, pp. 181–210. Springer Netherlands, Dordrecht (2009)
19. Weizenbaum, J.: ELIZA - a computer program for the study of natural language communication between man and machine. *Commun. ACM* **9**(1), 36–45 (1966)

Appendix VI

Unleashing Textual Descriptions of Business Processes

- **Publication Type:** Journal paper
- **Published In:** International Journal on Software and Systems Modeling
- **Impact Factor:** 1.915, Q1
- **Full Reference:** Josep Sànchez-Ferreres, Andrea Burattin, Josep Carmona, Marco Montali, Lluís Padró, and Luís Quishpi, “Unleashing textual descriptions of business processes”. In: *Software and Systems Modeling*. 2021. pp. 1-23. (Impact Factor: 1.915, Q1)

Unleashing Textual Descriptions of Business Processes

Josep Sànchez-Ferreres · Andrea Burattin ·
Josep Carmona · Marco Montali ·
Lluís Padró · Luís Quishpi

Abstract Textual descriptions of processes are ubiquitous in organizations, so that documentation of the important processes can be accessible to anyone involved. Unfortunately, the value of this rich data source is hampered by the challenge of analyzing unstructured information. In this paper we propose a framework to overcome the current limitations on dealing with textual descriptions of processes. This framework considers extraction and analysis, and connects to process mining via simulation. The framework is grounded in the notion of annotated textual descriptions of processes, which represents a middle-ground between formalization and accessibility, and which accounts for different modeling styles, ranging from purely imperative to purely declarative. The contributions of this paper are implemented in several tools, and case studies are highlighted.

Keywords Business Process Management · Natural Language Processing · Temporal Logics · Process Mining · Model Checking · Simulation

1 Introduction

Organizing business processes in an efficient and effective manner is the overarching objective of *Business Process Management* (BPM). Process specifications are the typical way of communicating how and in which order a given set of tasks or activities should be executed [17]. Several formal languages have been developed over the years to unambiguously define these processes. However, very often, these specifications are provided simply using natural language [32,33,2,46]. Due to

J. Sànchez-Ferreres, J. Carmona, L. Padró and L. Quishpi
Universitat Politècnica de Catalunya, Barcelona, Spain.
E-mail: {jsanchezf, jcarmona, padro, quishpi}@cs.upc.edu

A. Burattin
Technical University of Denmark, Copenhagen, Denmark.
E-mail: andbur@dtu.tk

M. Montali
Free University of Bozen-Bolzano, Bolzano, Italy.
E-mail: montali@inf.unibz.it

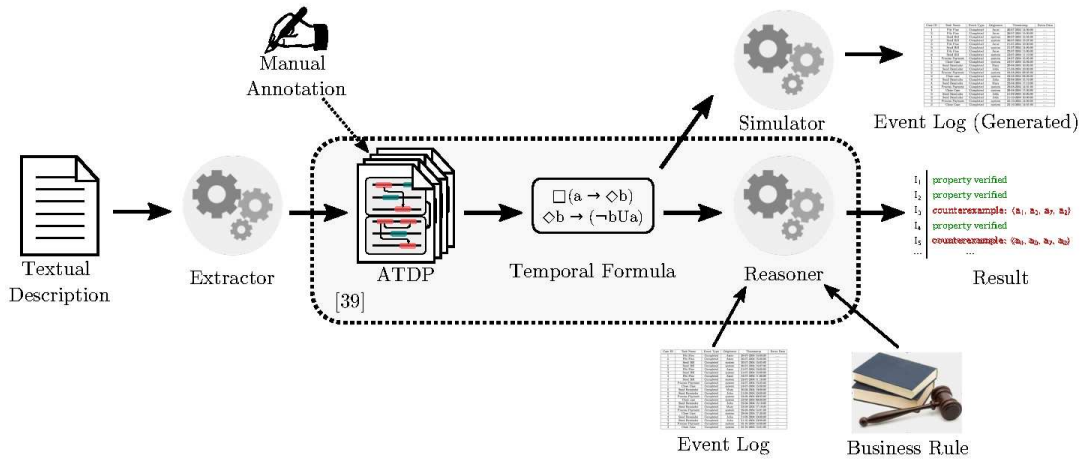


Fig. 1 General framework for extraction, analysis and simulation of textual descriptions of processes. The contributions introduced in [40] are highlighted.

the intrinsic ambiguities of natural languages, the exploitation of this information sources has proven to be a real challenge for organizations [49].

Taking into account the process information that is present in natural language texts brings completely fresh opportunities for organizations, such as enabling the connection between conceptual models and textual based representation of processes, or the ability to expose processes to a more general audience, among others. In spite of this, only in the past few years *Natural Language Processing* (NLP)-based analysis has been proposed in the BPM context, as reported in [26, 31, 30, 49].

This paper proposes a framework to enable formal process modelling on top of natural language. But more importantly, this paper also aims at opening a new direction for the use of textual description of processes in organizations: the possibility of enabling disciplines like verification, simulation or query answering on top of textual descriptions of processes (see Figure 1). This is possible due to the introduction of textual annotations, which can be partially extracted from a raw textual process description using advanced matching strategies [28] in combination with NLP, and for which a formal semantics using temporal logics over finite executions has been defined.

Part of the paper will be devoted to formalizing *Annotated Textual Descriptions of Processes* (ATDP). This formalism, originally introduced in [40], naturally enables the representation of a wide range of behaviors, ranging from procedural to completely declarative, but also hybrid ones. This is due to the notion of *scope*, that serves to frame the boundaries between declarative and imperative modelling strategies. Remarkably, and different from classical conceptual modeling principles, we have chosen to not solve but highlight ambiguities that can arise from a textual description of a process, so that a specification can have more than one possible interpretation.

ATDP specifications can be translated into linear temporal logic over finite traces [22, 12], opening the door to formal reasoning. The formal analysis proposed in this paper is mainly verification: to assess the compliance of a specification with respect to certain business rules. In addition to the obvious use of verification

to detect problems in a specification of a business process, it can be used to perform interpretation-aware reasoning. For instance, using verification one can select among the possible interpretations of an ATDP, which ones (if any) satisfy the reference business rules. Another use is to certify that any interpretation satisfies the reference business rules, in turn witnessing that the apparent flexibility in the process execution is not harmful.

Another interesting application of ATDP is simulation: to generate end-to-end executions (i.e., an *event log* [53]) that correspond to the underlying process. This would allow one to apply *process mining* techniques like *discovery*, so that a formal process model can be extracted using well-known discovery algorithms. For this, we introduce an approach combining state of the art techniques in the simulation of imperative and declarative process models [24].

This paper presents a revised and extended version of [40], including new relations and an iterated formalization, and a new positioning of the contribution with respect to related work. Additionally, the following new contributions are proposed in this paper:

- A new framework to automatically extract ATDP elements from textual descriptions.
- A new technique to simulate ATDP specifications to obtain event data, that has been tested on three realistic examples and validated using conformance checking techniques.
- A new open-source implementation of the set of techniques presented.

The rest of the paper is structured as follows: the next section positions this paper with respect to similar works in the literature. Then, Section 4 introduces the necessary ingredients for the understanding of the contributions of this paper. The formal description of ATDP specifications is provided in Section 5. Section 6 provides an overview of the current uses of ATDP specifications. Section 7 describes the tool-chain behind this work and illustrates its capabilities through some examples. Finally, Section 8 summarizes the achievements of this work and provides links for future work.

2 ATDP in a Nutshell

With the help of a realistic case, in this section we describe an example of ATDP specification. This will serve as a running example throughout the paper. Specifically, we use the textual description of the examination process of a Hospital, extracted from [45]. Figure 2 shows the full text, while Figure 3 contains a fragment of the visualization for an ATDP specification of the description.

One of the key features of the ATDP approach is the ability to capture *ambiguity*. In our example, we can see this at the topmost level: the text is associated with three different interpretations I_1 , I_2 and I_3 , providing three different process-oriented semantic views on the text. Each interpretation is a completely unambiguous specification of the process, which fixes a specific way of understanding ambiguous/unclear parts. Such parts could be understood differently in another interpretation. A specification in ATDP then consists of the union of all the valid interpretations of the process, which may partially overlap, but also contradict each other. For instance, in the example from Figure 3, interpretations I_1 and I_2 differ

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. 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. 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.

Fig. 2 Textual description of a patient examination process.

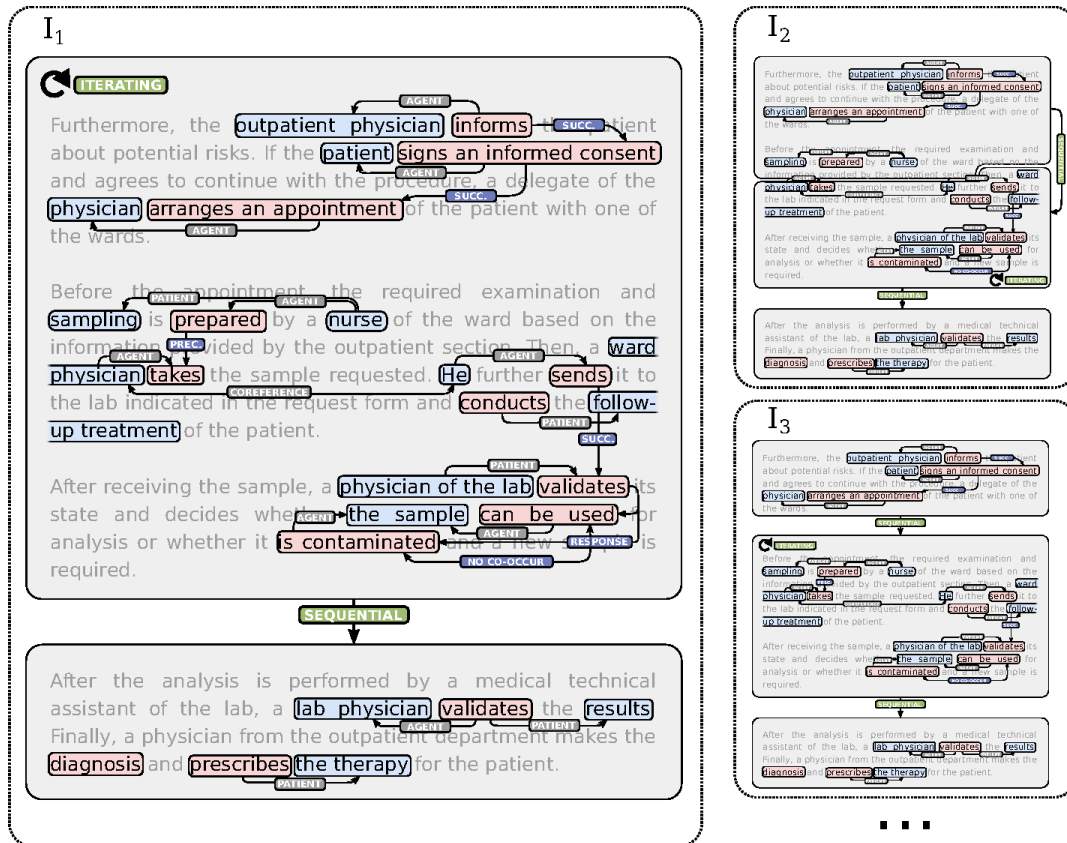


Fig. 3 Example annotation of a textual process description with multiple ambiguous interpretations. Some relations are omitted for brevity.

on the scope of the iteration. When a physician of the lab determines a sample is contaminated, the process needs to be restarted. The scope of this repetition is not clear from the text alone, and thus, it is not known whether a new appointment should be arranged or not. In spite of this ambiguity, there are common points in all interpretations of the text that allow for reasoning to be made, even when the process is not fully specified.

Each interpretation consists of a hierarchy of *scopes*, providing a recursive mechanism to isolate parts of the text that correspond to “phases” in the process. Each scope represents a conceptual block, which in turn may be decomposed into a set of lower-level scopes. Each scope dictates how its inner scopes are linked via control-flow relations expressing the allowed orderings of execution of such inner scopes. In our example, I_1 contains two scopes. A sequential relation indicates that the second scope is always executed when the first is completed, thus reconstructing the classical flow relation of conventional process modeling notation. All in all, the scope hierarchy resembles that of a process tree, following the variant used in [4].

Inside leaf scopes, *text fragments* are highlighted. There are different types of fragments, distinguished by color in our visual front-end. Some fragments (shown in red) describe the atomic units of behavior in the text, that is, activities and events, while others (shown in blue) provide additional perspectives beyond control flow. For example, `outpatient physician` is labelled as a *role* at the beginning of the text, while `informs` is labelled as an *activity*. Depending on their types, fragments can be linked by means of *relations*. Among such relations, we find:

Fragment relations that capture background knowledge induced from the text, such as for example the fact that the `outpatient physician` is the role responsible for performing (i.e., is the **Agent** of) the `informs` activity.

Temporal constraints linking activities so as to declaratively capture the acceptable courses of execution in the resulting process, such as for example the fact that `informs` and `signs an informed consent` are in *succession* (i.e., `informs` is executed if and only if `signs an informed consent` is executed afterwards).

As for temporal relations, we consider a relevant subset of the well-known patterns supported by the Declare declarative process modeling language [35]. In this light, ATDP can be seen as a multi-perspective variant of a process tree where the control-flow of leaf scopes is specified using declarative constraints over the activities and events contained therein. Depending on the adopted constraints, this allows the modeler to cope with a variety of texts, ranging from loosely specified to more procedural ones. At one extreme, the modeler can choose to nest scopes in a fine-grained way, so that each leaf scope just contains a single activity fragment; with this approach, a pure process tree is obtained. At the other extreme, the modeler can choose to introduce a single scope containing all activity fragments of the text, and then add temporal constraints relating arbitrary activity fragments from all the text; with this approach, a pure declarative process model is obtained instead.

3 Related Work

In order to automatically reason over a natural language process description, it is necessary to construct a formal representation of the actual process. The generation of a formal process model starting from a natural language description has

been investigated from several angles in the literature. We can classify these techniques along the spectrum of automation support: from fully manual to automatic.

The first available option consists in converting a textual description into a process model by manually modeling the process. This approach, widely discussed (e.g., [17,16]), has been thoroughly studied also from a psychological point of view, in order to understand which are the challenges involved in the “process of process modeling” [36,10]. These techniques, however, do not provide any automatic support, and the possibility for automated reasoning is completely dependant on the result obtained via the manual modeling. Therefore, ambiguities in the textual description are subjectively resolved.

On the opposite side of the spectrum, there are approaches that autonomously convert a textual description of a process model into a formal representation [21, 44,29]. In some cases, this representation is a final process model (e.g., using BPMN) [21,44,14]. Moreover, if the context of the process description is narrowed down (e.g., texts describing only cooking recipes [56], or phylogenetic analysis [23]), a more tailored extraction can be done. The limit of these techniques, however, is that they need to resolve ambiguities in the textual description, resulting in “hard-coded” interpretations. For instance, the presence of certain ambiguous phrases in a text such as “might” or “in the meantime” must be resolved into a concrete process model, forcing the system to take a specific interpretation.

There has been a recent focus on the extraction of process knowledge from textual descriptions, which is not necessarily aimed at providing formal process representations [20,38]. Clearly, by the use of recent deep AI techniques, the aforementioned frameworks have potential, but in order to be applicable, need to learn over a great amount of training data, a fact which hampers their use in a practical setting.

With a similar goal (autonomous generation of knowledge), but less related to the derivation of (formal) process representations, the analysis of textual descriptions to extract events, actions, states or state changes, sequences and similar knowledge has also been recently studied in different works [7,57,48,27,11,3]. Most of the aforementioned work is aimed at a more general problem (i.e., they are applicable even in textual descriptions that describe other phenomena, not only processes), so we believe they can be adapted to the particular case of extracting process knowledge.

In the middle of the spectrum, there are approaches that automatically process the natural language text and generate an intermediate artifact, useful to support consequent manual modeling by providing intermediate diagnostics [50,52,41,43,15]. The problem of having a single interpretation for ambiguities is a bit mitigated in this case since a human modeler is still in charge of the actual modeling. However, it is important to note that the system is biasing the modeler towards a single interpretation.

The approach presented in this paper drops the assumption of resolving all ambiguities in natural language texts. Therefore, if the text is clear and no ambiguities are manifested, then a precise process can be modeled. However, if this is not the case, instead of selecting one possible ambiguity resolution, our solution copes with the presence of several interpretations for the same textual description. The work presented in [51] also kept all process interpretations for the analysis, using the concept of *behavioral space* as a means to deal with the behavioral ambiguity of textual process descriptions. Interestingly, reasoning in [51] is casted as

checking compliance of an execution trace with respect to the behavioral space representation of a textual description. In contrast, our work presents a more general reasoning scheme, in which the aforementioned reasoning from [51] is possible, but also more general analyses like model checking, or even simulation, can be applied.

Another contribution of this paper is the simulation of ATDP. The generation of event logs from a process representation has already been investigated in the literature in the past. Logs generated from these systems can be used in several contexts, in particular within the process/data mining research communities, where having the golden standard (i.e., the reference model) is very important to improve the outcomes of mining algorithms.

One of the first techniques able to generate actual executions is reported in [13]. The main idea is to enrich a Petri net model with the information needed for simulation, using Colored Petri Net (CPN) tools as supporting infrastructure. The approach, though extremely flexible, is tailored to the simulation of Petri nets and the usage of the tool is also error-prone due to its intrinsic complexity. An improvement over this manual technique has been proposed in [5], where the author proposes a fully automatic technique capable of generating a Petri net or a model described in a subset on BPMN into a process mining-ready event log with support for data objects, representing not only the control-flow but the data perspective as well. In [24] a technique for the simulation of large populations of event trees [4] is reported. All the standard operators can be generated and, in addition, the data perspective can be generated as well, by consuming a DMN [1] model. Finally, [8] presents an approach for the generation of execution logs of Declare constraints. The technique first translates the declarative constraints into regular expressions and then generates an event log with possible executions compliant with the given set of constraints.

All these techniques suffer from the problem of being able to use only one specific type of models (either a Petri net, a BPMN, a Process Tree or a Declare model) as input. In the context of this paper, however, it is necessary to simulate specifications defined in a hybrid notation, i.e., imperative structures with declarative models as leaves. In addition, to the best of our knowledge, our simulation approach is the first one that accepts enriched textual descriptions as inputs.

4 Preliminaries

4.1 Linear Temporal Logics

In this paper, we use Linear Temporal Logic (LTL) [37] to define the semantics of the ATDP language. While LTL is traditionally defined over infinite traces, we adopt here a finite-trace interpretation, following [22, 12]. This matches the intuition that business process executions are all expected to reach, sooner or later, one of the end states defined by the process.

The resulting logic, called LTL_f , has the same syntax of LTL, but interprets formulae on linear models with finitely many time instants. At each instant, the model indicates which propositional symbols hold. Specifically, LTL_f formulae are built from a set \mathcal{P} of propositional symbols and are closed under the boolean connectives, the unary temporal operator \circ (*next-time*) and the binary temporal

operator U (*until*):

$$\varphi ::= a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \circ\varphi \mid \varphi_1 U \varphi_2 \quad \text{with } a \in \mathcal{P}$$

Intuitively:

- $\circ\varphi$ says that the *next* instance exists (i.e., we are not at the end of the trace), and in such a next instant φ holds.
- $\varphi_1 U \varphi_2$ says that at some future instant φ_2 will hold and *until* that point φ_1 always holds.

Common abbreviations used in LTL and LTL_f include the ones listed below:

- Standard boolean abbreviations, such as \top , \perp , \vee , \rightarrow .
- $\diamond\varphi = \top U \varphi$ says that φ will *eventually* hold at some future instant.
- $\square\varphi = \neg\diamond\neg\varphi$ says that from the current instant φ will *always* hold (until the last instance of the trace).
- $\varphi_1 W \varphi_2 = (\varphi_1 U \varphi_2 \vee \square\varphi_1)$ is interpreted as a *weak until*, and means that either φ_1 holds until φ_2 or in all instants of the trace.

It is important to stress that, even though LTL and LTL_f share the same syntax, the intended meaning of the same formula may radically change when moving from infinite to finite traces [12]. At the same time, it is possible to embed the finite-trace semantics of LTL_f into the standard LTL setting, at the price of adequately manipulating the traces and the formulae [12].

4.2 Natural Language Processing and Annotation

Natural Language Processing (NLP) is a wide research area within Artificial Intelligence that includes any kind of technique or application related to the automatic processing of human language. NLP goals range from simple basic processing such as determining in which language a text is written, to high-level complex applications such as Machine Translation, Dialogue Systems, or Intelligent Assistants. Given the recent advances in many NLP areas, there is an increasing interest in the applications and possibilities of these technologies to Business Process Management area [49].

Linguistic analysis tools can be used as a means to structure information contained in texts for its later processing in applications less related to language itself. This is our case: we use NLP analyzers to convert a textual description of a process model into a structured representation.

The NLP processing software used in this work is FreeLing¹ [34], 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).

¹ <http://nlp.cs.upc.edu/freeling>

- PoS-Tagging: Determine what 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 [19] 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.
- Semantic role labeling: 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.)
- 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*)

The three last steps are of special relevance since they allow the top-level predicate construction, and the identification of actors throughout the whole text: dependency parsing identifies syntactic subjects and objects (which may vary depending, e.g., on whether the sentence is active or passive), while semantic role labelling identifies semantic relations (the *agent* of an action is the same regardless of whether the sentence is active or passive). Coreference resolution identifies several mentions of the same actor as referring to the same entity (e.g. in Figure 3, *a delegate of the physician* and *the latter* refer to the same person, as well as the same object is mentioned as *the sample requested* and *it*).

Creating annotated versions of texts is customary in the NLP field, where many approaches are based on machine learning and require annotated text corpora both for training and for evaluating the performance of the developed systems. This need for annotated text has led the NLP community to develop several general-purpose annotation tools (e.g., Brat [47]). The next section shows how to describe processes by relying on textual annotations.

5 Processes as Annotated Textual Descriptions

One of the main elements in our framework is the *Annotated Textual Descriptions of Processes*, formalized as the ATDP language. We start this section by briefly introducing the foundational design principles of ATDP language (Section 5.1). Next, we present the core constructs of the language (Section 5.2) and formally define its semantics in LTL (Section 5.3).

Note that the aim of this section is not to provide an exhaustive enumeration of all patterns in ATDP, but rather to set a formal basis for the language allowing for future extensions to cover control-flow patterns in a more convenient way or help document other data-oriented aspects of a textual process description.

5.1 ATDP Design Principles

We have designed ATDP as a flexible modelling language that can capture well the subtleties of textual descriptions, while still remaining a formal representation to allow for automatic reasoning. During the design of ATDP, we have chosen to stick to the following design principles:

1. Models in ATDP are represented as annotations over plain text. This avoids misalignments between the informal plain text and the formal representation underneath.
2. ATDP combines imperative and declarative aspects of modelling notations. This ensures it can capture a wider range of behavioral constructs which we have found to naturally occur in textual process descriptions.
3. The language needs to directly address ambiguity, because most textual descriptions of processes contain some form of ambiguity. Partial reasoning must be possible even in the presence of ambiguity.
4. Automatic reasoning over models is paramount. This is why the formal semantics of ATDP are inspired by linear temporal logics and process trees.

5.2 ATDP Models

ATDP models are defined starting from an input text, which is split into *typed text fragments*. We now go step by step through the different components of our approach, finally combining them into a coherent model.

Fragment types. Fragments have no formal semantics associated by themselves. They are used as basic building blocks for defining ATDP models. We distinguish fragments through the following types.

Activity. This fragment type is used to represent the atomic units of work within the business process described by the text. Usually, these fragments are associated with verbs. An example activity fragment would be `validates (the sample state)`. Activity fragments may also be used to annotate other occurrences in the process that are relevant from the point of view of the control flow, but are exogenous to the organization responsible for the execution of the process. For instance, `(the sample) is contaminated` is also an activity fragment in our running example.

Role. The role fragment type is used to represent types of autonomous actors involved in the process, and consequently responsible for the execution of activities contained therein. An example is `outpatient physician`.

Business Object. This type is used to mark all the relevant elements of the process that do not take an active part in it, but that are used/manipulated by activities contained in the process. An example is the `(medical) sample` obtained and analyzed by physicians within the patient examination process.

When the distinction is not relevant, we may refer to fragments as the entities they represent (e.g. *activity* instead of *activity fragment*).

Given a set F of text fragments, we assume that the set is partitioned into three subsets that reflect the types defined above. We also use the following dot

notation to refer to such subsets: (i) F .activities for activities; (ii) F .roles for roles; (iii) F .objects for business objects.

Fragment relations. Text fragments can be related to each other by means of different non-temporal relations, used to express multi-perspective properties of the process emerging from the text. We consider the following relations over a set F of fragments.

Agent. An *agent relation* over F is a partial function

$$agent_F : F.\text{activities} \rightarrow F.\text{roles}$$

indicating the role responsible for the execution of an activity. For instance, in our running example we have $agent(\text{informs}) = \text{physician}$, witnessing that informing someone is under the responsibility of a physician.

Patient. A *patient relation*² over F is a partial function

$$patient_F : F.\text{activities} \rightarrow F.\text{roles} \cup F.\text{objects}$$

indicating the role or business object constituting the main recipient of an activity. For instance, in our running example, we have $patient(\text{prepare}) = \text{sample}$, witnessing that the **prepare** activity operates over a **sample**.

Coreference. A *coreference relation* over F is a (symmetric) relation

$$coref_F \subseteq F.\text{roles} \times F.\text{roles} \cup F.\text{objects} \times F.\text{objects}$$

that connects pairs of roles and pairs of business objects when they represent different ways to refer to the same entity. It consequently induces a coreference graph where each connected component denotes a distinct process entity. In our running example, all text fragments pointing to a **patient** role corefer to the same entity, whereas there are three different physicians involved in the text: the **outpatient physician**, the **ward physician** and the **physician of the lab**. These form disconnected coreference subgraphs.

Text scopes. To map the text into a process structure, we suitably adjust the notion of process tree used in [4]. In our approach, the blocks of the process tree are actually *text scopes*, where each scope is either a *leaf scope*, or a *branching scope* containing one or an ordered pair³ of (leaf or branching) sub-scopes.

Each activity is associated with one and only one leaf scope, whereas each leaf scope contains one or more activities, so as to non-ambiguously link activities to their corresponding process phases.

Each Branching scope, instead, is associated with a corresponding control-flow operator, which dictates how the sub-scopes are composed when executing the process. At execution time, each scope is enacted possibly multiple times, each time taking a certain amount of time (marked by a punctual scope start, and a later completion). We consider in particular the following scope relation types, together with their intuitive execution semantics:

² The term *patient*, as used in the formalization, is not related to the medical term used in the running example. Instead, it is borrowed from the related concept in the field of linguistics.

³ We keep a pair for simplicity of presentation, but all definitions carry over to n -ary tuples of sub-blocks.

- Sequential* (\rightarrow) A sequential branching scope s with children $\langle s_1, s_2 \rangle$ indicates that each execution of s amounts to the sequential execution of its sub-scopes, in the order they appear in the tuple. Specifically: (i) when s is started then s_1 starts; (ii) whenever s_1 completes, s_2 starts; (iii) the completion of s_2 induces the completion of s .
- Conflicting* (\times) A conflicting branching scope s with children $\langle s_1, s_2 \rangle$ indicates that each execution of s amounts to the execution of one and only one of its children, thus capturing a choice. Specifically: (i) when s is started, then one among s_1 and s_2 starts; (ii) the completion of the selected sub-scope induces the completion of s .
- Inclusive* (\vee) An inclusive branching scope s with children $\langle s_1, s_2 \rangle$ indicates that each execution of s amounts to the execution of at least one of s_1 and s_2 , but possibly both.
- Concurrent* (\wedge) A concurrent branching scope s with children $\langle s_1, s_2 \rangle$ indicates that each execution of s amounts to the interleaved, concurrent execution of its sub-scopes, without ordering constraints among them. Specifically: (i) when s is started, then s_1 and s_2 start; (ii) the latest, consequent completion of s_1 and s_2 induces the completion of s .
- Iterating* (\circlearrowleft) An iterating branching scope s with child s_1 indicates that each execution of s amounts to the iterative execution of s_1 , with one or more iterations. Specifically: (i) when s is started, then s_1 starts; (ii) upon the consequent completion of s_1 , then there is a non-deterministic choice on whether s completes, or s_1 is started again.

All in all, a *scope tree* T_F over the set F of fragments is a binary tree whose leaf nodes S_l are called *leaf scopes* and whose intermediate/root nodes S_b are called *branching nodes*, and which comes with two functions:

- a total *scope assignment* function $parent : F.\text{activities} \rightarrow S_l$ mapping each activity in F to a corresponding leaf scope, such that each leaf scope in S_l has at least one activity associated to it;
- a total *branching type* function $btype : S_b \rightarrow \{\rightarrow, \times, \vee, \wedge, \circlearrowleft\}$ mapping each branching scope in S_b to its control-flow operator.

Temporal constraints among activities. Activities belonging to the same leaf scope can be linked to each other by means of temporal relations, inspired by the Declare notation [35]. These can be used to declaratively specify constraints on the execution of different activities within the same leaf scope. Due to the interaction between scopes and such constraints, we follow here the approach in [18], where, differently from [35], constraints are in fact scoped.⁴

We consider in particular the following constraints:

Scoped Precedence Given activities a_1, \dots, a_n, b , $Precedence(\{a_1, \dots, a_n\}, b)$ indicates that b can be executed only if, within the same instance of its parent scope, at least one among a_1, \dots, a_n have been executed *before*.

Scoped Response Given activities a, b_1, \dots, b_n , $Response(a, \{b_1, \dots, b_n\})$ indicates that whenever a is executed within an instance of its parent scope, then at least one among b_1, \dots, b_n has to be executed *afterwards*, within the same scope instance.

⁴ It is interesting to notice that Declare itself was defined by relying on the patterns originally introduced in [18].

Scoped Weak Order Given two activities a, b , $\text{WeakOrder}(a, b)$ indicates that, whenever a and b are both present in a scope instance, a must appear always first. Further executions of a cannot occur without an execution of b in between. However, the execution of either a or b does not imply the other's. Later on, in Section 6.1.4 we justify the need for this constraint.

Scoped Non-Co-Occurrence Given activities a, b , $\text{NonCoOccurrence}(a, b)$ indicates that whenever a is executed within an instance of its parent scope, then b cannot be executed within the same scope instance (and vice-versa).

Scoped Alternate Response Given a sequence of activities a, b_1, \dots, b_n , $\text{AlternateResponse}(a, \{b_1, \dots, b_n\})$ indicates that whenever a is executed within an instance of its parent scope, then a cannot be executed again until, within the same scope, at least one among b_1, \dots, b_n is eventually executed.

Terminating Given activity a , $\text{Terminating}(a)$ indicates that the execution of a within an instance of its parent scope terminates that instance.

Initial Given activity a , $\text{Initial}(a)$ indicates a must be the first activity executed in its scope.

Mandatory Given activity a , $\text{Mandatory}(a)$ indicates that the execution of a must occur at least once for each execution of its scope.

Interpretations and models. We are now ready to combine the components defined before into an integrated notion of text interpretation. An *ATDP interpretation* I_X over text X is a tuple $\langle F, \text{agent}_F, \text{patient}_F, \text{coref}_F, T_F, C_F, \rangle$, where: (i) F is a set of *text fragments* over X ; (ii) agent_F is an *agent function* over F ; (iii) patient_F is a *patient function* over F ; (iv) coref_F is a *coreference relation* over F ; (v) T_F is a *scope tree* over the activities in F ; (vi) C_F is a set of *temporal constraints* over the activities in F , such that if two activities are related by a constraint in, then they have to belong to the same leaf scope according to T_F .

An *ATDP model* M_X over text X is then simply a finite set of ATDP interpretations over X .

5.3 ATDP Semantics

We now describe the execution semantics of ATDP interpretations, in particular formalizing the three key notions of scopes, scope types (depending on their corresponding control-flow operators), and temporal constraints over activities. This is done by using LTL_f , consequently declaratively characterizing those execution traces that conform to what is prescribed by an ATDP interpretation. We consider execution traces as finite sequences of atomic activity executions using interleaving semantics to represent concurrency.

Scope Semantics.

To define the notion of scope execution, for each scope s , we introduce a pair of artificial activities st_s and en_s which do not belong to F .activities. The execution of s starts with the execution of st_s , and ends with the execution of en_s . The following axioms define the semantics of scopes:

A1. An activity a inside a scope s can only be executed between st_s and en_s :

$$\neg a W st_s \wedge \Box(en_s \rightarrow \neg a W st_s)$$

A2. A scope s can only be started and ended inside of its parent s' :

$$\neg(st_s \vee en_s) W st_{s'} \wedge \Box(en_{s'} \rightarrow \neg(st_s \vee en_s) W st_{s'})$$

A3. Executions of the same scope cannot overlap in time. That is, for each execution of a scope s 's start there is a unique corresponding end:

$$\begin{aligned} & (\neg en_s W st_s) \wedge \Box(st_s \rightarrow \Diamond en_s) \wedge \\ & \Box(st_s \rightarrow \circ(\neg st_s U en_s)) \wedge \\ & \Box(en_s \rightarrow \circ(\neg en_s U st_s)) \end{aligned}$$

A4. Iterating scopes may be affected by the presence of terminating activities, as defined by the following property: A terminating activity a_t inside an iterating scope s , child of s' , stops the iteration. That is, the execution of s cannot be repeated anymore inside its parent:

$$\Box(st_{s'} \rightarrow ((a_t \rightarrow (\neg st_s U en_{s'})) U en_{s'}))$$

Temporal Constraint Semantics. Next, we define the semantics of temporal constraints between activities. Note that, in all definitions we will use the subindex s to refer to the scope of the constraint.

$$\begin{aligned} \text{Precedence}_s(\{a_1, \dots, a_K\}, b) & := \bigvee_{i=1}^K \Box(st_s \rightarrow (\neg b W a_i)) \\ \text{Response}_s(a, \{b_1, \dots, b_N\}) & := \bigvee_{i=1}^N \Box(st_s \rightarrow (a \rightarrow (\neg en_s U b_i))) \\ \text{NonCoOccurrence}_s(a, b) & := \Box(st_s \rightarrow (a \rightarrow (\neg b U en_s))) \wedge \\ & \quad \Box(st_s \rightarrow (b \rightarrow (\neg a U en_s))) \\ \text{AlternateResponse}_s(a, b) & := \text{Response}_p(a, b) \wedge \\ & \quad \Box(st_s \rightarrow (a \rightarrow \circ(\neg a U (b \vee en_s)))) \\ \text{WeakOrder}_s(a, b) & := \Box(st_s \rightarrow ((\neg en_s W a \wedge \neg en_s W b) \rightarrow \\ & \quad \text{AlternateResponse}_s(a, b))) \\ \text{Terminating}_s(a) & := \Box(a \rightarrow \circ en_s) \\ \text{Initial}_s(a) & := \Box(st_s \rightarrow \circ a) \\ \text{Mandatory}_s(a) & := \Box(st_s \rightarrow (\neg en_s U a)) \end{aligned}$$

Scope Relation Semantics. In all our definitions, let $\langle s_1, s_2 \rangle$ denote the children of a branching scope s , associated to the control-flow operator being defined. Note that by $\text{Sequence}(a, b)$ we refer to the formula $\text{Precedence}(\{a\}, b) \wedge \text{Response}(a, \{b\})$ and the \oplus operator is the logical exclusive or.

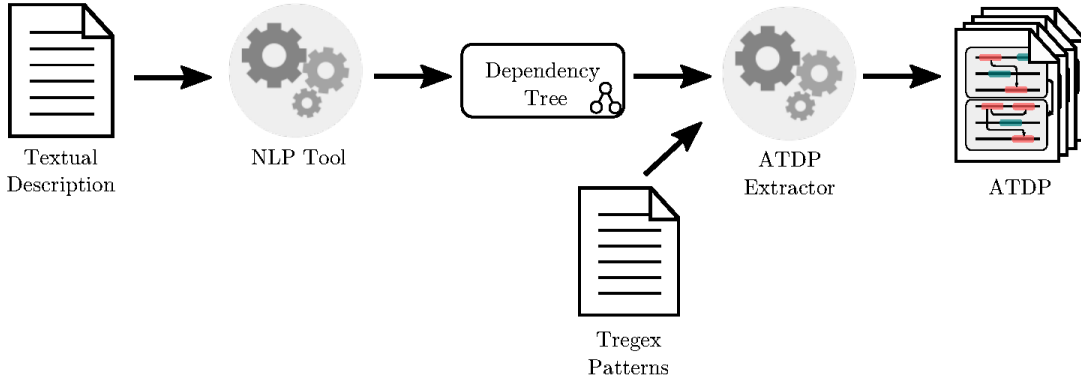


Fig. 4 Overview of automatic extraction.

Sequential (\rightarrow) : $\text{Sequence}_s(en_{s_1}, st_{s_2}) \wedge \text{Mandatory}_s(st_{s_1}) \wedge \text{Mandatory}_s(st_{s_2})$

Conflicting (\times) : $\Box(st_s \rightarrow (\neg en_s U st_{s_1}) \oplus (\neg en_s U st_{s_2}))$

Inclusive (\vee) : $\Box(st_s \rightarrow (\neg en_s U st_{s_1}) \vee (\neg en_s U st_{s_2}))$

Concurrent (\wedge) : $\Box(st_s \rightarrow (\neg en_s U st_{s_1}) \wedge (\neg en_s U st_{s_2}))$

Iterating (\odot) : This relation is defined by negation, with any non-iterating scope s , child of s' , fulfilling the property:

$$(st_{s'} \rightarrow (\neg en_{s'} U st_s \wedge (st_s \rightarrow \circ(\neg st_s U en_{s'})) U en_{s'}))$$

6 Using ATDP Specifications in Organizations

We now revisit more carefully the general overview of the current uses and required steps for using ATDP specifications in organizations, that were overviewed in the introduction and depicted in Figure 1. Starting from a textual description, a combination of automatic and manual annotation would enable extracting an ATDP. In Section 6.1 we propose a strategy to automate the extraction of ATDP specifications so that only a limited manual effort is required. Once an ATDP specification is obtained, different alternatives are possible. If reference business rules are available, one can cast the verification of these rules with the specification as a model checking instance, as it is fully described in Section 6.2. Finally, one can simulate the ATDP specification into a new event log, which then paves the way towards process mining practices like discovery, as seen in Section 6.3

6.1 Automating the Extraction of ATDP through Tree Query

In this section, we provide the necessary concepts and a general overview on how to automatically extract some elements of the ATDP for a textual description of a process. Figure 4 illustrates the main steps of the methodology: given a textual

Operator	Meaning
A << B	A dominates B
A >> B	A is dominated by B
A < B	A immediately dominates B
A > B	A is immediately dominated by B
A >- B	A is the last child of B
A >: B	A is the only child of B
A \$-- B	A is a right sister of B

Table 1 Basic operators to create Tregex patterns

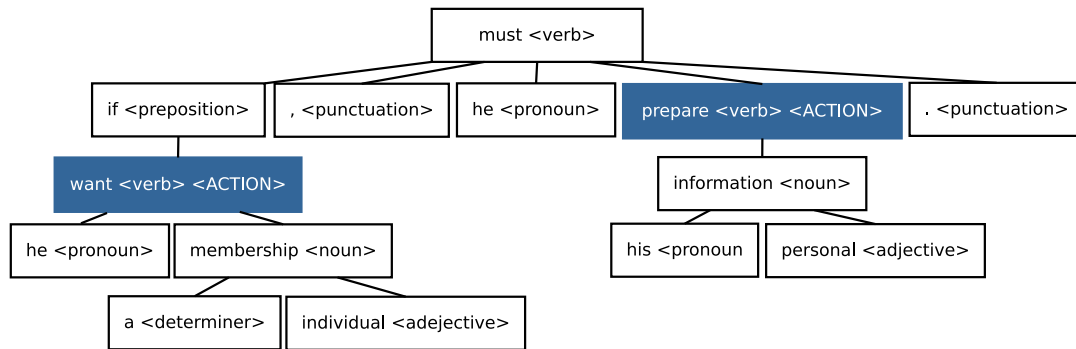


Fig. 5 Dependency tree representation used to apply patterns.

description, NLP analysis is done to extract, among other information, a dependency tree for each sentence (see Section 4.2). Then tree query is performed on the forest of dependency trees arising from a textual description; to accomplish this, tree patterns match parts of a dependency tree that reflect with high confidence a certain ATDP fragment or relation. The remainder of this section will provide the necessary details to accomplish this step.

Tregex⁵ [28] is a pattern matching algorithm that allows matching regular-expression-like patterns on tree structures. If applied on a dependency tree, this technique can search for complex labeled tree dominance relations involving different types of information in the nodes (e.g. PoS tags, word forms, lemmas). A Tregex pattern is a regular expression-like pattern that is designed to match node configurations within a tree where the nodes are labeled, and the expression combines those labels via a set of operators. The basic operators used in this work to specify Tregex tree queries are listed in Table 1.

To be able to search for ATDP elements, we transform the dependency tree nodes obtained from the NLP analyzers to a format suitable for Tregex patterns: A node in the transformed dependency tree is a structured string, containing information about the lemma and PoS tag of each word. Additionally, some nodes include an <ACTION> label to mark nodes corresponding to activity fragments in the process model ATDP, based on the results from the semantic role labelling step. Figure 5 shows an example of such tree for the input sentence “*If he wants an individual membership, he must prepare his personal information.*” The nodes for

⁵ See <https://nlp.stanford.edu/software/tregex.html>

the verbs `want` and `prepare` contain [`want <verb> <ACTION>`] and [`prepare <verb> <ACTION>`], respectively.

In each generated dependency tree we apply a cascade of several Tregex patterns, each of which detects a particular ATDP element. Below we provide several examples of tree queries to extract specific ATDP elements. The description of the whole set of tree queries to extract the rest of ATDP elements is left out of this work for the sake of space.

The techniques described in this section can be used to quickly bootstrap an initial annotation of a textual description. We have found the proposed techniques to be empirically accurate for the chosen domains and writing styles. However, it is still required for a domain expert to audit the output of the automatic annotator, and very different writing styles or document structures may require a partial redesign of the extraction rules.

6.1.1 Extraction of activity fragments

Activity fragments represent the units of execution inside the process model. In order to extract activity fragments we rely on the output of FreeLing NLP processing, which marks as predicates all non-auxiliary verbs as well as some nominal predicates (e.g. *reception*, *delivery*, etc). However, many verbs in a process description may be predicates from a linguistic perspective, but do not correspond to actual process activities. Thus, we use a set of patterns that discard predicates unlikely to be describing a relevant process task. Some examples are the following:

- `/want/=toRemove`

This pattern simply removes the verb *want* as an activity. Subjective verbs (e.g. *want*, *think*, *believe*) are unlikely to describe activities and thus are filtered out. For instance in the sentence “*If the customer wants an individual ticket, he must prepare his personal information*”, *wants* is removed from the activity list, while *prepare* is kept.

- `/<ACTION>/=toRemove >> /<ACTION>/=result !>> /and/or/`

The second pattern removes any action candidate that is in a subordinate clause under another action. The idea is that a subordinate clause is describing some details about one actor in the main clause, but not a relevant activity. For instance, in the sentence “*the examination is prepared based on the information provided by the outpatient section*”, the verbs *base* and *provide* would be removed as activities, since the main action described by this sentence is just *prepare* (*examination*). The pattern has an additional constraint checking that the tree does not contain a coordinating conjunction (*and/or*), since in that case, both predicates are likely to be activities (e.g. in “*He sends it to the lab and conducts the follow-up treatment*”, although *conduct* is under the tree headed by *send*, the presence of *and* in between blocks the rule application).

6.1.2 Extraction of role entity fragments

To identify the roles –or autonomous actors– of the process we leverage the results from the NLP analysis and focus on the elements with a semantic role of *Agent*. For each of those elements, the extracted text should be modified to better represent the role: fragments that begin with *the* or prepositions such as *by*, *of* or *from*

can be modified to not contain these elements. For example, in the sentence “*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*” the results of the semantic role labelling step would return the whole subtree headed by *physician* (i.e. *an outpatient physician, who decides...*). The role entity fragments rule will strip down such a long actor removing the determiner and the relative clause, while keeping the core actor and its main modifiers: `outpatient physician`. To that end, the following Tregex pattern is recursively applied to the dependency tree to obtain the relevant modifiers of the main entity word:

```
– /noun|adjective/=result > /mainEntityWord/
```

6.1.3 Extraction of conflict temporal relations

Conflict relations naturally arise when conditions are introduced in a process description. In ATDP the only conflict relation is `NonCoOccurrence`. To that end, we consider discourse markers that mark conditional statements, like: *if*, *whether*, *either* and *in case of*. Each discourse marker needs to be tailored to a specific grammatical structure.

For instance, the following pattern would extract the knowledge that the sample can't both be safely used and contaminated at the same time from the sentence “*she decides whether the sample can be used for analysis or whether it is contaminated*”:

```
– /whether/ << (<ACTION>/=origin << (/or/ << /<ACTION>/=destination))
```

6.1.4 Extraction of some order temporal relations

The `Precedence`, `Response` and `WeakOrder` constraints are used to express the order of execution of the activities in an ATDP specification. This first pattern can be used to identify a particular case of `Response` that typically occurs in conditional sentences:

```
– /<ACTION>/=destination >: (/if/ > /<ACTION>/=origin)
```

The pattern captures the case where an identified condition is inside an *if* clause (that is, below the *if* token in the dependency tree), which has a candidate action as the condition's consequent. In those cases, it is safe to assume that the action in the consequent *responds to* the occurrence of the condition. For example, in the sentence “*If the bank confirms the payment request, the total amount is then charged to the user account.*”, this rule would extract the knowledge that `charge (total amount)` responds to `confirm (payment request)`.

For more general cases, the subtleties between the different order constraints cannot be easily distinguished by an automatic analyzer. In those cases, we take a conservative approach and extract the least restrictive constraint, `WeakOrder`. To illustrate this, we can infer that `generates` and `pays` are in `WeakOrder` in the sentence “*The Payment Office of SSP generates a payment report and then pays the vendor*” by using the following pattern:

```
– /<ACTION>/=origin < (/and/ << (/<ACTION>/=destination < /then/))
```

Using this kind of local information alone, it is not safe to infer neither `Precedence` nor `Response` in the above example. We thus defer the strengthening of these constraints to a manual annotator.

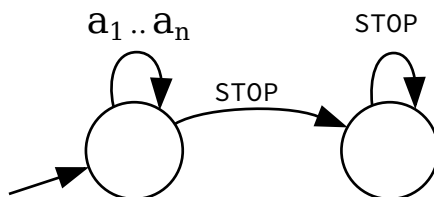


Fig. 6 Automata representation of the transition system used for the model checker encoding. Activities in $a_1..a_n$ represent both the activity fragments modelled by the user and the start/end activities for every scope in the ATDP specification.

6.2 Reasoning on ATDP Specifications

A specification in ATDP is the starting point for reasoning over the described process. This section shows how to encode the reasoning as a model checking instance, so that a formal analysis can be applied on the set of interpretations of the model. Furthermore, we present three use cases in the scope of business process management: *checking model consistency*, *compliance checking* and *conformance checking*.

6.2.1 Casting Reasoning as Model Checking

Reasoning on ATDP specifications can be encoded as an instance of model checking, which allows performing arbitrary temporal LTL_f queries ((cf. Section 5.3) on the model. We do this in two steps.

As a first step, we observe that determining whether an ATDP specification entails a given LTL_f formula Q can be encoded as standard LTL_f *satisfiability checking* for the following formula:

$$(A \wedge \mathcal{C}_F \wedge \mathcal{C}_{T_F}) \implies Q \quad (1)$$

where A is the conjunction of all LTL_f formulae defined by the axioms, \mathcal{C}_F is the conjunction of the activity temporal constraints, \mathcal{C}_{T_F} is the conjunction of all LTL_f defined by the semantics of the process tree.

As a second step, we recall that satisfiability via model checking can in principle be realized, as described in [39], by building a so-called *universal transition system* that generates all possible traces over a given alphabet, then verifying whether such system satisfies the formula of interest (in our case, (1)).

We adopt this idea to encode reasoning on ATDP processes into the well-established NuSMV model checker [9]. To make the approach operationally correct, we have to consider two crucial aspects of our approach:

(Interleaving semantics) At each moment in time, only one propositional symbol is true, witnessing that the corresponding activity is executed.

(Finite-trace semantics) Formula (1) is interpreted over finite traces, whereas NuSMV natively works over infinite traces.

We tackle these two aspects as follows. The *universal transition system* is constructed following the schema of the state machine (with activity-labeled edges) in Figure 6.

Essentially, the universal transition system picks an arbitrary number of times activities to be executed (at each time, just one activity is actually executed, as

the interleaving semantics dictates). Then, a special **STOP** activity (not present in the original **ATDP**, but introduced artificially) signals that the trace has reached its end, and once this occurs, then **STOP** is repeated forever. If we ensure that, on top of this transition system, **STOP** is eventually selected, then the resulting universal transition system generates infinite traces containing an initial, *finite* prefix with genuine activities, followed by an infinite suffix where **STOP** is repeated forever.

On top of this universal transition system, we then verify a variant of formula (1) defined as follows:

$$(A \wedge \mathcal{C}_F \wedge \mathcal{C}_{T_F} \wedge \Diamond \text{STOP}) \implies f(Q) \quad (2)$$

This formula differs from (1) in two respects. First, it contains $\Diamond \text{STOP}$ in the body of the implication, so as to enforce that the universal transition systems generates trace of the shape described above (i.e., finite trace prefixes followed by an infinite repetition of **STOP**). Second, it does not use Q directly, but instead applies a translation function f to it. This translation function is needed because Q is an LTL_f formula, while NuSMV adopts LTL over infinite traces. In fact, f is inductively defined as in [22, 12], in such a way that the original LTL_f formula becomes a corresponding LTL formula that can be checked over the universal transition system defined above:

$$\begin{aligned} f(a) &= a \\ f(\circ\varphi) &= \circ(f(\varphi) \wedge \neg \text{STOP}) \\ f(\neg\varphi) &= \neg f(\varphi) \\ f(\varphi_1 U \varphi_2) &= f(\varphi_1) U (f(\varphi_2) \wedge \neg \text{STOP}) \\ f(\varphi_1 \wedge \varphi_2) &= f(\varphi_1) \wedge f(\varphi_2) \end{aligned}$$

One may wonder why this translation function is not applied to the entire formula, but only to Q . The reason is that while Q is an arbitrary LTL_f formula, the premise of the implication of (2) has a fixed shape, determined by the LTL_f encoding of the **ATDP** semantics, and this shape is so that the formula and its translation are semantically equivalent. This means that the premise is not able to distinguish finite from infinite formulae or, using the technical terminology introduced in [12], that the premise is “insensitive to infinity”.

Non-temporal information can be introduced in the queries without increasing the problem complexity, since the information is statically defined. For example, when the text mentions that several activities are performed by a certain role, this information remains invariant during the whole model-checking phase. Thus, queries concerning roles can be translated directly into queries about the set of activities performed by that role. A possible encoding of this into a model checker consists of adding additional variables during the system definition.

When dealing with multiple interpretations, the above framework is extended with two types of queries:

Existential: Is the proposition true in any interpretation of the process?

$$\exists I \in \text{ATDP} : (A_I \wedge \mathcal{C}_{F_I} \wedge \mathcal{C}_{T_{F_I}}) \implies Q$$

Complete: Is the proposition true in all interpretations of the process?

$$\forall I \in \text{ATDP} : (A_I \wedge \mathcal{C}_{F_I} \wedge \mathcal{C}_{T_{F_I}}) \implies Q$$

Existential and complete queries can be used to reason in uncertain or incomplete specifications of processes.

An application of complete queries would be finding invariant properties of the process. That is, a property that holds in all possible process interpretations.

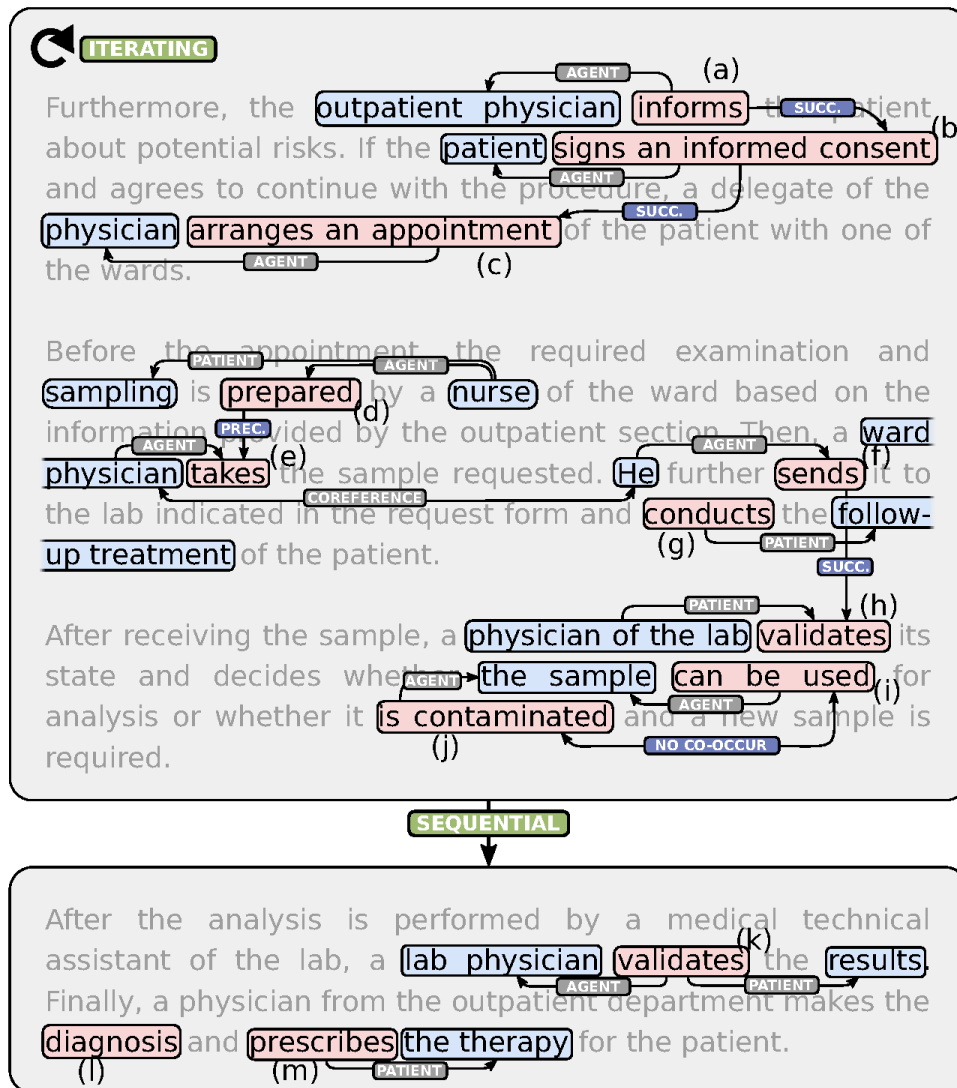


Fig. 7 Scope \mathcal{I}_1 from the ATDP specification of Figure 3, inserting short letter aliases per activity. Some relations have been omitted for brevity.

Existential queries, in turn, fulfill a similar role when the proposition being checked is an undesired property of the process. By proving invariant properties, it is possible to extract information from processes even if these are not completely specified or in case of contradictions. A negative result for this type of query would also contain the non-compliant interpretations of the process, which can help the process owner in gaining some insights about which are the assumptions needed to comply with some business rule.

6.3 Simulation of ATDP Specifications

The simulation of a process model consists of generating event logs that conform to the process description. The generated logs can be used to infer statistical information from the processes described therein. Another common use case for

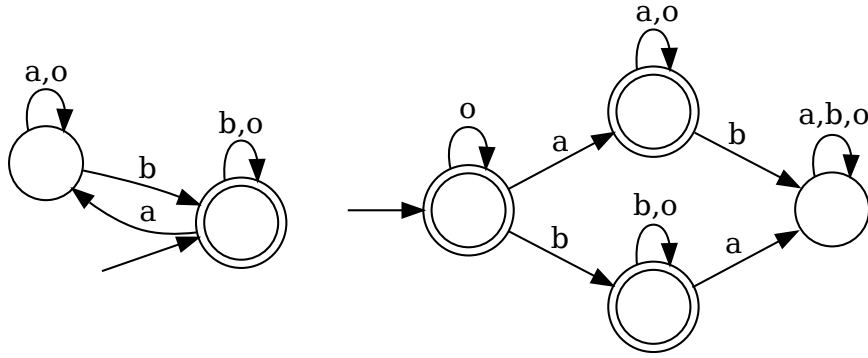


Fig. 8 Automata implementing two ATDP constraints: $\text{Response}(a, b)$ (left) and $\text{NonCoOccurrence}(a, b)$ (right). The symbol o is used to indicate any other activities in the scope not affected by this constraint.

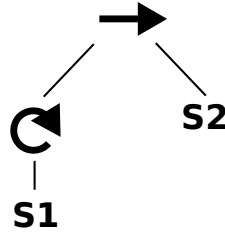


Fig. 9 Process Tree visualization of the scope-level behaviour of the ATDP in Figure 7

simulation is the evaluation of process mining algorithms. In this section we detail how to generate event logs from ATDP specifications.

As previously mentioned, ATDP specifications can be seen as a multi-perspective variant of process trees, where each of the leaf elements are declarative process models instead of atomic activities. Thus, in order to implement the simulation of ATDP, we combine two well-known algorithms [24,8] for the simulation of Process Trees and Declare models respectively. An outline of the algorithm is shown in Algorithm 1.

Simulation of Process Trees follows the implementation described in [24]. The algorithm simulates each of the tree operator nodes by recursively simulating its children, and combining the traces that are obtained according to the semantics of each operator. Following our running example, let us focus on the scope \mathcal{I}_1 , shown in Figure 7. If the first scope generates trace $t_1 = \langle a, b, c, d, e, g, f, h, i \rangle$ and the second scope generates trace $t_2 = \langle k, l, m \rangle$. The *Sequential* operator between the first and second scope would combine t_1 and t_2 by concatenating them. To better illustrate this example, Figure 9 shows the Process Tree corresponding to interpretation I_1 of Figure 3. Other operators are implemented in a similar fashion: The *Exclusive* operator would choose one of t_1 and t_2 at random, and the *Concurrent* operator would randomly interleave t_1 and t_2 . See [24] for more details on how to simulate process trees.

Being closely related to the Declare modeling language, the simulation of Leaf Scopes is implemented by following the algorithm described in [8]. The algorithm works by implementing the Declare semantics with regular expressions. Each of the temporal constraints defines a Deterministic Finite Automaton (DFA). For instance, Figure 8 shows the resulting automata for two basic Declare constraints

Algorithm 1 Simulation of an ATDP

```

def simulate_scope(scope):
    result = []

    if scope.type == "Leaf":
        constraints = scope.constraints
        automata = [constraint_to_automaton(c)
                    for c in constraints]
        leaf_automaton = intersect_all(automata)
        result = random_walk(leaf_automaton)

    else if scope.type == "Iterating":
        while not stop_criterion():
            result = concatenate(result,
                                simulate_scope(scope.children[0]))

    else:
        left_trace = simulate_scope(scope.children[0])
        right_trace = simulate_scope(scope.children[1])

        if scope.type == "Sequential":
            return concatenate(left_trace, right_trace)

        else if scope.type == "Conflicting":
            return choose_one(left_trace, right_trace)

        else if scope.type == "Concurrent":
            return interleave_at_random(left_trace,
                                       right_trace)

        else if scope.type == "Inclusive":
            return choose_one(
                left_trace,
                right_trace,
                interleave_at_random(left_trace, right_trace))

    return result

```

also present in ATDP. The DFA for all temporal constraints in the leaf scope are then intersected to form an automaton that accepts the regular language of the traces accepted by that scope. By performing random walks on this automaton, random traces can be obtained that conform to each Leaf Scope specification. By random-walking the automaton for $\text{Response}(a,b)$ in Figure 8, one may obtain the traces $t_3 = \langle a, b \rangle$ and $t_4 = \langle a, a, b, a, b \rangle$ but not $t_5 = \langle a, b, a \rangle$.

7 Tool Support and Use Case Examples

The techniques described in this paper have been implemented and are available as five standalone tools:

- The ATDP library. The library is available at <https://github.com/PADS-UPC/atdplib-model>.
- The ATDP extractor, which extracts ATDP elements from textual descriptions, using tree query techniques on the result of NLP analysis. The tool is available at <https://github.com/PADS-UPC/atdpextractor>.
- The ATDP reasoner, which translates ATDP specifications into the model checking language for NuSMV tool, thus enabling to use this model checking envi-

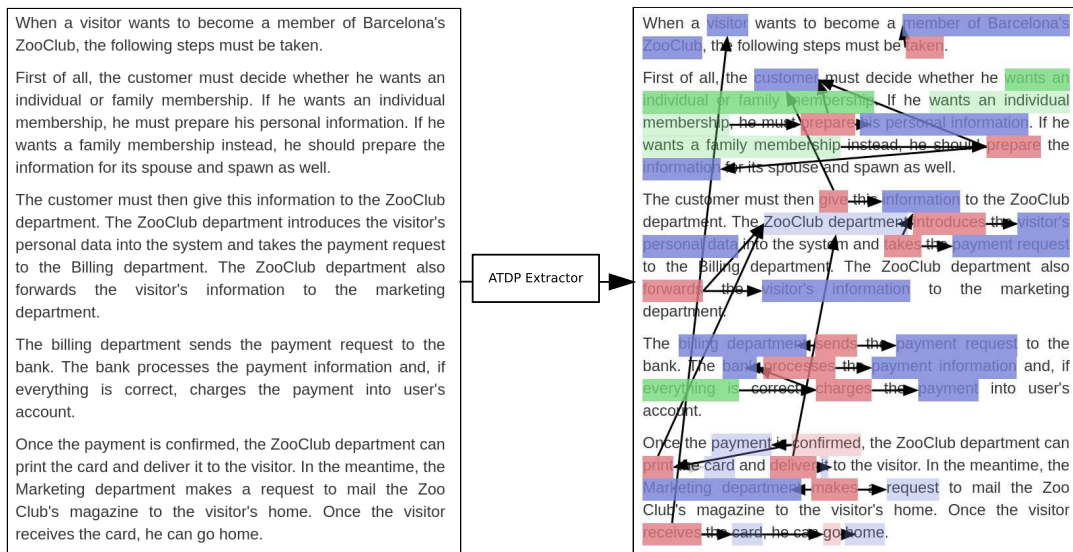


Fig. 10 Example of automatic ATDP extraction used in real application.

ronment for reasoning. The tool is available at <https://github.com/PADS-UPC/atdp2nusmv>.

- The ATDP simulator, which enables simulating ATDP specifications for deriving an event log in XES format corresponding to the simulated traces. The simulator is available at <https://github.com/PADS-UPC/atdp-simulator>.
- The ATDP editor, which enables to edit manually an ATDP specification for some of the relations of this paper. The tool is available at <https://github.com/PADS-UPC/atd-editor>.

In the remainder of this section, we focus in the three ones presented in this paper: the ATDP extractor, the ATDP reasoner and the ATDP simulator.

7.1 ATDP Extractor: a Tool to Annotate Textual Descriptions

Figure 10 is an example of what a text looks like before and after using the ATDP extractor. On the left, a textual description of a process is shown, and its automatic annotation is reported on the right part of the figure.

Use Case: The Model Judge

A real application currently using the ATDP extractor is the *Model Judge* [15,42], available at <https://modeljudge.cs.upc.edu>.

Model Judge is an educational platform for learning and teaching process models in the Business Process Model and Notation (BPMN) graphical language.

Instructors of the Model Judge can select exercises for their students to practice modeling. An exercise consists of a textual description of a process that the student needs to model in BPMN. At anytime, a student can validate its solution, which is compared against the gold standard ATDP solution provided by the platform.

In order to make the platform extensible, instructors can propose new exercises. For this, they can use an exercise editor that is provided with the platform, so that

Fig. 11 Automatic annotation of an exercise in the Model Judge platform.

some of the annotations of this paper can be manually inserted by the instructor in a plain textual description of a process. To alleviate the annotation effort, the ATDP extractor is enabled in the exercise editor to propose a first set of annotations automatically. Figure 11 shows an example of automatic annotation obtained in the exercise editor.

7.2 ATDP Reasoner: Model Checking ATDP Specifications

The encoding technique described in Section 6.2.1 has been implemented in a prototype tool, ATDP2NuSMV. The tool can be used to convert an ATDP specification into a NuSMV instance. ATDP2NuSMV is distributed as a standalone tool, that can be used in any system with a modern Java installation, and without further dependencies.

In the remainder of this subsection, we present use cases that have been tested with ATDP2NuSMV and NuSMV. The ATDP specifications as well as the exact query encodings can be found in the repository. The use case examples are based on a full version of the specification presented in Figure 3.

Use Case 1: Model Consistency.

An ATDP specification can be checked for consistency using proof by contradiction. Specifically, if we set $Q = \perp$, the reasoner will try to prove that $A \wedge \mathcal{C}_F \wedge \mathcal{C}_{T_F} \rightarrow \perp$, that is, whether a false conclusion can be derived from the axioms and constraints

describing our model. Since this implication only holds in the case $\perp \rightarrow \perp$, if the proof succeeds we will have proven that $A \wedge \mathcal{C}_F \wedge \mathcal{C}_{T_F} \equiv \perp$, i.e. that our model is not consistent. On the contrary, if the proof fails we can be sure that our model does not contain any contradiction.

To illustrate this use case, we use interpretations `hosp-1` and `hosp-1-bad`, available in our repository. The first interpretation consists of a complete version of the specification in Figure 3, where F .activities includes $a_1 = \text{takes (the sample)}$ and $a_2 = \text{validates (sample state)}$, and constraints in \mathcal{C}_F include: `Mandatory(a_1)`, `Precedence($\{a_1\}, a_2$)` and `Response($a_1, \{a_2\}$)`. NuSMV falsifies the query in interpretation `hosp-1` with a counter-example. When the model is consistent, the property is false, and the resulting counter example can be any valid trace in the model.

The second specification, `hosp-1-bad` adds `Precedence($\{a_2\}, a_1$)` to the set of relations R . This relation contradicts the previously existing `Precedence($\{a_1\}, a_2$)`, thus resulting in an inconsistent model. Consequently, NuSMV cannot find a counter-example for the query in interpretation `hosp-1-bad`. This result can be interpreted as the model being impossible to fulfill by any possible trace, and thus inconsistent.

Use Case 2: Compliance Checking.

Business rules, as those arising from regulations or SLAs, impose further restrictions that any process model may need to satisfy. On this regard, compliance checking methods assess the adherence of a process specification to a particular set of predefined rules.

The presented reasoning framework can be used to perform compliance checking on ATDP specifications. An example rule for our running example might be: “An invalid sample can never be used for diagnosis”. The relevant activities for this property are annotated in the text: $a_3 = \text{(the sample) can be used}$, $a_4 = \text{(the sample) is contaminated}$, $a_5 = \text{makes the diagnosis}$, and the property can be written in LTL as: $Q = \Box(a_4 \rightarrow (\neg a_5 U a_3))$.

In the examples from our repository, interpretations `hosp-2-i`, with $i=\{1,2,3\}$, correspond to the three interpretations of the process shown in Figure 3. Particularly, the ambiguity between the three interpretations is the scope of the repetition when the taken sample is contaminated. The three returning points correspond to: `sign an informed consent`, `sampling is prepared` and `take the sample`. NuSMV finds the property true for all three interpretations, meaning that we can prove the property $\Box(a_4 \rightarrow (\neg a_5 U a_3))$ without resolving the main ambiguity in the text.

Use Case 3: Conformance Checking.

Conformance checking techniques put process specifications next to event data, to detect and visualize deviations between modeled and observed behavior [6]. On its core, conformance checking relies on the ability to find out whereas an observed trace can be reproduced by a process model.

A decisional version of conformance checking can be performed, by encoding traces inside Q as an LTL formulation. Given a trace $t = \langle a_1, a_2, \dots, a_N \rangle$, we can test conformance against an ATDP interpretation with the following query⁶:

⁶ The proposed query does not account for the start and end activities of scopes, which are not present in the original trace. A slightly more complex version can be crafted that accounts

$$Q = \neg(a_1 \wedge \circ(a_2 \wedge \circ(\dots \wedge \circ(a_N \wedge \circ\text{STOP}))))$$

This query encodes the proposition “Trace t is not possible in this model”. This proposition will be false whenever the trace is accepted by the model. Other variants of this formulation allow for testing trace patterns: partial traces or projections of a trace to a set of activities. In this case, the counter-example produced will be a complete trace which fits the model and the queried pattern.

As an example of this use-case, we provide the example ATDP interpretation `hosp-3` in our repository. We project the set of relevant activities to the set $a_6 = \text{informs (the patient)}$, $a_7 = \text{signs (informed consent)}$ $a_8 = \text{arranges (an appointment)}$. Two trace patterns are tested, the first: $t_1 = \langle \dots a_6, a_7, a_8, \dots \rangle$ and $t_2 = \langle \dots a_7, a_6, a_8, \dots \rangle$. NuSMV finds the trace pattern t_1 fitting the model, and produces a full execution trace containing it. On the other hand, t_2 does not fit the model, which is successfully proven by NuSMV.

7.3 ATDP Simulator: Extracting Event Logs from ATDP Simulations

The ATDP Simulator is an implementation of the algorithm described in Section 6.3. The implementation uses the `atdplib-model` library as the reference Java implementation to parse and handle the ATDP file format. The ATDP Simulator itself, is available as a separate tool and is implemented in Clojure, a functional language for the Java Virtual Machine.

The ATDP Simulator is implemented as a command line tool which takes an ATDP specification with a single interpretation, and will produce a log file in the XES[55] file format with the requested number of traces.

We now use the ATDP specification of an academic example text describing the process of a client obtaining a membership at Barcelona’s Zoo. Figure 12 shows the complete ATDP specification for this example. In the visual representation, some high-level abbreviations for temporal constraints not described in [40] are used: Alternatives (in the figure, `ALTS.`), used to describe multiple branches of a decision and Succession (in the figure, `SUCC.`), which combines the Precedence and Response relations. These relations use the base relations from [40] as building blocks and can thus be considered syntactic sugar. In this specification, only the temporal aspects of the process is shown for illustrative purposes.

We used the ATDP Simulator to produce example traces for the process in Figure 12. Table 2 shows some example traces. In order to validate the resulting traces, we used the well-known *Inductive Miner* [25] to discover a process model. As seen in Figure 13, the resulting BPMN is consistent with the ATDP specification.

In order to validate the simulation, we have done another experiment: we have used three realistic examples to test the simulation method. The experiment consists in three pairs of process representations, where each pair is a process model as a Petri net, and the corresponding faithful textual description as an ATDP specification. Then we have simulated the textual description and tested the conformance of the generated event log with respect to the process model. In particular, we

for any invisible activity to be present between the visible activities of the trace. We do not show it here for the sake of simplicity.

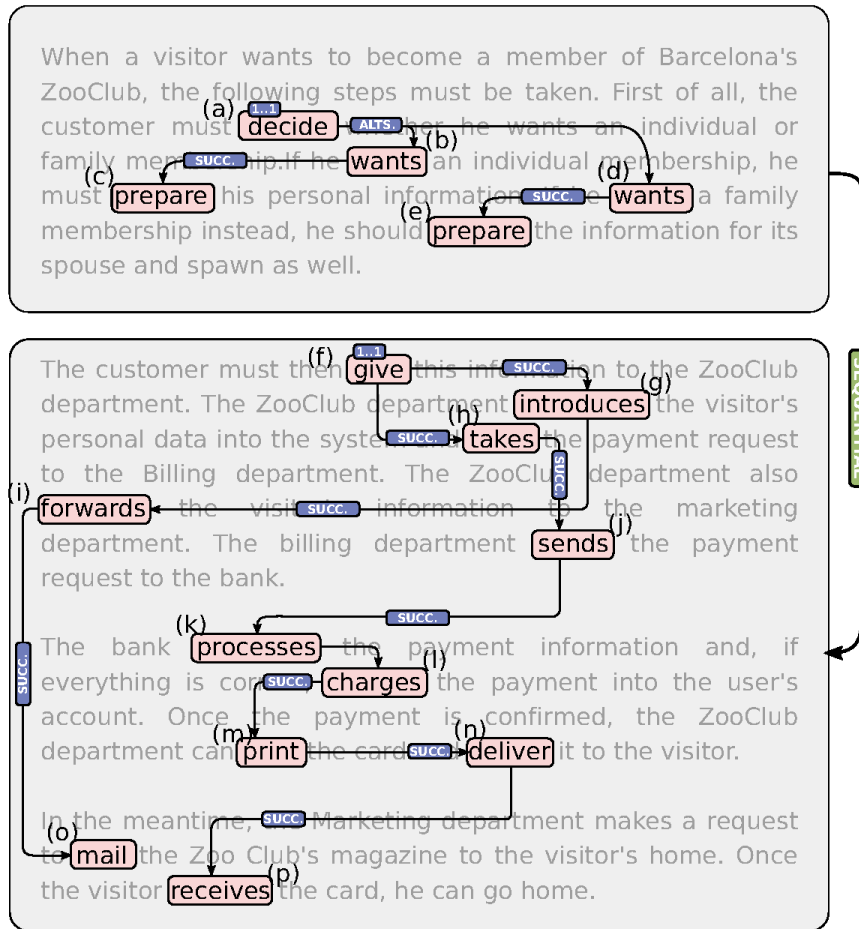


Fig. 12 Control-flow projection of scope the ATDP specification of the Zoo process.

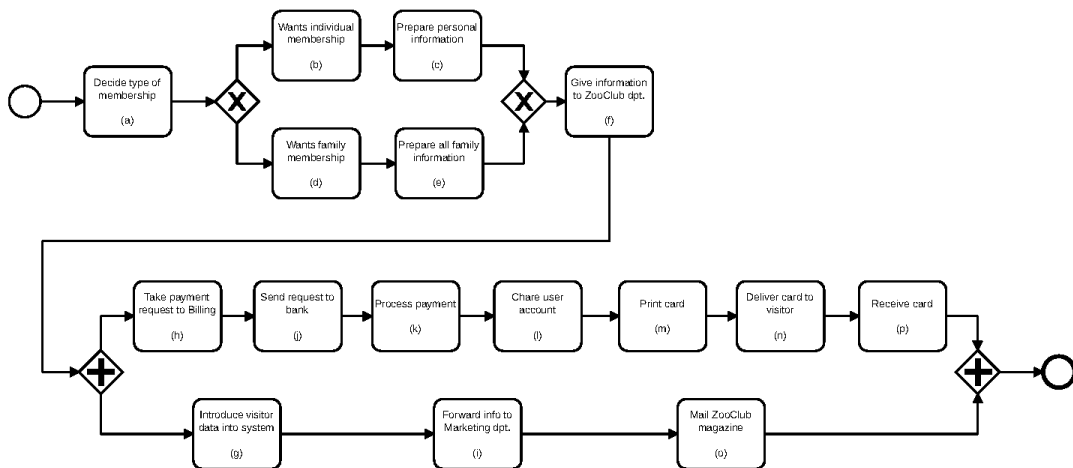


Fig. 13 BPMN Model obtained by mining the log from Table 2 using the Inductive Miner.

Trace	Frequency
$\langle a, b, c, f, g, i, o, h, j, k, l, m, n, p \rangle$	34
$\langle a, d, e, f, g, i, o, h, j, k, l, m, n, p \rangle$	27
$\langle a, d, e, f, g, h, i, o, j, k, l, m, n, p \rangle$	22
$\langle a, b, c, f, g, h, i, o, j, k, l, m, n, p \rangle$	22
...	...

Table 2 Example log with several traces obtained by simulating the ATDP specification of Figure 12.

have analyzed the *fitness* of each pair, i.e., the ability of the model in reproducing the traces of the log generated, a real number between 0 and 1 [6]. A high fitness indicates that the model and the log agree on the main behavior described in both process representations. For analyzing fitness, we have used the ProM platform [54].

From the results on Table 3 one can see that the results match the expectations one may have when relating structured information (i.e., a formal graphical notation like a Petri net), and a less structured information like an ATDP specification where some of the relations have a declarative form and the main actions and relations highlighted in each may differ. Still, in spite of this representational difference, the simulation obtains an object that keeps in average a significant portion of the behavior as described in the original model.

Aside from validation of the generated event logs, we believe enabling synthetic event log generation for ATDP specifications can open new applications for this paradigm. Allowing use of well-established techniques from the field of process mining on top of semi-structured representations. A more thorough exploration of this topic has been considered for future work.

7.4 Discussion

Although the previous use cases provide applications of the paradigm introduced in this paper, further investigation needs to be carried out to relate the design decisions made with the actual deployment of the ideas as reported in this section. On the automatic extraction of ATDP from text, for instance, we have realized that the notion of scopes is a real challenge, since it requires a deep analysis of the textual description. Another challenge arises when operationalizing the reasoning of ATDP specifications, where the proposed translation to LTL_f (as described in Section 6.2.1) needs to be adapted for enabling current model checking technology to be applied, due to the state-explosion problem. Finally, we have realized that when simulating an ATDP specification there may be some hidden biases arising from its structure.

8 Conclusions and Future Work

This paper presents a first attempt to automatically bridge the existing gap between unstructured process information and its operational use in organizations.

Process	Fitness
Reimbursement Process	0.69
Dispatch Process	0.64
Zoo Process	0.96
Average	0.76

Table 3 Conformance checking for simulated ATDP specifications and the corresponding gold models.

The paper proposes annotated textual descriptions of process as the right balance between formalization and accessibility, and contributes in all the necessary steps to make the operationalization of unstructured data that talks about processes possible.

We formalize ATDP specifications, and express its semantics in temporal logic, thus opening the door to formal reasoning. We show how this reasoning can be done by casting questions as model checking queries; several examples are provided that witness the reasoning capabilities of our approach.

We also show that current NLP technology can assist into the automatic extraction of ATDP elements. In particular, we propose methods for extracting automatically fragments of ATDP specifications, using NLP analysis and tree queries over the dependency trees corresponding to sentences describing a process. We show how this method is currently used in an educational platform for process modeling.

Finally, we connect ATDP specifications with process mining [53], by proposing a simulation approach that enables generating an event log that encompasses the main behavior of the process. Techniques like *process discovery* or *conformance checking* are then possible, to discover formal process models or to analyse the conformance against real-life event data, respectively.

Several future research avenues are ahead of us, so we report on the most promising ones. First, extending the extraction of ATDP specifications to capture all the possible annotations is an investigation that we are currently tackling. Second, to extend the formal connection between reasoning and model checking, and to propose alternative encodings so that it can be applied on larger specifications, will be an important matter to consider. Finally, to validate the usability of the ATDP specifications with real users would let us to have a better understanding of the accessibility of the language proposed.

Acknowledgments

This work has been supported by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R.

References

1. Decision Model and Notation (DMN). Standard, OMG - Object Management Group, 2016.

2. Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. Automated checking of conformance to requirements templates using natural language processing. *IEEE Transactions on Software Engineering*, 41(10):944–968, October 2015.
3. Martin Bergner, Hans-Georg Fill, and Florian Johannsen. Supporting business process improvement with natural language processing: a model-based approach. 2016.
4. Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. A genetic algorithm for discovering process trees. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2012.
5. Andrea Burattin. PLG2: multiperspective process randomization with online and offline simulations. In *BPM Demo Track*, 2016.
6. Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich. *Conformance Checking - Relating Processes and Models*. Springer, 2018.
7. Yubo Chen, Liheng Xu, Kang Liu, Daojian Zeng, and Jun Zhao. Event extraction via dynamic multi-pooling convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 167–176, Beijing, China, July 2015. Association for Computational Linguistics.
8. Claudio Di Ciccio, Mario Luca Bernardi, Marta Cimitile, and Fabrizio Maria Maggi. Generating event logs through the simulation of declare models. In *EOMAS 2015*, pages 20–36, 2015.
9. Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A new symbolic model checker. *STTT*, 2(4):410–425, 2000.
10. Jan Claes, Irene Vanderfeesten, J. Pinggera, H.A. Reijers, B. Weber, and G. Poels. A visual analysis of the process of process modeling. *Information Systems and e-Business Management*, 13(1):147–190, 2015.
11. Rajarshi Das, Tsendsuren Munkhdalai, Xingdi Yuan, Adam Trischler, and Andrew McCallum. Building dynamic knowledge graphs from text using machine reading comprehension. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.
12. Giuseppe De Giacomo, Riccardo De Masellis, and Marco Montali. Reasoning on LTL on finite traces: Insensitivity to infiniteness. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 1027–1033. AAAI Press, 2014.
13. Ana Karla Alves de Medeiros and Christian W. Günther. Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms. In *Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 177–190, 2005.
14. Luis Delicado, Josep Sanchez-Ferrerres, Josep Carmona, and Lluís Padró. NLP4BPM - Natural Language Processing Tools for Business Process Management. In *BPM Demo Track*, 2017.
15. Luis Delicado, Josep Sanchez-Ferrerres, Josep Carmona, and Lluís Pardo. The Model Judge - A Tool for Supporting Novices in Learning Process Modeling. In *BPM Demo Track*, 2018.
16. Remco Dijkman, Irene Vanderfeesten, and Hajo A. Reijers. Business process architectures: overview, comparison and framework. *Enterprise Information Systems*, 10(2):129–158, feb 2016.
17. Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer, 2018.
18. Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *Proceedings of ICSE*, pages 411–420. ACM, 1999.
19. Christiane Fellbaum. *WordNet. An Electronic Lexical Database*. Language, Speech, and Communication. The MIT Press, 1998.
20. Wenfeng Feng, Hankz Hankui Zhuo, and Subbarao Kambhampati. Extracting action sequences from texts based on deep reinforcement learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4064–4070. International Joint Conferences on Artificial Intelligence Organization, 7 2018.
21. Fabian Friedrich, Jan Mendling, and Frank Puhmann. Process model generation from natural language text. In Haralambos Mouratidis and Colette Rolland, editors, *Advanced Information Systems Engineering*, pages 482–496, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
22. Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, 2013.

23. A. Halioui, P. Valtchev, and A. B. Diallo. Bioinformatic workflow extraction from scientific texts based on word sense disambiguation. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 15(6):1979–1990, 2018.
24. Toon Jouck. *Empirically Evaluating Process Mining Algorithms: Towards Closing the Methodological Gap*. PhD thesis, University of Hasselt, 2018.
25. Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering block-structured process models from event logs - A constructive approach. In *Proceedings of PETRI NETS*, pages 311–329. Springer, 2013.
26. Henrik Leopold. *Natural language in business process models*. PhD thesis, Springer, 2013.
27. Henrik Leopold, Han van der Aa, and Hajo A. Reijers. Identifying candidate tasks for robotic process automation in textual process descriptions. In Jens Gulden, Iris Reinhartz-Berger, Rainer Schmidt, Sérgio Guerreiro, Wided Guédria, and Palash Bera, editors, *Enterprise, Business-Process and Information Systems Modeling - 19th International Conference, BPMDS 2018, 23rd International Conference, EMMASAD 2018, Held at CAiSE 2018, Tallinn, Estonia, June 11-12, 2018, Proceedings*, volume 318 of *Lecture Notes in Business Information Processing*, pages 67–81. Springer, 2018.
28. Roger Levy and Galen Andrew. Tregex and tsurgeon: tools for querying and manipulating tree data structures. In *LREC*, pages 2231–2234. Citeseer, 2006.
29. Bilal Maqbool, Farooque Azam, Muhammad Waseem Anwar, Wasi Haider Butt, Jahan Zeb, Iqra Zafar, Aiman Khan Nazir, and Zuneera Umair. A Comprehensive Investigation of BPMN Models Generation from Textual Requirements - Techniques, Tools and Trends. In *Information Science and Applications 2018 - ICISA 2018, Hong Kong, China, June 25-27th, 2018*, volume 514 of *Lecture Notes in Electrical Engineering*, pages 543–557. Springer, 2019.
30. Jan Mendling, Bart Baesens, Abraham Bernstein, and Michael Fellmann. Challenges of smart business process management: An introduction to the special issue. *DSS*, 100:1–5, 2017.
31. Jan Mendling, Henrik Leopold, and Fabian Pittke. 25 challenges of semantic process modeling. *International Journal of Information Systems and Software Engineering for Big Companies*, 1(1):78–94, 2015.
32. Vanessa Tavares Nunes, Flávia Maria Santoro, and Marcos R.S. Borges. A context-based model for knowledge management embodied in work processes. *Information Sciences*, 179(15):2538 – 2554, 2009.
33. Avner Ottenssooser, Alan Fekete, Hajo A. Reijers, Jan Mendling, and Con Menictas. Making sense of business process descriptions: An experimental comparison of graphical and textual notations. *Journal of Systems and Software*, 85(3):596–606, 2012.
34. Lluís Padró and Evgeny Stanilovsky. Freeing 3.0: Towards wider multilinguality. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC)*, pages 2473–2479, 2012.
35. M. Pesic, H. Schonenberg, and W. M. P. van der Aalst. DECLARE: Full support for loosely-structured processes. In *Proc. of the IEEE Int. Enterprise Distributed Object Computing Conference*, pages 287–298. IEEE Computer Society, 2007.
36. J. Pinggera. *The Process of Process Modeling*. PhD thesis, University of Innsbruck, Department of Computer Science, 2014.
37. Amir Pnueli. The temporal logic of programs. pages 46–57. IEEE, 1977.
38. Chen Qian, Lijie Wen, Akhil Kumar, Leilei Lin, Li Lin, Zan Zong, Shu’ang Li, and Jianmin Wang. An approach for process model extraction by multi-grained text classification. In Schahram Dustdar, Eric Yu, Camille Salinesi, Dominique Rieu, and Vik Pant, editors, *Advanced Information Systems Engineering*, pages 268–282, Cham, 2020. Springer International Publishing.
39. Kristin Y. Rozier and Moshe Y. Vardi. LTL satisfiability checking.
40. Josep Sànchez-Ferreres, Andrea Burattin, Josep Carmona, Marco Montali, and Lluís Padró. Formal reasoning on natural language descriptions of processes. In *Proceedings of BPM*, pages 86–101, 2019.
41. Josep Sànchez-Ferreres, Josep Carmona, and Lluís Padró. Aligning textual and graphical descriptions of processes through ILP techniques. In *Proceedings of CAiSE*, pages 413–427, 2017.
42. Josep Sànchez-Ferreres, Luis Delicado, Amine Abbad Andaloussi, Andrea Burattin, Guillermo Calderón-Ruiz, Barbara Weber, Josep Carmona, and Lluís Padró. Supporting the process of learning and teaching process models. *IEEE Trans. Learn. Technol.*, 13(3):552–566, 2020.

43. Josep Sànchez-Ferreres, Han van der Aa, Josep Carmona, and Lluís Padró. Aligning textual and model-based process descriptions. *Data & Knowledge Engineering*, 118:25–40, 2018.
44. Pol Schumacher. *Workflow Extraction from Textual Process Descriptions*. PhD thesis, Johann Wolfgang Goethe Universität Frankfurt/Main, 2016.
45. F Semmelrodt. Modellierung klinischer prozesse und compliance regeln mittels BPMN 2.0 und eCRG. Master’s thesis, University of Ulm, 2013.
46. Andrea Di Sorbo, Sebastiano Panichella, Corrado Aaron Visaggio, Massimiliano Di Penta, Gerardo Canfora, and Harald C. Gall. Exploiting natural language structures in software informal documentation. *IEEE Transactions on Software Engineering*, 2019.
47. Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun’ichi Tsujii. Brat: a web-based tool for nlp-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 102–107. Association for Computational Linguistics, 2012.
48. Niket Tandon, Bhavana Dalvi, Joel Grus, Wen-tau Yih, Antoine Bosselut, and Peter Clark. Reasoning about actions and state changes by injecting commonsense knowledge. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun’ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 57–66. Association for Computational Linguistics, 2018.
49. 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 *Proceedings of the 27th International Conference on Computational Linguistics (COLING)*, pages 2791–2801, 2018.
50. Han van der Aa, Henrik Leopold, and Hajo A. Reijers. Detecting inconsistencies between process models and textual descriptions. In Hamid Reza Motahari-Nezhad, Jan Recker, and Matthias Weidlich, editors, *Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings*, volume 9253 of *Lecture Notes in Computer Science*, pages 90–105. Springer, 2015.
51. Han van der Aa, Henrik Leopold, and Hajo A. Reijers. Dealing with behavioral ambiguity in textual process descriptions. In Marcello La Rosa, Peter Loos, and Oscar Pastor, editors, *Business Process Management - 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings*, volume 9850 of *Lecture Notes in Computer Science*, pages 271–288. Springer, 2016.
52. Han van der Aa, Henrik Leopold, and Hajo A. Reijers. Comparing textual descriptions to process models - the automatic detection of inconsistencies. *Inf. Syst.*, 64:447–460, 2017.
53. Wil M.P. van der Aalst. *Process Mining*. Springer, 2016.
54. Eric Verbeek, Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Prom 6: The process mining toolkit. In *Proceedings of the Business Process Management 2010 Demonstration Track, Hoboken, NJ, USA, September 14-16, 2010*, 2010.
55. H. M. W. Verbeek, Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. XES, XESame, and ProM 6. In *Information Systems Evolution*, pages 60–75, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
56. Kirstin Walter, Mirjam Minor, and Ralph Bergmann. Workflow extraction from cooking recipes. In *Process-Oriented Case-Based Reasoning Workshop*, pages 207–216, 01 2011.
57. Bishan Yang and Tom M. Mitchell. Joint extraction of events and entities within a document context. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 289–299, San Diego, California, June 2016. Association for Computational Linguistics.

Appendix VII

The Model Judge – A Tool for Supporting Novices in Learning Process Modeling

- **Publication Type:** Demo paper
- **Published In:** International Conference in Business Process Management, Demo Session (BPM 2018)
- **Full Reference:** Luis Delicado, Josep Sànchez-Ferreres, Josep Carmona and Lluís Padró. "The Model Judge - A Tool for Supporting Novices in Learning Process Modeling". In: *BPM 2018 Demonstration Track*. 2018.

The Model Judge – A Tool for Supporting Novices in Learning Process Modeling

Luis Delicado^[0000–0002–8866–9440], Josep Sànchez-Ferreres^[0000–0002–7204–4307],
Josep Carmona^[0000–0001–9656–254X], and Lluís Padró^[0000–0003–4738–5019]

Computer Science Department
Universitat Politècnica de Catalunya
Barcelona, Spain

{ldelicado, jsanchezf, jcarmona, padro}@cs.upc.edu

Abstract. Process models are a fundamental element in the BPM lifecycle. Hence, it is of paramount importance for organizations to rely on high-quality, accurate and up-to-date process models, to avoid taking decisions on the basis of a wrong picture of the reality. In this demo we present `modeljudge.cs.upc.edu`, a platform to boost the training of novice modelers when confronted with the task of translating a textual description into a process model in BPMN notation. The platform is integrated with Natural Language Processing (NLP) analysis and textual annotation, together with a novel model-to-text alignment technique. By using this platform, a novice modeler will receive diagnostics in real-time, which may contribute to a more satisfactory modeling experience.

Keywords: Process Modeling · Natural Language Processing · Education

1 Introduction

Due to the wide usage of process models in organizations, correctness and quality of models have a direct influence in the execution of business processes. However, research has shown that industrial process models often contain errors, which can lead to many problems, like increased costs in production.

Automating the detection of syntactic errors is a common feature in modeling 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. 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 algorithm for the computation of diagnostics is based on the technique for automatic computation of alignments between process model and textual descriptions presented in [6].

Significance of the tool for the BPM field. As it is pointed out in [1], process models play a central role in the management of processes within organizations. Although recent automated techniques can help into the discovery of a process model [8], the process of process modeling it is still a crucial element in the BPM lifecycle [5]. Frameworks integrating different modeling notations, like the one presented in this paper for

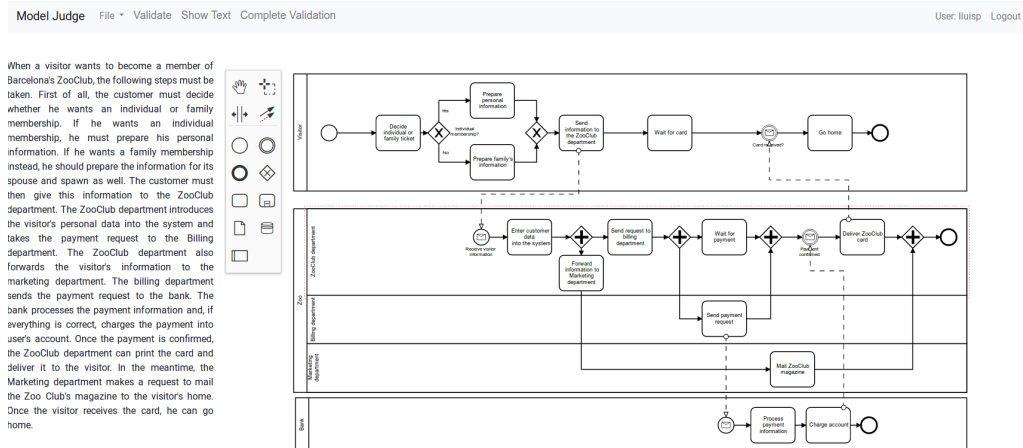


Fig. 1. Screenshot of the modeljudge workspace: textual description and model editor.

textual descriptions and BPMN, will help into narrowing the gap between processes and their representations within organizations.

2 Tool Description and Features

The Model Judge is presented as a web-based platform, that can be accessed through any 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.

Working with the Model Judge is very much like creating a process model using any BPMN editor. What makes the platform special is the underlying engine to provide automatic feedback (see below). When modeling, a textual description of the process model is available on the left part, while the modeling editor is shown in the right part. Fig. 1 shows the workspace.

With the goal of providing an accurate evaluation of students' models, and inspired by the success of judges to support learning to program ([2,4] among many others), 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 model well-formedness and control flow. *Pragmatic* diagnostics verify the phrasing of the process model labels and enforce certain grammatical rules. Finally, *semantic* diagnostics check for coverage (there is no missing information from the underlying process) and for relevance (no irrelevant information is included in the model). Fig. 2 provides different types of diagnostics reported for a particular example.

Two different types of feedback can be provided to the student, depending on the granularity of the information required:

- **Validation:** returns an aggregated diagnostic that reflects if the model has some errors of the types explained before, but it does not say what or where is exactly

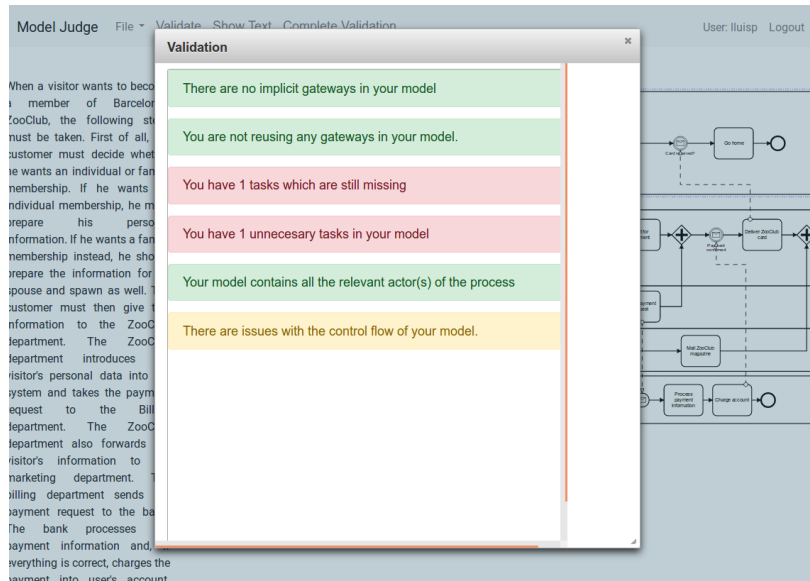


Fig. 2. Screenshot of the feedback for a model where a relevant task *introduce user data into the system* has been replaced by a non-relevant task *feed the dog*.











the problem. The motivation for this check is to allow for a mild test to guide the students without giving away the whole solution. The feedback provided in Fig. 2 is a validation.

- **Complete Validation:** apart from the overall information provided by the Validation, this check also provides a detailed list of all the problems detected, and individually explained. The motivation for this check is to allow an assessment similar to the one obtained if a teacher was correcting the model, and would be typically provided once the student finishes and hands over the exercise.

In order to use the platform, a user (either student or instructor) needs an account, with its associated storage space. This space acts as a cloud drive for models, allowing to manually save multiple versions of an exercise. Moreover, the students enrolled in a particular course, may enable the platform to record a history of their modeling session. This can be used for analyzing their behavior, which can be reported back to the instructor to get a clearer picture of the modeling process of their students.

Currently, there are 9 different exercises available in the platform. Any modeler can register into the platform and practice with them. Instructors can also design a course, by selecting the exercises that must be included. Fig. 3 shows the main page for defining a course as instructor. Once a course is created, a unique code will be created that can be shared with the students of this course.

Support for adding new exercises is restricted to the developers of the platform. However, support is planned to allow instructors to create their own exercises. In order to create a new exercise for the Model Judge, a textual description of a process is required. This textual description will be used as the problem statement for the students so they can understand the process to be modeled. Instructors will then have to anno-

Teachers Zone			Name Course
Course	Code	Historial/ Final Validation	
Prueba	X776l	 	<input type="text"/>
GPN-UCSM-01-Hospital	X784g	 	<input type="text"/>
GPN-UCSM-02-Hospital	X868g	 	<input type="text"/>
GPN-UCSM-01-Zoo	X970g	 	<input type="text"/>
GPN-UCSM-02-Zoo	X1277g	 	<input type="text"/>

Exercices

Fig. 3. Screenshot of the interface for the definition new courses by an instructor.

tate the relevant parts of the process: *Actions*, *Entities* and *Conditions* as well as their relations: *Agent*, *Patient*. This annotation process is partially performed by a Natural Language Processing algorithm, which provides an initial annotation to be refined. Figure 4 shows a fragment of a text annotation corresponding to one of the exercises in the platform.

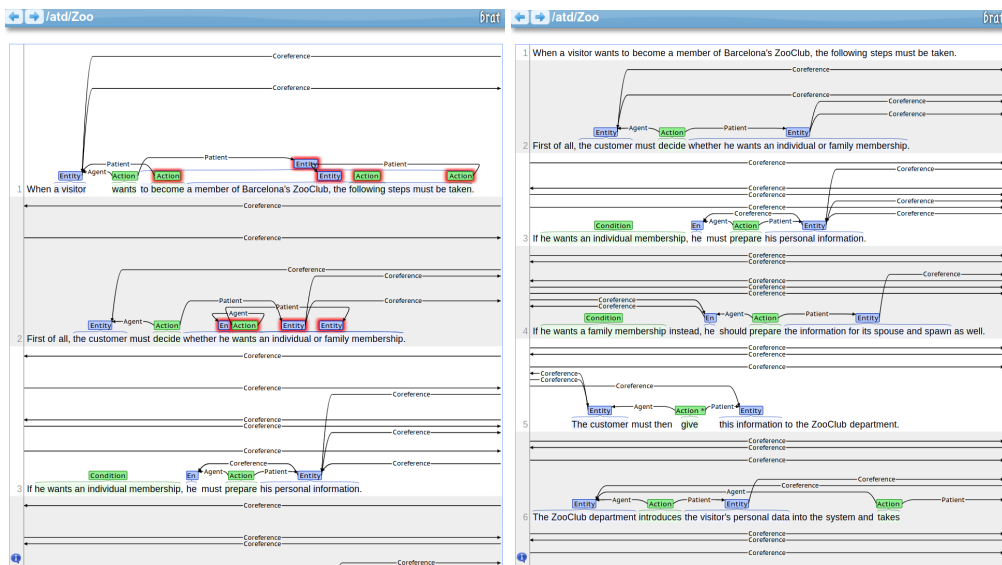


Fig. 4. The annotation interface for the definition of exercises: (left) An automatically generated annotation. (right) After the instructor performs a manual validation.

A screencast of the Model Judge which shows a typical session working with the model judge can be found in <https://youtu.be/xJ3TeKlvIf0>.

3 Architecture, Libraries Used and Maturity of the Tool

Model Judge is built as a web application. A distributed server manages several instances of the application and balances the load between them. The front-end of it is built using the PrimeFaces Java framework, which internally communicates with the core Java application, responsible for the generation of diagnostics. Several functionalities in Model Judge rely on external libraries: *FreeLing* [3] is used for all the Natural Language Processing tasks and the *Gurobi* ILP solver is used to compute optimal alignments, as described in [6]. Additionally, the BRAT [7] text annotation software is used as an external tool in order to create new exercises for the platform.

The Model Judge has been tested in two separate modeling courses. The first was performed on the Technical University of Denmark (DTU) during February 2018. The second course was performed at the Catholic University of Santa María (UCSM) in Peru during March 2018. For every student of these courses, we stored periodically (every minute) information for the whole modeling session. Additionally, 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.

References

1. Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013.
2. Adrian Kosowski, Michal Malafiejski, and Tomasz Noinski. Application of an online judge & contester system in academic tuition. In *Advances in Web Based Learning - ICWL 2007, 6th International Conference, Edinburgh, UK, August 15-17, 2007, Revised Papers*, pages 343–354, 2007.
3. Lluís Padró and Evgeny Stanilovsky. Freeling 3.0: Towards wider multilinguality. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC*, pages 2473–2479, Istanbul, Turkey, May 2012.
4. Jordi Petit, Omer Giménez, and Salvador Roura. Judge.org: An educational programming judge. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education, SIGCSE '12*, pages 445–450, New York, NY, USA, 2012. ACM.
5. Jakob Pinggera, Pnina Soffer, Dirk Fahland, Matthias Weidlich, Stefan Zugal, Barbara Weber, Hajo A. Reijers, and Jan Mendling. Styles in business process modeling: an exploration and a model. *Software and System Modeling*, 14(3):1055–1080, 2015.
6. Josep Sànchez-Ferreres, Josep Carmona, and Lluís Padró. Aligning textual and graphical descriptions of processes through ILP techniques. In *Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings*, pages 413–427, 2017.
7. Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. Brat: a web-based tool for nlp-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 102–107. Association for Computational Linguistics, 2012.
8. Wil M. P. van der Aalst. *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.