Quishpi, L.; Carmona, J.; Padró, L. Extracting annotations from textual descriptions of processes. A: International Conference on Business Process Management. "Business Process Management, 18th International Conference, BPM 2020: Seville, Spain, September 13-18, 2020: proceedings". Berlín: Springer, 2020, p. 184-201. ISBN 978-3-030-58666-9.
The final authenticated version is available online at https://doi.org/10.1007/978-3-030-58666-9_11.

Extracting Annotations from Textual Descriptions of Processes

Luis Quishpi, Josep Carmona, and Lluís Padró

Computer Science Department
Universitat Politècnica de Catalunya
Barcelona, Spain.
{quishpi,jcarmona,padro}@cs.upc.edu

Abstract. Organizations often have textual descriptions as a way to document their main processes. These descriptions are primarily used by the company's personnel to understand the processes, specially for those ones that cannot interpret formal descriptions like BPMN or Petri nets. In this paper we present a technique based on Natural Language Processing and a query language for tree-based patterns, that extracts annotations describing key process elements like actions, events, agents/patients, roles and control-flow relations. Annotated textual descriptions of processes are a good compromise between understandability (since at the end, it is just text), and behavior. Moreover, as it has been recently acknowledged, obtaining annotated textual descriptions of processes opens the door to unprecedented applications, like formal reasoning or simulation on the underlying described process. Applying our technique on several publicly available texts shows promising results in terms of precision and recall with respect to the state-of-the art approach for a similar task.

1 Introduction

The consolidation of the BPM discipline in organizations is closely related to the ability of going beyond current practices, especially for the assumption that data that talks about processes is an structured source of information. In practice, this assumption is often not met, i.e., organizations document their processes in textual descriptions, as a way to bootstrap their accessibility [15].

The ubiquity of textual descriptions of processes has caused recent research in proposing mechanisms to make actionable this information source, e.g., the discovery of formal process models [7,16], the alignment between structured and non-structured process information [13,17], or the use of annotations and natural language techniques to encompass process information [12,10].

The contribution in [12] shows how formal reasoning is possible on top of *Annotated Textual Descriptions of Processes* (`ATDP`, see an example in Figure 1), by equipping `ATDP` with a formal, trace semantics that links any `ATDP` specification to a linear temporal logic formula. However, in [12] (but also in similar works

like [10]), it is assumed that these annotations are manually crafted, thus hampering the ability to automate the application of formal analysis on annotated textual descriptions of processes.

In this paper we propose a novel technique to extract an important subset of the ATDP language. In contrast to existing approaches, we use a query language that considers the hierarchical structure of the sentences: our technique applies flexible tree-based patterns on top of the *dependency tree* corresponding to the NLP analysis of each sentence. This makes the approach very robust to variations that can arise in the discursive description of the main process elements, but also contributes to reduce considerably the number of patterns that need to be defined and maintained to perform the extraction.

The open-source tool related to this paper contributes to make ATDP specifications actionable. Currently, several functionalities are available[1]: an ATDP library, an ATDP editor, an ATDP interface to a model checker, and now an ATDP extractor.

Running Example Figure 1 shows the results of the technique proposed in this paper. The text describes the process of examining patients in a hospital[2]. For this text, the techniques of this paper are able to automatically extract activities (highlighted in red), roles (highlighted in blue), conditions (highlighted in green), and relations between these extracted elements (e.g., control-flow relations between actions).

The paper is organized as follows: next section shortly describes the work related to this contribution. In Section 3 we overview the main ingredients of this paper contribution, presented in Section 4. Experiments and tool support are reported in Section 5, whilst Section 6 concludes the paper and provides suggestions for future work.

2 Related work

For the sake of space, we only report here the related work that focuses on the extraction of process knowledge from textual descriptions [1,7,16] or legal documents [3,19], or the work that considers textual annotations in the scope of BPM [10,12].

For the former, the work by Friedrich et al. [7] is acknowledged as the state-of-the-art for extracting process representations from textual descriptions, so we focus our comparison on this approach. As we will see in the evaluation section, our approach is significantly more accurate with respect to the state-of-the-art in the extraction of the main process elements. Likewise, we have incorporated as well the patterns from [16], and a similar outcome is reported in the experiments. Moreover, we believe that our ideas can be easily applied in the scope of legal documents, proposed in [3,19].

[1] https://github.com/PADS-UPC

[2] This example is inspired from [14].

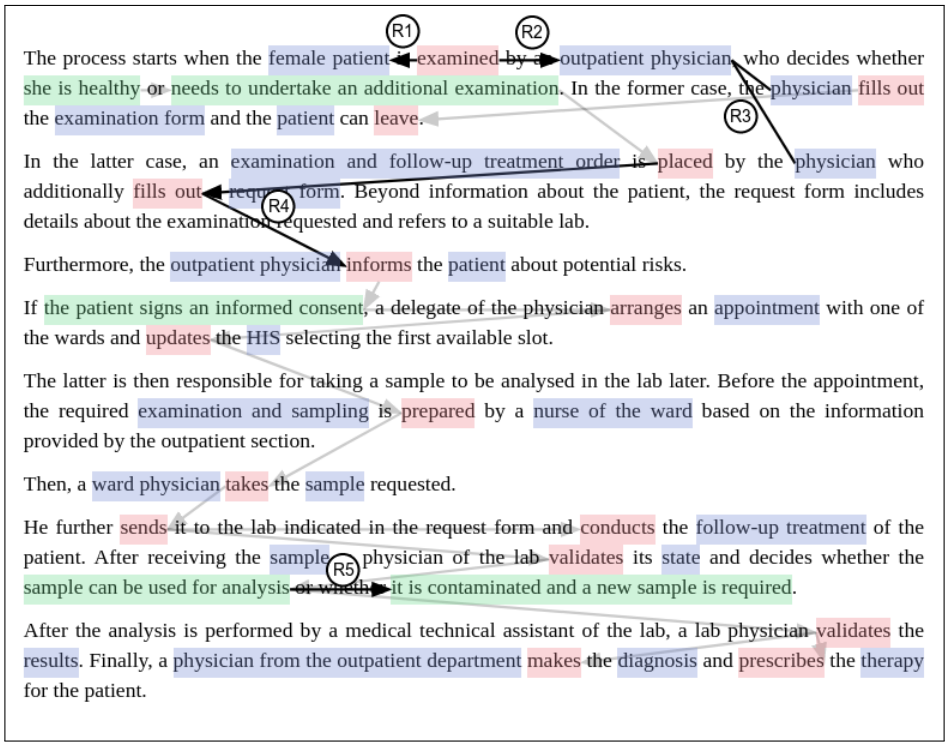


Fig. 1. Extracted annotated textual description of a patient examination process. R1: Patient of `examine` is `female patient`, R2: Agent of `examine` is `outpatient physician`, R3: Coreference between `outpatient physician` and `physician`, R4: Sequential between `fill out` and `informs`, R5: Conflict between conditions `sample can be used for analysis` and `it is contaminated and a new sample is required`

For the later type of techniques ([10,12]), we see these frameworks as the principal application for our techniques. In particular, we have already demonstrated in the platform `https://modeljudge.cs.upc.edu` an application of the use of annotations in the scope of teaching and learning process modeling[3].

3 Preliminaries

3.1 Natural Language Processing and Annotation

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

[3] The reader can see a tutorial for annotating process modeling exercises in the `ModelJudge` platform at `https://modeljudge.cs.upc.edu/modeljudge_tutorial/`.

used in this work is FreeLing[4] [11], 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 (words, punctuation signs, numbers, etc.), and group these tokens into sentences.

Morphological analysis: Find out all possible parts-of-speech (PoS) for each token.

PoS-Tagging: Determine the right PoS for each word in a sentence. (e.g. the word *dance* is a verb in *I dance all Saturdays* but 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 [5] as the sense catalogue and synset codes as concept identifiers.

Dependency parsing: Given a sentence, get its syntatic structure as a dependency parse tree (DT). DT are an important element in our approach. The reader can see an example of a dependency tree in Fig. 4.

Semantic role labeling: Given a sentence, identify its predicates and the main actors (agent, patient, recipient, etc) involved in each predicate, regardless of the surface structure of the sentence (active/passive, main/subordinate, etc.). E.g. In the sentence *John gave Mary a book written by Peter*, SRL would extract two predicates: `give` (with semantic roles `Agent(John,give)`, `Patient(book,give)`, and `Recipient(Mary,give)`), and `write` (with semantic roles:
`Agent(Peter,write)` and `Patient(book,write)`).

Coreference resolution: Given a document, group mentions referring to the same entity (e.g. a person can be mentioned 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 labeling identifies semantic relations (the *agent* of an action is the same regardless of whether the sentence is active or passive). Coreference resolution links several mentions of an actor as referring to the same entity (e.g. in Figure 1, *a delegate of the physician* and *the latter* refer to the same person. Also, the same object is mentioned as *the sample requested* and *it*).

All NLP components are developed and evaluated on standard benchmarks in the area, obtaining state-of-the art results. In particular, dependency trees –which are the basic element upon which we build our proposal– are created by a Deep Learning parser [4] which obtains accuracy scores over 90%.

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

3.2 Annotated Textual Descriptions of Processes (ATDP)

ATDP is a formalism proposed in [12], aiming to represent process models on top of textual descriptions. This formalism naturally enables the representation of a wide range of behaviors, ranging from procedural to completely declarative, but also hybrid ones. Different from classical conceptual modeling principles, this highlight ambiguities that can arise from a textual description of a process, so that a specification can have more than one possible interpretation[5].

ATDP specifications can be translated into linear temporal logic over finite traces [8,2], opening the door to formal reasoning, automatic construction of formal models (e.g. in BPMN) from text, and other interesting applications such as simulation: to generate end-to-end executions (i.e., an *event log* [18]) that correspdond to the process described in the text, which would allow the application of *process mining* algorithms.

In Figure 1, 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*.

ATDP models are defined over an input text, which is marked with *typed text fragments*, which may correspond to *entities*, or *activities*. Marked fragments can be related among them via a set of *fragment relations*.

Entity fragments. The types of entity fragments defined in ATDP are:

- *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` in Figure 1.
- *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 process activities. An example is the `(medical) sample` obtained and analyzed by physicians during the patient examination process.

Activity fragments. ATDP distinguishes the following types of activity fragments:

- *Condition.* It is considered discourse markers that mark conditional statements, like: *if*, *whether* and *either*. Each discourse marker needs to be tailored to a specific grammatical structure.
- *Task and Event.* Those fragment types are 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 task fragment is `validates`

[5] In this work we consider a flattened version of the ATDP language, i.e., without the notion of *scopes*.

(`results`). Event fragments are 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.

Fragment Relations. Text fragments can be related to each other by means of different relations, used to express properties of the process emerging from the text:

- *Agent.* Indicates the role responsible for the execution of an activity.
- *Patient.* Indicates the role or business object on which an activity is performed.
- *Coreference.* Induces a coreference graph where each connected component denotes a distinct process entity.
- *Sequential.* Indicates the sequential execution of two activity fragments `A1` and `A2` in a sentence. We consider two important relations from [12]: Precedence and Response. Moreover, to cover situations where ambiguities in the text prevent selecting any of the two aforementioned relations, we also incorporate a less restrictive constraint WeakOrder, that only applies in case both activities occur in a trace.
- *Conflicting.* A conflict relation between two condition activity fragments $\langle C1, C2 \rangle$ in a sentence indicates that one and only one of them can be executed, thus capturing a choice. This corresponds to the relation NonCoOccurrence from [12].

Once `ATDP` are extracted, several possibilities arise, ranging from simulation, to formal analysis. For any of the previous techniques, there are available open-source libraries (see Section 5).

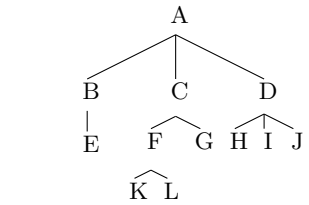
3.3 TRegex

In this paper, we use Tregex[6] [9], a query language that allows the definition of regular-expression-like patterns over tree structures. Tregex is designed to match patterns involving the content of tree nodes and the hierarchical relations among them. In our case we will be using Tregex to find substructures within syntactic dependency trees. Applying Tregex patterns on a dependency tree allows us to search for complex labeled tree dominance relations involving different types of information in the nodes. The nodes can contain symbols or a string of characters (e.g. lemmas, word forms, PoS tags) and Tregex patterns may combine those tags with the available dominance operators to specify conditions on the tree. Additionally, as in any regular expression library, subpatterns of interest may be specified and the matching subtree can be retrieved for later use. This is achieved in Tregex using unification variables as shown in pattern (2) in Figure 2.

Figure 2 describes the main Tregex operators used in this research to specify pattern queries.

[6] `https://nlp.stanford.edu/software/tregex.html`

Operator	Meaning
`X << Y`	X dominates Y
`X >> Y`	X is dominated by Y
`X !>> Y`	X is not dominated by Y
`X < Y`	X immediately dominates Y
`X > Y`	X is immediately dominated by Y
`X >, Y`	X is the first child of Y
`X >- Y`	X is the last child of Y
`X >: Y`	X is the only child of Y
`X $-- Y`	X is a right sister of Y



(1) `E>>(A<<G)`	(4) `F!>>A`
(2) `E>>(A=x)>:(B=y)`	(5) `H>:D`
(3) `K!>>B>>(A<D)`	(6) `A<J`

Fig. 2. Some operators provided by Tregex (left). The tree on the right would match patterns (1), (2), (3), and would not match patterns (4), (5), (6). Note that unless parenthesized, all operators refer to the first element in the pattern. Pattern (2) captures nodes `A` and `B` into variables `x` and `y`.

4 Approach

Our proposed technique automatically extracts ATDP elements applying Tregex patterns on textual description of business process, commonly used in industry and business organizations [15]. The result is a set of ATDP elements that may be used to derive formal representations of process models, generate logs via simulation, or automatically reason about process behaviour (see [12] for more details on `ATDP` possibilities).

The technique follows the steps shown in Figure 3. The first step consists of performing a NLP analysis to extract, among other information, a dependency tree for each sentence (see Section 3.1).

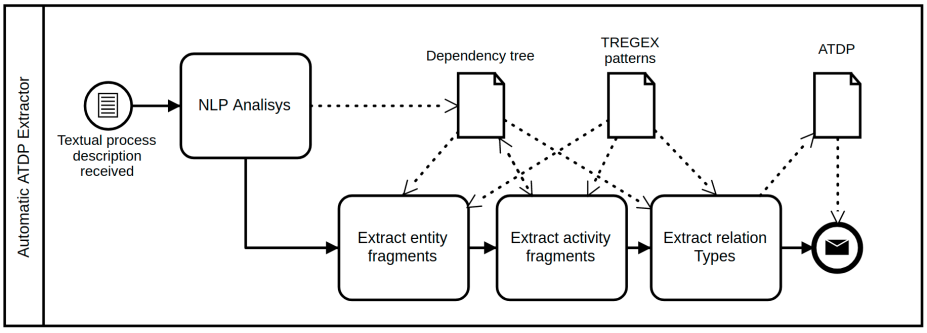


Fig. 3. General framework for automatic `ATDP` extraction

The obtained dependency trees (one for each sentence) are transformed to a format suitable for Tregex patterns: A node in the transformed dependency tree is a structured string, containing information about the lemma, PoS tag, and syntactic function of each word. Additionally, nodes marked as predicates

by the NLP semantic role labeling step are decorated with an extra <ACTION> label, that identifies them as potential activity fragments. Figure 4 shows the transformed tree for the input sentence "*If the patient signs an informed consent, a delegate of the physician arranges an appointment with one of the wards and updates the HIS selecting the first available slot*".

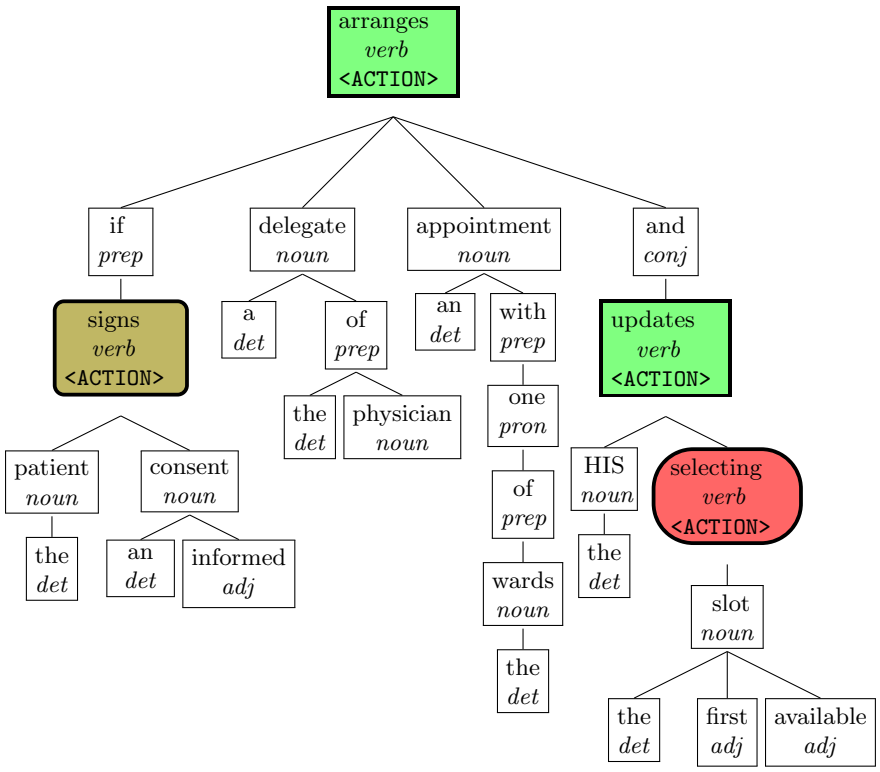


Fig. 4. Dependency tree for the sentence "*If the patient signs an informed consent, a delegate of the physician arranges an appointment with one of the wards and updates the HIS selecting the first available slot*". Oval red node *selecting* is filtered out as a process activity since it is inside a subordinate clause. Round-corner olive *signs* is marked as a condition activity, since it is inside the conditional clause. Sharp-corner green nodes *arranges* and *updates* are kept as process activities since they are at the main clause level in the sentence (accounting for the *and* coordinating structure).

Next step consists of extracting process entities (actors and business objects) as described below in Section 4.1. Afterwards, we extract activity fragments relying on an incremental procedure based on Tregex patters: First, a set of patterns identify which <ACTION> nodes are actually conditions, and change their mark to <CONDITION>. Next, other patterns discard non-relevant <ACTION> nodes, and relabel relevant ones as <ACTIVITY>. Finally, a last batch of patterns decide

which of the remaining actions should be relabeled as `<EVENT>`. See Section 4.2 for details on each of these pattern sets. Finally, after relabeling the appropriate tree nodes, the last set of patterns extracts relations between the resulting activity nodes (Section 4.3).[7]

4.1 Extraction of entity fragments.

Roles and Business objects. To identify the roles –i.e. autonomous actors– of the process, we leverage the results from the NLP analysis and focus on the elements with a semantic role of *Actor*. Similarly, business object are detected extracting elements with semantic role of *Object*. For each of those elements, the extracted text should be modified to better represent the role or business object: fragments that begin with *the* or prepositions such as *by*, *of* or *from* can be modified to not contain these elements, but just the core description of the role/object.

To that end, the following Tregex pattern is recursively applied to the dependency tree to select the relevant modifiers of the main entity word (i.e. nouns and adjectives directly modifying the head noun in the phrase):

PE1 `/noun|adjective/=result > /EntityHeadWord/`

This pattern will extract only nouns and adjectives directly modifying each main entity word (note that this pattern will we instantiated for each word heading a phrase identified as *Actor* or *Object* by the NLP).

For instance, 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 labeling step for *Agent* would return the whole subtree headed by *physician* (i.e. *an outpatient physician, who decides... examination*). The role entity fragment Tregex pattern will strip down such a long actor/object removing the determiner and the relative clause, while keeping the core actor/object and its main modifiers, thus extracting respectively `outpatient physician` as a role, and `female patient` as a business object.

4.2 Extraction of Activity fragments.

In order to extract activity fragments, we rely on the output of NLP, which marks as predicates all non-auxiliary verbs as well as some nominal predicates (e.g. *integration*, *meeting*, 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, or relabel them as *condition* or *event* fragment.

[7] It is important to notice that the selection, configuration and application of rules influences the `ATDP` extraction. Exploring possible extensions and strategies to achieve a complete and minimal set of rules will be object of future research.

Condition fragments. To detect conditional phrases, we use patterns that check for nodes having part-of-speech with a domination relation with nodes containing words such as "`if`", "`whether`" or "`either`". If a match is found, the tree node captured in variable `result` is marked as `<CONDITION>`. Condition patterns are:

PC1 `/<verb>/=result >, /whether/`
PC2 `/<verb>|<not>/=result >, (/or/ >> /<whether>/)`
PC3 `/<verb>/=result >, /if/`
PC4 `/<verb>/=result < /either/=condition << /or/`
PC5 `/<verb>/=result > (/or/ >> (/<verb>/ < /either/))`

The first pattern checks for a node `<verb>` that is the first child of a node with word "`whether`". In the second pattern, the matched node is either a `<verb>` or the word `<not>` that is the first child of a node with word "`or`" which is in turn child of node "`if`".

For instance, patterns PC1 and PC2 can be used to determine that `she is healthy` and `needs to undertake an additional examination` are conditions in the sentence "*... who decides whether she is healthy or needs to undertake an additional examination.*". Similarly, pattern PC3 matches the tree in Figure 4, and extracts the condition fragment `the patient signs an informed consent`.

Finally, patterns PC4 and PC5 extract the conditional phrase in sentences containing "`either...or`" constructions.

Event fragments. The strategy to identify events is similar to conditional fragments identification. The main difference is that, instead of conditional markers, we consider the presence of time markers such as *once*, *as soon*, *when* and *whenever*. The following Tregex expressions are used to identify event fragments:

PV1 `/<ACTION>/=result > /once/`
PV2 `/<ACTION>/=result > (/be/ > /once/)`
PV3 `/<ACTION>/=result >> (/as/ >- /soon/)`
PV4 `/<ACTION>/=result <, /when/`
PV5 `/<ACTION>/=result < /whenever/`

If a match is found, the tree node captured in variable `result` is marked as `<EVENT>`. For instance, in the sentence "*Once the payment is confirmed, the ZooClub department can print the card...*", after applying pattern PV2, the fragment `confirm(payment)` is identified as an event fragment.

Task fragments. Task fragments represent the atomic activity of execution inside the process model, and their identification is done after extracting conditions and events.

PT1 `/be <ACTION>/=toRemove`
PT2 `/start <ACTION>/=toRemove`
PT3 `/have <ACTION>/=toRemove`

PT4 `/want <ACTION>/=toRemove`

These patterns simply discard verbs *be*, *start*, *have* and *want* as activities. Subjective verbs (e.g. *want*, *think*, *believe*, etc.) are unlike to describe activities and thus are filtered out. For instance in the sentence "*The process starts when the female patient is examined by an outpatient physician,...*", *start* is removed from the activity list, while *examine* is kept.

The following patterns remove any action candidates that are in a subordinate clause under another action.

PT5 `/<ACTION>/=toRemove >> /<ACTION>/=result !>> /and|or/`
PT6 `/<ACTION>/=toRemove >> (/<ACTION>/=result >> /and|or/)`

The idea is that a subordinate clause is describing details about some element in the main clause, but it is not a relevant activity thus it must be removed. 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 first 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 pattern application), meanwhile in the second pattern takes as reference the coordinating conjunction *and*/*or* to remove any action candidates that are in a subordinate clause under the main action.[8]

If a match is found by patterns PT5 or PT6, the tree node captured in variable `result` is marked as `<ACTIVITY>` and in the tree node captured in variable `toRemove` the tag `<ACTION>` is removed. For instance, in the tree in Figure 4, the action *select (first available slot)* that is under *update (HIS)* is discarded by pattern PT5.

4.3 Relation types

Agent, Patient and Coreference. These relations are straightforwardly extracted from the output (predicates and arguments) of the Semantic Role Labelling phase. However, only those involving entities or activities detected by the patterns described above are considered. For instance, in the first sentence of our running sample (Figure 1) we have that the *agent* for activity `examine` is `outpatien physician`, while the *patient* is `female patient`, stating that `examine` someone is under the responsibility of a `outpatient physician` and the `examine` activity operates over a `female patient` business object.

Regarding coreference, in our running example (Figure 1), all text fragments pointing to the `patient` role corefer to the same entity, whereas there

[8] Observe that the absence of parenthesis in pattern 1 means that the "`!>>/and|or/`" condition is applied to the first action in the pattern, while in pattern 2, the parenthesis force that condition on the second action.

are three different physicians involved in the text: the `outpatient physician`, the `physician` and the `ward physician`, which form disconnected coreference subgraphs.

Sequential. The extraction of sequential relations is performed on the results of activity fragment extraction described in Section 4.2. We use patterns to capture Precedence, Response and WeakOrder constraints, which express the order of execution of the activities in an ATDP specification.

These first patterns can be used to identify a particular case of Response that typically occurs in conditional sentences:

```
PR1 /<CONDITION>/=from >, (/if/ >/<ACTIVITY>/=to)
PR2 /<CONDITION>/=from >, (/if/ >(/verb/ </<ACTIVITY>/=to))
PR3 /<CONDITION>/=from >, (/if/>(/verb/<(/be|to/</<ACTIVITY>/=to)))
```

These patterns capture the case where a `<CONDITION>` (identified by previous patterns) is inside an *if* clause (that is, below the *if* token in the dependency tree), which has an `<ACTIVITY>` as the condition's consequent. In those cases, it is safe to assume that the activity in the consequent *responds to* the occurrence of the condition. For instance, in the dependency tree in Figure 4, pattern PR1 would extract that `arrange (appointment)` *responds to* `sign(consent)`.

With the following patterns we are able to extract sequential relations between `<EVENT>` and `<ACTIVITY>` nodes. These are typical cases of Precedence:

```
PP1 /<EVENT>/=from > /<ACTIVITY>/=to
PP2 /<EVENT>/=from >> (/verb/ << /<ACTIVITY>/=to)
```

For instance, pattern PP1 checks for an `<EVENT>` immediately dominated by an `<ACTIVITY>`, so in the sentence "*An intaker keeps this registration with him at times when visiting the patient*", it would extract the sequential relation from `visit(patient)` to `keep(registration)`.

Pattern PP2 checks for an `<EVENT>` that is dominated by a verb which also dominates an `<ACTIVITY>`. When applied to the sentence "*Once the payment is confirmed, the ZooClub department can print the card*" we are able to extract the sequence between `confirm(payment)` and `print(card)`.

We also extract sequences from one condition fragment to several activities using the following patterns:

```
PS1 /<CONDITION>/=to >> (/whether/ >> /<ACTIVITY>|<EVENT>/=from)
PS2 /<CONDITION>/=to >> /<ACTIVITY>|<EVENT>/=from < /either/
PS3 /<CONDITION>/=to >> (/or/ >> /<ACTIVITY>|<EVENT>/=from)
```

For instance 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.*" pattern PS1 will extract the sequential relations `examine(patient)` → `is healthy`, and `examine (patient)` → `needs (undertake examination)`.

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, using the patterns:

```
PW1 /<ACTIVITY>/=from < (/and/ << (/<ACTIVITY>/=to < /then/ ))
PW2 /<ACTIVITY>/=from << (/and/ << (/<ACTIVITY>/=to < /then/))
PW3 /<ACTIVITY>/=from >> (/after/ > /<ACTIVITY>/=to)
PW4 /<ACTIVITY>/=from >> (/after/ > (/be/ < /<ACTIVITY>/=to))
PW5 /<ACTIVITY>/=to > (/be/>(/after/>(/<ACTIVITY>/=from>/be/)))
PW6 /<ACTIVITY>/=from < (/before/ < /<ACTIVITY>/=to)
PW7 /<ACTIVITY>|<EVENT>/=from << (/<ACTIVITY>/=to < /and/)
PW8 /<ACTIVITY>/=from < (/and/ < /<ACTIVITY>/=to)
```

To illustrate the WeakOrder, using pattern PW1 we can infer that `generate` and `pay` are in WeakOrder in the sentence "*The Payment Office of SSP generates a payment report and then pays the vendor*".

Additionally, we incorporated the approach used in [16] to extract relations Response, Precedence, and Succession based on the mandatory aspect of each activity: Given a sentence stating that activity A happens before activity B, if both A and B are mandatory, the relation is marked as Succession. If B is mandatory but A is not, the result is a Response relation. If B is not mandatory, a Precedence relation is extracted. Patterns M1, M2, M3 below are used to decide whether a task is mandatory or not. Their occurrence order is determined by the patterns previously presented in this section.

```
M1 /<ACTIVITY>/ >/must|will|would|shall|should/
M2 /<ACTIVITY>/ >(/be/ >/must|will|would|shall|should/)
M3 /<ACTIVITY>/ >(/be/ >(/have/ >/must|will|would|shall|should/))
```

Conflict. 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 conditional discourse markers that affect `<CONDITION>` nodes extracted by patterns in section 4.2:

```
PX1 /whether/ << (/<CONDITION>/=from << (/or/<</<CONDITION>/=to))
PX2 /<CONDITION>/=from << (/or/ << /<CONDITION>/=to)
```

For instance, pattern PX1 would extract the constraint that the sample can not be safely used and contaminated at the same time from the sentence "*... decides whether the sample can be used for analysis or whether it is contaminated*"

Pattern PX2 extracts general conflicts, such as the conditional fragments `approve` and `deny` from the sentence "*The next step is for the IT department to analyse the request and either approve or deny it.*" are considered in conflict. In order to this pattern to work, previously the verbs `approve` and `deny` were labeled as `<CONDITION>` in section 4.2, patterns PC4 and PC5.

5 Tool Support and Experiments

This section presents experiments evaluating the performance of the proposed approach[9] to automatically extract `ATDP` from text. We will report two different experiments for two types of evaluation: we will compare activity extraction with a baseline based on [6], and relation extraction with the recent approach [16].

[9] `https://github.com/PADS-UPC/atdp-extractor`

Activity Extraction. The evaluation is performed comparing the activity fragments extracted against gold standard annotations carried out by a human, who selected just the activities deemed relevant to the process. We collected a test data set consisting of 18 of those text-model pairs, shown in Table 1. The first 13 models stem from material in the appendix of Master thesis [6], and the last 5 from our academic dataset[10] used in [13]. In both cases each example includes a textual process description paired with the corresponding BPMN models created by a human. As a gold reference for evaluation, we manually created one `ATDP` for each example following the activities in those BPMN models, i.e. marking as activity fragments only the text pieces that had a corresponding element in the BPMN model.

Since the approach in [6] is not publicly available, we wrote Tregex patterns that would mimic that reference approach. Namely, we use patterns that take the output of the NLP analysis step, we filter out weak verbs `be`, `have`, `do`, `achieve`, `start`, `exist`, and `base`, and we use conditional indicators like `if`, `whether`, and `otherwise` to detect conditions.

Table 1 shows the results obtained by our tool compared with the baseline based in [6], which relies in extracting as activities most of the verbs detected by the NLP tool. The former scenario uses all the patterns described in this paper. Precision is computed as the percentage of right fragments among predicted fragments ($P = \#ok/\#pred$). Recall is the percentage of expected fragments extracted ($R = \#ok/\#gold$). F_1 score is the harmonic mean of precision and recall ($F_1 = 2PR/(P+R)$). We only count extracted fragments as right if they match the gold annotations in words and type (`<ACTIVITY`, `<EVENT>`, `<CONDITION>`).

Obtained results show that the baseline based in [6] obtains higher recall (since it extracts more activities), but lower precision (since many of the extracted activities are not in the gold annotations, i.e. they are not relevant for the process). On the other hand, our approach is a bit more conservative and extracts less activities, thus getting slightly lower recall, but largely higher precision. Overall, the trade-off F_1 score is consistently better when using our tree-based patterns.

Relation Extraction. To evaluate the extraction of relations, we compare our approach with the one presented in [16][11]. In addition to the 18 benchmarks from Table 1, we have added the data collection used in [16]. In this later case, we exclude from the evaluation the WeakOrder constraints generated by our system, since the reference dataset does not include such relation. Table 2 reports the comparison, that shows again the tendency to improve the results both for the dataset used in [16] and our dataset.

[10] `https://github.com/setzer22/alignment_model_text/tree/master/datasets/NewDataset`

[11] `https://github.com/hanvanderaa/declareextraction`

Source	#gold	Patterns emulating [6]					Our patterns				
		#pred	#ok	P	R	F_1	#pred	#ok	P	R	F_1
1-1_bicycle_manufacturing	15	22	11	50	**73**	60	18	11	**61**	**73**	**67**
1-2_computer_repair	10	14	9	64	**90**	75	10	8	**80**	80	**80**
2-1_sla_violation	46	93	41	44	**89**	59	70	40	**57**	87	**69**
3-1_2009-1_mc_finalice	8	15	7	47	**88**	61	10	6	**60**	75	**67**
3-2_2009-2_conduct_directions	7	12	7	58	**100**	74	9	6	**67**	86	**75**
3-6_2010-1_claims_notification	12	19	12	63	**100**	77	15	12	**80**	**100**	**89**
4-1_intaker_workflow	34	55	23	42	**68**	52	49	23	**47**	**68**	**55**
5-1_active_vos_tutorial	8	10	8	**80**	**100**	**89**	10	8	**80**	**100**	**89**
6-1_acme	22	41	20	49	**91**	64	36	19	**53**	86	**66**
7-1_calling_leads	6	11	6	55	**100**	**71**	9	5	**56**	83	67
8-1_hr_process_simple	5	7	5	71	**100**	**83**	5	4	**80**	80	80
9-2_exercise_2	8	16	8	50	**100**	67	9	8	**89**	**100**	**94**
10-2_process_b3	15	21	14	**67**	**93**	**78**	21	14	**67**	**93**	**78**
Total	196	336	171	51	**87**	64	271	164	**61**	84	**70**
1081511532_rev3	10	14	8	57	**80**	67	10	7	**70**	70	**70**
1120589054_rev4	11	17	11	65	**100**	79	14	11	**79**	**100**	**88**
1364308140_rev4	12	13	9	69	**75**	72	10	8	**80**	67	**73**
20818304_rev1	12	12	8	**67**	**67**	**67**	11	7	64	58	61
784358570_rev2	16	27	13	48	**81**	61	22	13	**59**	**81**	**68**
Total	61	83	49	59	**80**	68	67	46	**69**	75	**72**
TOTAL	257	419	220	53	**86**	65	338	210	**62**	82	**71**

Table 1. Column *#gold* contains the number of activity fragments marked by a human as relevant to the process. Columns *#pred* and #ok show the number of fragments predicted by the tool, and how many of them were in the gold annotations. Columns P, R, and F_1 show precision, recall and F-measure respectively.

6 Conclusions and Future Work

We have proposed a novel technique to extract `ATDP` specifications from textual descriptions. Continuing with our effort to unleash the use of unstructured data that talks about processes, this is one of the key functionalities that was missing. When comparing our approach with the state-of-the-art technique, we witness an improvement in accuracy for the task of detecting the main process elements.

For future work, we will extend the approach to also extract other `ATDP` elements, specially the ones regarding *scopes* and their relations, that are a real challenge since they need to be defined at a different granularity level. Another idea to explore is the definition of patterns across sentences, that may help improving the accuracy, but needs an adaption for the use of our query language, which is defined over single sentences. Orthogonal to these two lines, we also will explore the incorporation of machine learning techniques that can use the annotations performed by an expert (e.g., in the platform `Model Judge` teachers annotate textual descriptions of processes), to learn automatically the patterns that define the process elements. Finally, we plan to perform a deeper evaluation of our techniques, possible including real-life data from our current projects.

Source	#gold	Public code by [16]					Our patterns				
		#pred	#ok	P	R	F_1	#pred	#ok	P	R	F_1
1-1_bicycle_manufacturing	26	30	9	30	35	32	33	21	**64**	**81**	**71**
1-2_computer_repair	19	13	6	46	32	37	18	12	**67**	**63**	**65**
2-1_sla_violation	90	100	42	**42**	47	44	128	53	41	**59**	**49**
3-1_2009-1_mc_finalice	15	19	9	47	60	53	17	10	**59**	**67**	**63**
3-2_2009-2_conduct_directions	13	8	7	**88**	54	**67**	11	8	73	**62**	**67**
3-6_2010-1_claims_notification	22	23	11	48	50	49	25	13	**52**	**59**	**55**
4-1_intaker_workflow	65	101	40	40	62	48	91	43	**47**	**66**	**55**
5-1_active_vos_tutorial	13	14	6	43	46	44	16	11	**69**	**85**	**76**
6-1_acme	40	48	16	33	40	36	68	27	**40**	**68**	**50**
7-1_calling_leads	12	16	7	44	58	50	16	9	**56**	**75**	**64**
8-1_hr_process_simple	11	5	5	**100**	45	**62**	10	6	60	**55**	57
9-2_exercise_2	14	12	6	50	43	46	17	13	**76**	**93**	**84**
10-2_process_b3	27	25	14	**56**	**52**	**54**	29	14	48	**52**	50
Total	367	414	178	43	49	46	479	240	**50**	**65**	**57**
1081511532_rev3	18	18	8	44	**44**	44	17	8	**47**	**44**	**46**
1120589054_rev4	20	27	11	41	**55**	47	16	10	**63**	50	**56**
1364308140_rev4	31	16	13	**81**	42	55	22	15	68	**48**	**57**
20818304_rev1	24	12	10	**83**	42	56	24	15	63	**63**	**63**
784358570_rev2	37	31	13	42	35	38	37	18	**49**	**49**	**49**
Total	130	104	55	53	42	47	116	66	**57**	**51**	**54**
datacollection_1	132	93	69	**74**	52	61	127	94	**74**	**71**	**73**
datacollection_2	51	51	45	88	88	88	50	46	**92**	**90**	**91**
datacollection_3	201	172	123	**72**	**61**	**66**	197	120	61	60	60
Total	384	316	237	**75**	62	68	374	260	70	**68**	**69**
TOTAL	881	834	470	56	53	55	969	566	**58**	**64**	**61**

Table 2. Results of performed experiments with respect to [16]. The information is the same than in Table 1, but with counts correspond to both activities and relations.

Acknowledgments This work has been supported by MINECO and FEDER funds under grant TIN2017-86727-C2-1-R, and by the Ecuadorian National Secretary of Higher Education, Science and Technology (SENESCYT).

References

1. João Carlos de A. R. Gonçalves, Flávia Maria Santoro, and Fernanda Araujo Baião. Business process mining from group stories. In *Proceedings of the 13th International Conference on Computers Supported Cooperative Work in Design, CSCWD 2009, April 22-24, 2009, Santiago, Chile*, pages 161–166. IEEE, 2009.
2. 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.
3. Mauro Dragoni, Serena Villata, Williams Rizzi, and Guido Governatori. Combining natural language processing approaches for rule extraction from legal documents.

In *AI Approaches to the Complexity of Legal Systems*, pages 287–300, Cham, 2018. Springer International Publishing.
4. Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. Transition-based dependeny parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, (ACL-IJCNLP 2015)*, Beijing, China, 2015.
5. Christiane Fellbaum. *WordNet. An Electronic Lexical Database*. Language, Speech, and Communication. The MIT Press, 1998.
6. Fabian Friedrich. Automated generation of business process models from natural language input. *School of Business and Economics. Humboldt-Universität zu Berli*, 2010.
7. 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.
8. Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI*, 2013.
9. Roger Levy and Galen Andrew. Tregex and tsurgeon: tools for querying and manipulating tree data structures. In *LREC*, pages 2231–2234. Citeseer, 2006.
10. Hugo A. López, Søren Debois, Thomas T. Hildebrandt, and Morten Marquard. The process highlighter: From texts to declarative processes and back. In *Proceedings of the Dissertation Award, Demonstration, and Industrial Track at BPM 2018 co-located with 16th International Conference on Business Process Management (BPM 2018), Sydney, Australia, September 9-14, 2018*, pages 66–70, 2018.
11. 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.
12. Josep Sànchez-Ferreres, Andrea Burattin, Josep Carmona, Marco Montali, and Lluís Padró. Formal reasoning on natural language descriptions of processes. In *Int. Conference on Business Process Management*, pages 86–101. Springer, 2019.
13. 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.
14. F Semmelrodt. Modellierung klinischer prozesse und compliance regeln mittels BPMN 2.0 und eCRG. Master's thesis, University of Ulm, 2013.
15. 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.
16. Han van der Aa, Claudio Di Ciccio, Henrik Leopold, and Hajo A Reijers. Extracting declarative process models from natural language. In *International Conference on Advanced Information Systems Engineering*, pages 365–382. Springer, 2019.
17. 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.
18. Wil M.P. van der Aalst. *Process Mining*. Springer, second edition, 2016.
19. Karolin Winter and Stefanie Rinderle-Ma. Detecting constraints and their relations from regulatory documents using nlp techniques. In *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*, pages 261–278, Cham, 2018. Springer International Publishing.