**Software and Systems Modeling manuscript No.**
(will be inserted by the editor)

# Unleashing Textual Descriptions of Business Processes

**Josep Sànchez-Ferreres · Andrea Burattin ·
Josep Carmona · Marco Montali ·
Lluís Padró · Luís Quishpi**

the date of receipt and acceptance should be inserted later

**Abstract** Textual descriptions of processes are ubiquous 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 [16]. 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

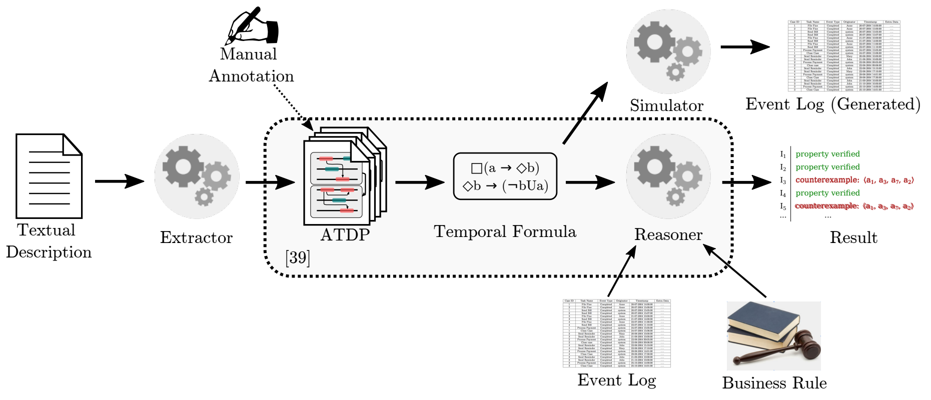


**Fig. 1** General framework for extraction, analysis and simulation of textual descriptions of processes. The contributions introduced in [40] are highlighted.

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, 11], 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 toolchain 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.

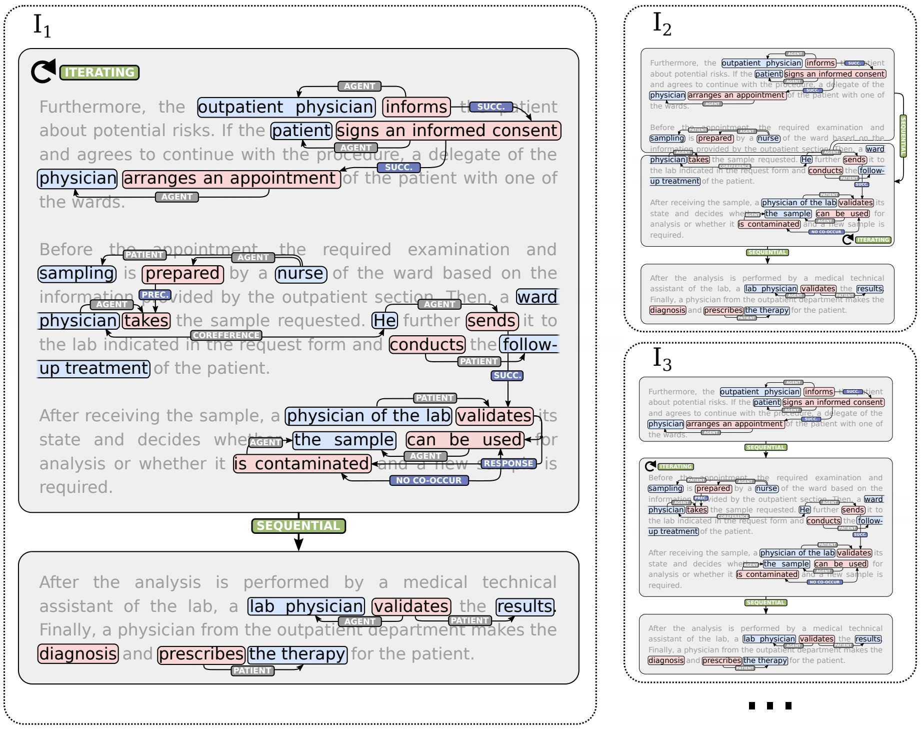
## 2 ATDP in a Nutshell

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.



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. 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 [3].

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., [16,15]), 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" [36,9]. 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) [21,44,13]. 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 [19,38]. 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 [6,57,48,27,10,20]. 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, 14]. 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 [12]. 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 [4], 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 [3] 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, [7] 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 [22,11]. 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 $\mathrm{LTL}_f$, 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, $\mathrm{LTL}_f$ 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 $\varphi$ *holds*.
- $\varphi_1\, U\, \varphi_2$ says that at some future instant $\varphi_2$ will hold and *until* that point $\varphi_1$ always holds.

Common abbreviations used in LTL and LTL$_f$ include the ones listed below:
- Standard boolean abbreviations, such as $\top$, $\bot$, $\vee$, $\rightarrow$.
- $\Diamond\varphi = \top\, U\, \varphi$ says that $\varphi$ will *eventually* hold at some future instant.
- $\Box\varphi = \neg\Diamond\neg\varphi$ says that from the current instant $\varphi$ will *always* hold (until the last instance of the trace).
- $\varphi_1\, W\, \varphi_2 = (\varphi_1\, U\, \varphi_2 \vee \Box\varphi_1)$ is interpreted as a *weak until*, and means that either $\varphi_1$ holds until $\varphi_2$ 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 [11]. 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 [11].

### 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 FreeLing[1] [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 [18] as the sense catalogue and synset codes as concept identifiers.

Constituency/dependency parsing: Given a sentence, get its syntatic 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)* $F.\mathsf{activities}$ for activities; *(ii)* $F.\mathsf{roles}$ for roles; *(iii)* $F.\mathsf{objects}$ 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

$$agent_F : F.\mathsf{activities} \to F.\mathsf{roles}$$

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

*Patient.* A *patient relation*[2] over $F$ is a partial function

$$patient_F : F.\mathsf{activities} \to F.\mathsf{roles} \cup F.\mathsf{objects}$$

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

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

$$coref_F \subseteq F.\mathsf{roles} \times F.\mathsf{roles} \cup F.\mathsf{objects} \times F.\mathsf{objects}$$

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 [3]. 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 pair[3] 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* ($\circlearrowright$) 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 $\mathit{parent} : F.\mathsf{activities} \rightarrow S_l$ 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 $\mathit{btype} : S_b \rightarrow \{\rightarrow, \times, \vee, \wedge, \circlearrowright\}$ 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 [17], where, differently from [35], constraints are in fact scoped.[4]

We consider in particular the following constraints:

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

*Scoped Response* Given activities $a, b_1, \ldots, b_n$, $\mathsf{Response}(a, \{b_1, \ldots, b_n\})$ indicates that whenever $a$ is executed within an instance of its parent scope, then at least one among $b_1, \ldots, b_n$ 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 [17].

*Scoped Weak Order* Given two activities $a, b$, $\mathsf{WeakOrder}(a, b)$ 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 $a, b$, $\mathsf{NonCoOccurrence}(a, b)$ 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 $a, b_1, \ldots, b_n$, $\mathsf{AlternateResponse}(a, \{b_1, \ldots, b_n\})$ 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 $b_1, \ldots, b_n$ is *eventually* executed.

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

*Initial* Given activity $a$, $\mathsf{Initial}(a)$ indicates $a$ must be the first activity executed in its scope.

*Mandatory* Given activity $a$, $\mathsf{Mandatory}(a)$ 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 $\langle F, agent_F, patient_F, coref_F, T_F, \mathcal{C}_F, \rangle$, 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)* $\mathcal{C}_F$ 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 $F.\mathsf{activities}$. 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 $s'$:

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

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:

$$\begin{aligned}(\neg en_s\, W\, st_s) \;\wedge\; \Box(st_s \to \Diamond en_s) \;\wedge\\ \Box(st_s \to \circ(\neg st_s\, U\, en_s)) \;\wedge\\ \Box(en_s \to \circ(\neg en_s\, U\, st_s))\end{aligned}$$

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:

$$\Box(st_{s'} \to ((a_t \to (\neg st_s\, U\, en_{s'}))\, U\, en_{s'}))$$

**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.

$$\begin{aligned}
\mathsf{Precedence}_s(\{a_1,..,a_K\},b) &:= \bigvee_{i=1}^{N} \Box(st_s \to (\neg b\, W\, a_i))\\
\mathsf{Response}_s(a,\{b_1,..,b_N\}) &:= \bigvee_{i=1}^{N} \Box(st_s \to (a \to (\neg en_s\, U\, b_i)))\\
\mathsf{NonCoOccurrence}_s(a,b) &:= \Box(st_s \to (a \to (\neg b\, U\, en_s))) \wedge\\
&\phantom{:=\ } \Box(st_s \to (b \to (\neg a\, U\, en_s)))\\
\mathsf{AlternateResponse}_s(a,b) &:= \mathsf{Response}_p(a,b) \;\wedge\\
&\phantom{:=\ } \Box(st_s \to (a \to \circ(\neg a\, U(b \vee en_s))))\\
\mathsf{WeakOrder}_s(a,b) &:= \Box(st_s \to ((\neg en_s\, W\, a \wedge \neg en_s\, W\, b) \to\\
&\phantom{:=\ }\qquad \mathsf{AlternateResponse}_s(a,b)))\\
\mathsf{Terminating}_s(a) &:= \Box(a \to \circ en_s)\\
\mathsf{Initial}_s(a) &:= \Box(st_s \to \circ a)\\
\mathsf{Mandatory}_s(a) &:= \Box(st_s \to (\neg en_s\, U\, a))
\end{aligned}$$

**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 $\mathsf{Sequence}(a,b)$ we refer to the formula $\mathsf{Precedence}(\{a\},b) \wedge \mathsf{Response}(a,\{b\})$ and the $\oplus$ operator is the logical exclusive or.

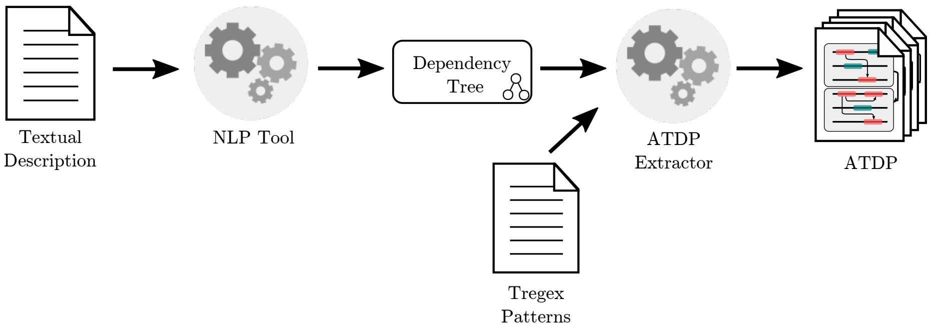


**Fig. 4** Overview of automatic extraction.

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

***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*** $(\circlearrowleft)$ : 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

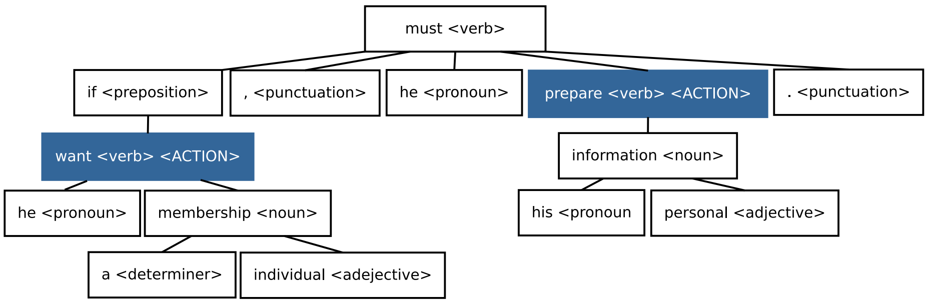


**Fig. 5** Dependency tree representation used to apply patterns.

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.

Tregex[5] [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 unlike 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 stript 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.

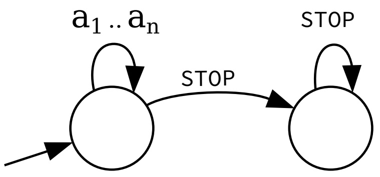


**Fig. 6** Automata representation of the transition system used for the model checker encoding. Activities in $a_1..a_n$ represent both the activity fragments modelled by the user and the start/end activities for every scope in the `ATDP` specification.

## 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 $\text{LTL}_f$ 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 $\text{LTL}_f$ formula $Q$ can be encoded as standard $\text{LTL}_f$ *satisfiability checking* for the following formula:

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

where $A$ is the conjunction of all $\text{LTL}_f$ 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 $\text{LTL}_f$ 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 [8]. 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:

$$(A \wedge \mathcal{C}_F \wedge \mathcal{C}_{T_F} \wedge \Diamond\texttt{STOP}) \implies f(Q) \tag{2}$$

This formula differs from (1) in two respects. First, it contains $\Diamond$`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 [22,11], 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:

$$\begin{aligned} f(a) &= a \\ f(\circ\varphi) &= \circ(f(\varphi) \wedge \neg\texttt{STOP}) \\ f(\neg\varphi) &= \neg f(\varphi) \\ f(\varphi_1 U \varphi_2) &= f(\varphi_1) U (f(\varphi_2) \wedge \neg\texttt{STOP}) \\ f(\varphi_1 \wedge \varphi_2) &= f(\varphi_1) \wedge f(\varphi_2) \end{aligned}$$

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 [11], that the premise is "insensitive to infinity".

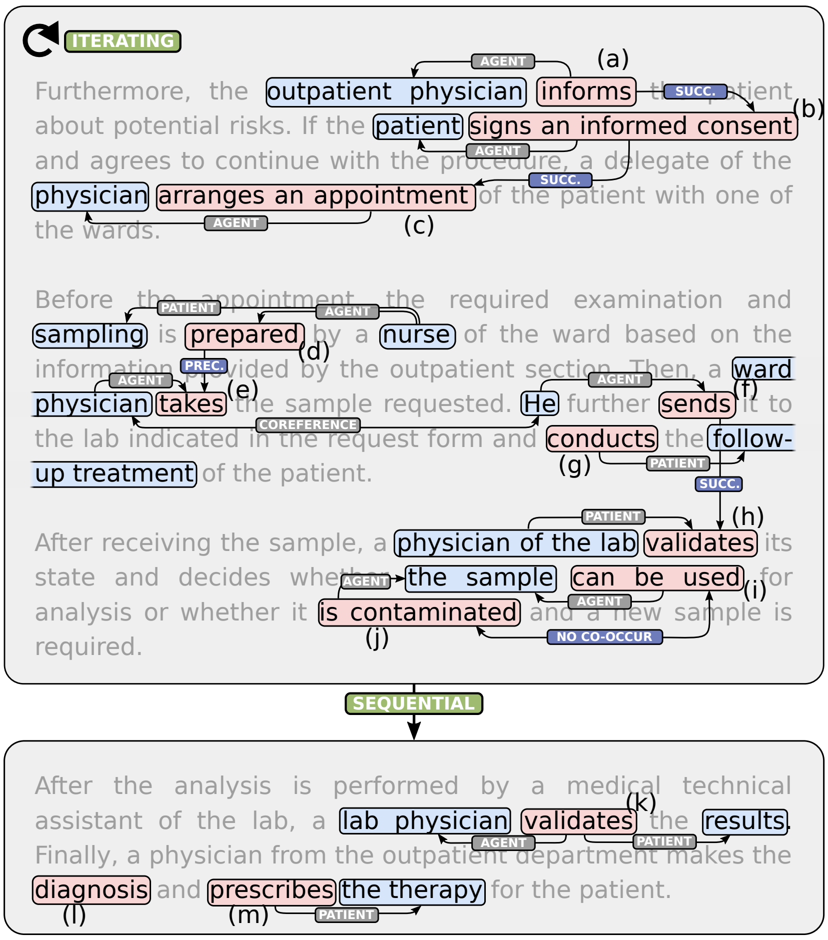
Non-temporal information can be introduced in the queries without increasing the problem complexity, since the information is statically defined. For example, when the text mentions that several activities are performed by a certain role, this information remains invariant during the whole model-checking 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?
$\exists I \in \texttt{ATDP} : (A_I \wedge \mathcal{C}_{F_I} \wedge \mathcal{C}_{T_{F_I}}) \implies Q$
Complete: Is the proposition true in all interpretations of the process?
$\forall I \in \texttt{ATDP} : (A_I \wedge \mathcal{C}_{F_I} \wedge \mathcal{C}_{T_{F_I}}) \implies Q$

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.


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. 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

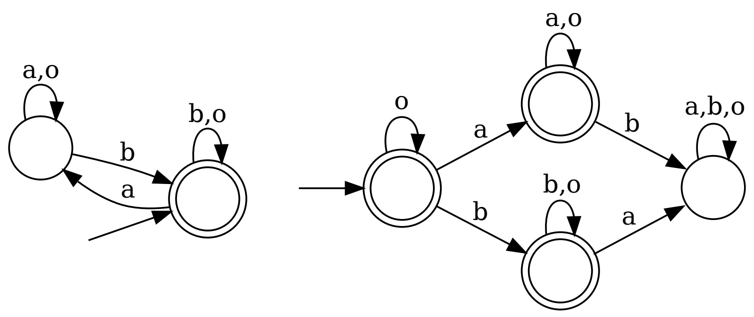


**Fig. 8** Automata implementing two ATDP constraints: Response$(a, b)$ (left) and NonCoOccurrence$(a, b)$ (right). The symbol $o$ is used to indicate any other activities in the scope not affected by this constraint.

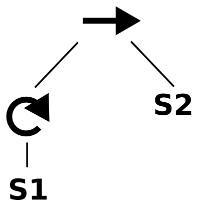


**Fig. 9** Process Tree visualization of the scope-level behaviour of the ATDP in Figure 7

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, 7] 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 [7]. 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

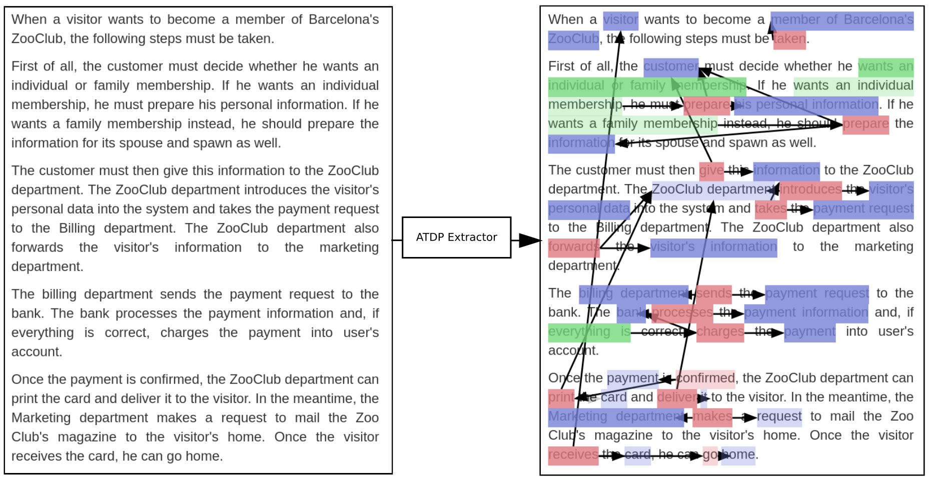
```
def simulate_scope(scope):
    result = []

    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 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-



When a visitor wants to become a member of Barcelona's ZooClub, the following steps must be taken.

First of all, the customer must decide whether he wants an individual or family membership. If he wants an individual membership, he must prepare his personal information. If he wants a family membership instead, he should prepare the information for its spouse and spawn as well.

The customer must then give this information to the ZooClub department. The ZooClub department introduces the visitor's personal data into the system and takes the payment request to the Billing department. The ZooClub department also forwards the visitor's information to the marketing department.

The billing department sends the payment request to the bank. The bank processes the payment information and, if everything is correct, charges the payment into user's account.

Once the payment is confirmed, the ZooClub department can print the card and deliver it to the visitor. In the meantime, the Marketing department makes a request to mail the Zoo Club's magazine to the visitor's home. Once the visitor receives the card, he can go home.

When a visitor wants to become a member of Barcelona's ZooClub, the following steps must be taken.

First of all, the customer must decide whether he wants an individual or family membership. If he wants an individual membership, he must prepare his personal information. If he wants a family membership instead, he should prepare the information for its spouse and spawn as well.

The customer must then give this information to the ZooClub department. The ZooClub department introduces the visitor's personal data into the system and takes the payment request to the Billing department. The ZooClub department also forwards the visitor's information to the marketing department.

The billing department sends the payment request to the bank. The bank processes the payment information and, if everything is correct, charges the payment into user's account.

Once the payment is confirmed, the ZooClub department can print the card and deliver it to the visitor. In the meantime, the Marketing department makes a request to mail the Zoo Club's magazine to the visitor's home. Once the visitor receives the card, he can go home.

**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.

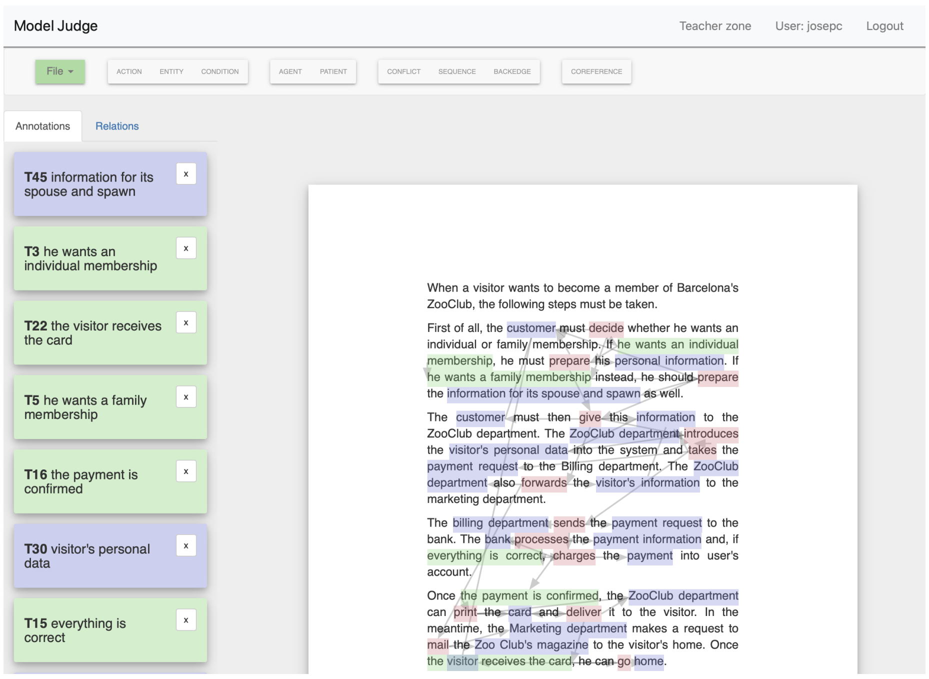
### *Use Case: The Model Judge*

A real application currently using the `ATDP` extractor is the *Model Judge* [14, 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



**T45** information for its spouse and spawn

**T3** he wants an individual membership

**T22** the visitor receives the card

**T5** he wants a family membership

**T16** the payment is confirmed

**T30** visitor's personal data

**T15** everything is correct

When a visitor wants to become a member of Barcelona's ZooClub, the following steps must be taken.

First of all, the customer must decide whether he wants an individual or family membership. If he wants an individual membership, he must prepare his personal information. If he wants a family membership instead, he should prepare the information for its spouse and spawn as well.

The customer must then give this information to the ZooClub department. The ZooClub department introduces the visitor's personal data into the system and takes the payment request to the Billing department. The ZooClub department also forwards the visitor's information to the marketing department.

The billing department sends the payment request to the bank. The bank processes the payment information and, if everything is correct, charges the payment into user's account.

Once the payment is confirmed, the ZooClub department can print the card and deliver it to the visitor. In the meantime, the Marketing department makes a request to mail the Zoo Club's magazine to the visitor's home. Once the visitor receives the card, he can go home.

**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 $Q = \bot$, the reasoner will try to prove that $A \wedge \mathcal{C}_F \wedge \mathcal{C}_{T_F} \rightarrow \bot$, 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 $\bot \to \bot$, if the proof succeeds we will have proven that $A \wedge \mathcal{C}_F \wedge \mathcal{C}_{T_F} \equiv \bot$, 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 $F.\mathsf{activities}$ includes $a_1 =$ `takes (the sample)` and $a_2 =$ `validates (sample state)`, and constraints in $\mathcal{C}_F$ include: $\mathsf{Mandatory}(a_1)$, $\mathsf{Precedence}(\{a_1\}, a_2)$ and $\mathsf{Response}(a_1, \{a_2\})$. `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 $\mathsf{Precedence}(\{a_2\}, a_1)$ to the set of relations $R$. This relation contradicts the previously existing $\mathsf{Precedence}(\{a_1\}, a_2)$, 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: $a_3 =$ `(the sample) can be used`, $a_4 =$ `(the sample) is contaminated`, $a_5 =$ `makes the diagnosis`, and the property can be written in LTL as: $Q = \Box(a_4 \to (\neg a_5 \, U \, a_3))$.

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 $\Box(a_4 \to (\neg a_5 \, U \, a_3))$ 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 [5]. 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 $t = \langle a_1, a_2, \cdots, a_N \rangle$, we can test conformance against an `ATDP` interpretation with the following query[6]:

[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

$$Q = \neg(a_1 \wedge \circ(a_2 \wedge \circ(... \wedge \circ(a_N \wedge \circ\texttt{STOP}))))$$

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 $a_6$ = `informs (the patient)`, $a_7$ = `signs (informed consent)` $a_8$ = `arranges (an appointment)`. Two trace patterns are tested, the first: $t_1 = \langle \cdots a_6, a_7, a_8, \cdots \rangle$ and $t_2 = \langle \cdots a_7, a_6, a_8, \cdots \rangle$. `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.

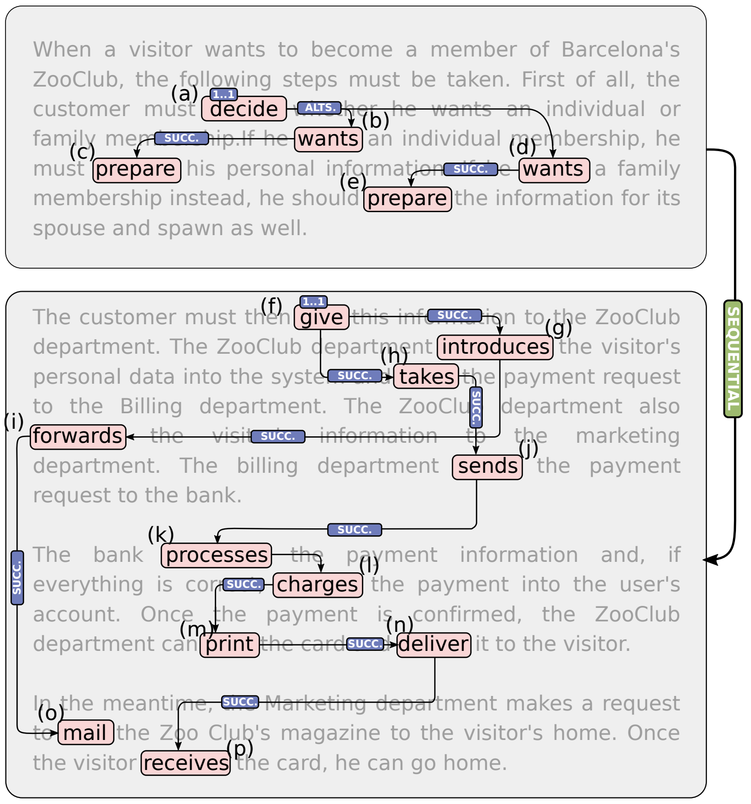
The `ATDP` Simulator is implemented as a command line tool which takes an `ATDP` specification with a single interpretation, and will produce a log file in the XES[55] file format with the requested number of traces.

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.


When a visitor wants to become a member of Barcelona's ZooClub, the following steps must be taken. First of all, the customer must decide whether he wants an individual or family membership. If he wants an individual membership, he must prepare his personal information. If he wants a family membership instead, he should prepare the information for its spouse and spawn as well.

The customer must then give this information to the ZooClub department. The ZooClub department introduces the visitor's personal data into the system and takes the payment request to the Billing department. The ZooClub department also forwards the visitor information to the marketing department. The billing department sends the payment request to the bank.

The bank processes the payment information and, if everything is correct, charges the payment into the user's account. Once the payment is confirmed, the ZooClub department can print the card and deliver it to the visitor.

In the meantime, the Marketing department makes a request to mail the Zoo Club's magazine to the visitor's home. Once the visitor receives the card, he can go home.

**Fig. 12** Control-flow projection of scope the ATDP specification of the Zoo process.

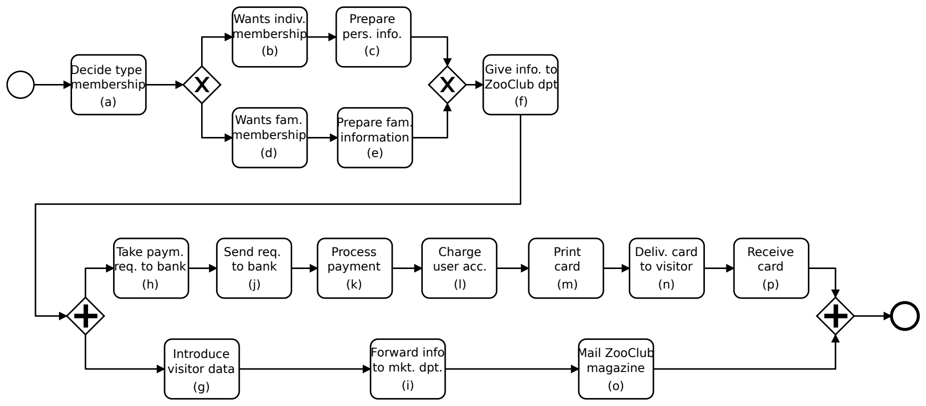

Decide type membership (a)
Wants indiv. membership (b)
Prepare pers. info. (c)
Wants fam. membership (d)
Prepare fam. information (e)
Give info. to ZooClub dpt (f)
Take paym. req. to bank (h)
Send req. to bank (j)
Process payment (k)
Charge user acc. (l)
Print card (m)
Deliv. card to visitor (n)
Receive card (p)
Introduce visitor data (g)
Forward info to mkt. dpt. (i)
Mail ZooClub magazine (o)

**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 [5]. 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 adapter 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. 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.
4. Andrea Burattin. PLG2: multiperspective process randomization with online and offline simulations. In *BPM Demo Track*, 2016.
5. Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich. *Conformance Checking - Relating Processes and Models.* Springer, 2018.
6. 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.
7. 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.
8. Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A new symbolic model checker. *STTT*, 2(4):410–425, 2000.
9. 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.
10. 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.
11. 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.
12. 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.
13. Luis Delicado, Josep Sanchez-Ferreres, Josep Carmona, and Lluís Padró. NLP4BPM - Natural Language Processing Tools for Business Process Management. In *BPM Demo Track*, 2017.
14. Luis Delicado, Josep Sanchez-Ferreres, Josep Carmona, and Lluis Pardo. The Model Judge - A Tool for Supporting Novices in Learning Process Modeling. In *BPM Demo Track*, 2018.
15. Remco Dijkman, Irene Vanderfeesten, and Hajo A. Reijers. Business process architectures: overview, comparison and framework. *Enterprise Information Systems*, 10(2):129–158, feb 2016.
16. Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management.* Springer, 2018.
17. 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.
18. Christiane Fellbaum. *WordNet. An Electronic Lexical Database.* Language, Speech, and Communication. The MIT Press, 1998.
19. 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.
20. Günther Fliedl, Christian Kop, and Heinrich C Mayr. From textual scenarios to a conceptual schema. *Data & Knowledge Engineering*, 55(1):20–37, 2005.
21. Fabian Friedrich, Jan Mendling, and Frank Puhlmann. 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, EMMSAD 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 Ottensooser, Alan Fekete, Hajo A. Reijers, Jan Mendling, and Con Menictas. Making sense of business process descriptions: An experimental comparison of graphical and textual notations. *Journal of Systems and Software*, 85(3):596–606, 2012.
34. 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.
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. Franziska Semmelrodt. *Modellierung klinischer Prozesse und Compliance Regeln mittels BPMN 2.0 und eCRG.* PhD 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.