Extracting Declarative Process Models from Natural Language

Han van der Aa[1], Claudio Di Ciccio[2], Henrik Leopold[3,4], Hajo A. Reijers[5]

[1] Department of Computer Science, Humboldt-Universität zu Berlin, Germany
[2] Institute for Information Business, Vienna University of Economics and Business, Austria
[3] Kühne Logistics University, Hamburg, Germany
[4] Hasso Plattner Institute, University of Potsdam, Germany
[5] Department of Information and Computing Sciences, Utrecht University, The Netherlands

Abstract. Process models are an important means to capture information on organizational operations and often represent the starting point for process analysis and improvement. Since the manual elicitation and creation of process models is a time-intensive endeavor, a variety of techniques have been developed that automatically derive process models from textual process descriptions. However, these techniques, so far, only focus on the extraction of traditional, imperative process models. The extraction of declarative process models, which allow to effectively capture complex process behavior in a compact fashion, has not been addressed. In this paper we close this gap by presenting the first automated approach for the extraction of declarative process models from natural language. To achieve this, we developed tailored Natural Language Processing techniques that identify activities and their inter-relations from textual constraint descriptions. A quantitative evaluation shows that our approach is able to generate constraints that closely resemble those established by humans. Therefore, our approach provides automated support for an otherwise tedious and complex manual endeavor.

1 Introduction

In many business processes the activities are executed in a typical order, without considerable deviations or exceptions. These business processes can be well captured using traditional, imperative business process modeling notations, such as the Business Process Model and Notation (BPMN), because the resulting model specifies which execution orders are allowed. However, other business processes, often referred to as *knowledge-intensive business processes*, are more complex. Their execution orders cannot be fully specified in advance [10]. Such processes are much better captured using declarative process models, since these do not depend on an explicit definition of the allowed behavior [4]. Instead, they use constraints to define the boundaries of the permissible process behavior. This enables declarative models to represent complex processes in a compact way. What both types of process models have in common is their general usefulness for effectively conveying how business processes are executed and for analyzing business processes [13].

Nevertheless, the elicitation and creation of process models does not come without problems. First, many process actors and domain experts lack the knowledge necessary to establish process models themselves [15, 30]. Second, the elicitation of process models represents a highly time-consuming task [17]. As a result of both these issues, a wealth of process-related

Table 1: Description and notation of considered DECLARE constraints

Type	Constraint	Explanation	Examples			
Existence	INIT(a)	a is the *first* to occur	✓ *acc*	✓ *abac*	× *cc*	×*bac*
	END(a)	a is the *last* to occur	✓ *bca*	✓ *baca*	× *bc*	×*bac*
Relation	RESPONSE(a,b)	If a occurs, then b occurs after a	✓ *caacb*	✓ *bcc*	× *caac*	×*bacc*
	PRECEDENCE(a,b)	b occurs only if preceded by a	✓ *cacbb*	✓ *acc*	× *ccbb*	×*bacc*
Mutual rel.	SUCCESSION(a,b)	a occurs if and only if b occurs after a	✓ *cacbb*	✓ *accb*	× *bac*	×*bcca*
Negative rel.	NOTSUCCESSION(a,b)	a never occurs before b	✓ *bbcaa*	✓ *cbbca*	× *aacbb*	×*abb*

information is often captured in more accessible representation formats, such as textual documents [3, 30]. Recognizing the widespread use and relevance of textual documents to capture process information, a variety of techniques have been developed that automatically extract process models from natural language (e.g. [15, 16, 31]). However, these existing techniques focus solely on *imperative* process models. Therefore, there is currently no technique available that can extract *declarative* process models from natural language text.

To overcome this gap, we use this paper to introduce the first approach for the automatic extraction of declarative process models from natural language. To achieve this, we developed tailored Natural Language Processing (NLP) techniques that identify activities and their inter-relations from textual constraint descriptions. By considering the semantics of these extracted components, we subsequently generate declarative constraints aiming to capture the logic defined in the textual description. A quantitative evaluation shows that our approach is able to generate constraints that show a high resemblance to those established manually. As such, our automated approach supports an otherwise tedious and complex manual endeavor.

The remainder of this paper is organized as follows. Section 2 introduces declarative process modeling and the challenges associated with the automatic extraction of declarative constraints from text. Section 3 presents our proposed approach. Section 4 describes a quantitative evaluation in which we demonstrate the usefulness of our approach. Section 5 elaborates on related work before Section 6 provides a conclusion and discusses directions for future research.

2 Problem Illustration

This section provides essential background regarding declarative process modeling (Section 2.1) and introduces core challenges associated with the extraction of declarative models from natural language (Section 2.2).

2.1 Declarative Modeling

A declarative model represents the behavior of a process by means of *constraints*, i.e., temporal rules that specify the conditions under which activities can or cannot be executed. For the purposes of this paper, we focus on DECLARE, one of the most well-established declarative process modeling languages to date [4].

DECLARE provides a standard library of templates (*repertoire* [9]), which are constraints parametrized over activities. In this paper, we consider five of the most commonly occurring templates [14], depicted in Table 1, focusing in particular on the ones that are topmost in the

Table 2: Exemplary natural language descriptions and DECLARE constraints

ID	Description	Constraint
S_0	The process starts when a claim is received.	INIT(*receive claim*)
S_1	A claim must be created before it can be approved.	PREC.(*create claim*,*approve claim*)
S_2	Before a claim is approved, it must be created.	PREC.(*create claim*,*approve claim*)
S_3	Creation of a claim should precede its approval.	PREC.(*create claim*,*approve claim*)
S_4	If a claim is approved, then it must have been created first.	PREC.(*create claim*,*approve claim*)
S_5	A claim must be created before it can be approved or rejected.	PREC.(*create claim*,*approve claim*) PREC.(*create claim*,*reject claim*)
S_6	When an order is shipped, an invoice can be sent.	PREC.(*ship order*,*send invoice*)
S_7	When an order is shipped, an invoice must be sent.	RESP.(*ship order*,*send invoice*)
S_8	When an order is shipped, an invoice must be sent first.	PREC.(*send invoice*,*ship order*)
S_9	An invoice cannot be paid before it is received.	NOTSCSN.(*pay invoice*,*receive invoice*)
S_{10}	An invoice can be paid before it is received.	SCSN.(*pay invoice*,*receive invoice*)
S_{11}	The process ends when the invoice has been paid	END(*pay invoice*)

subsumption hierarchy of DECLARE constraints [9]. *Existence* constraints are *unary* constraints, i.e., predicating on single tasks. For instance, INIT(a) and END(b) establish that all process instances must begin with activity a and terminate with b, respectively. *Relation* constraints are *binary* and relate the execution of an activity to the occurrence of another one in the same instance (i.e., in one particular execution of the process). For example, RESPONSE(a,b) requires that if a is carried out, then b must be eventually performed. PRECEDENCE(a,b) imposes that b cannot be executed if a has not occurred earlier in the process instance. SUCCESSION(a,b) is a so called *mutual relation* constraint that expresses the joint conditions of RESPONSE and PRECEDENCE over a and b. Aside from these five templates, we also consider their negative counterparts. For instance, NOTSUCCESSION(a,b) is a *negative relation* constraint which states that if a is executed, then b cannot be carried out any longer.

2.2 Extraction Challenges

To extract declarative constraints from natural language, several challenges must be addressed. These challenges range from identifying sentences that contain constraints to extracting specific constraints from individual sentences. In this work, we primarily focus on challenges related to the latter type. We use the exemplary constraint descriptions from Table 2 as a means to illustrate six core challenges addressed in this work. Note that the majority of these challenges are caused by the flexibility of natural language, which allows the same DECLARE constraint to be described by a wide variety of sentences.

C1: ***Synonymous terms and phrases.*** Synonyms represents terms that have an equivalent meaning. For instance, the synonymous verbs "*end*", "*complete*", and "*finalize*", can all be used to indicate the final step of a process, such as seen for S_{11}. On a higher level of granularity, entire phrases or sentences may be used to describe the same concept. This requires an extraction approach to analyze and recognize the meaning of a sentence, rather than simply relying on a number of specific terms.

C2: ***Description order.*** Linguistic variability also manifests itself in the order in which a sentence describes the constraint's components. Some descriptions, such as S_1 and S_3, use a chronological order, first describing the *create claim* task, whereas S_2 and S_4 use the reverse order. To extract the appropriate constraints from these descriptions, it is therefore key to recognize the semantic relations that exist among the described actions. Crucially, this relation can depend on small textual cues, such as the inclusion of the word "*first*" at the end of sentence S_8. In this case, a single term fully reverses the semantics of the constraint in comparison to S_7.

C3: ***Noun-based actions.*** Identifying and extracting the actions contained in a textual description represents a core task of process model extraction. State-of-the-art approaches (cf. [15]) that address this task generally assume that activities can be identified by considering verbs in sentences, i.e., *verb-based* activity descriptions such as "*a claim must be created*" in S_1. However, in the case of declarative constraint descriptions, this assumption is often violated. They also describe activities using *noun-based* forms, such as "*creation of a claim*" in S_3. This presents a considerable challenge, since this means that existing extraction techniques cannot be applied in our context.

C4: ***Constraint restrictiveness.*** The differences between the binary constraints Response(*a*,*b*), Precedence(*a*,*b*), and Succession(*a*,*b*) are considerable from a semantic perspective. However, the differences in their textual descriptions can be minor. Compare, for instance, sentences S_6 and S_7 from Table 2. The term "*can*" in S_6 indicates an optional follow-up to "*ship order*," i.e., Precedence(*ship order*,*send invoice*). By contrast, the term "*must*" in S_7 indicates that *send invoice* has to occur, resulting in a Response constraint. To be able to appropriately differentiate among these constraint types, an extraction approach should identify terms that indicate how restrictive a constraint is meant to be.

C5: ***Negation.*** The ability to recognize negation in constraint descriptions is crucial, because the presence of a single negating term can reverse the semantics of the expression. This can, for example, be observed in description S_9, in which the use of "*cannot*" completely changes the meaning of the constraint in comparison to S_{10}.

C6: ***Multi-constraint descriptions.*** Single sentences may describe more than one declarative constraint. This commonly manifests itself through the use of *coordinating conjunctions*, indicated by terms such as "*and*" and "*or*". These terms can be used to link semantic components of a description that can indicate additional constraints. For example, consider the description S_5. This description is identical to S_1, aside from the conjunction at the end, which reveals that the Precedence constraint should be applied to both the approval and rejection of a claim.

Next, we describe our extraction approach, which aims to overcome these challenges.

3 Extraction Approach

Figure 1 provides an overview of the three-step approach we propose to extract declarative constraints from textual descriptions. First, linguistic processing is performed to extract *semantic components* from a constraint description. Those components are specifically targeted at the extraction of declarative constraints, e.g., the terms that enable differentiation among the various constraint types. By building on general-purpose NLP techniques, our approach is able to deal with a broad variety of linguistic patterns used in constraints (challenges C1

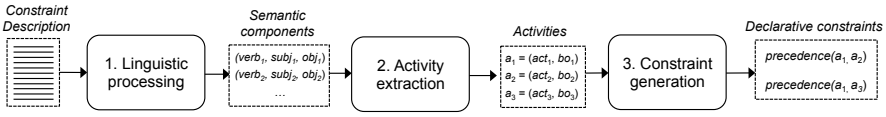

Fig. 1: Overview of the extraction approach

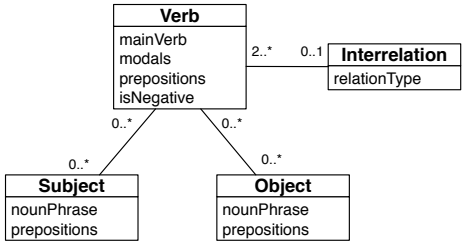

Fig. 2: Main semantic components extracted in the linguistic-processing step

and C2). Second, our approach analyzes the semantic components in order to identify the activities named in the description. A main novelty in this step is the explicit consideration of noun-based activities, which addresses challenge C3. Finally, in the third step, our approach generates constraints based on identified activities and other semantic information. This step particularly focuses on the identification of subtle semantic differences, which are crucial for the proper differentiation among constraint templates (challenge C4) and negation (C5). The final outcome consists of one or more declarative constraints (challenge C6).

3.1 Step 1: Linguistic Processing

In the first step of our approach, we employ widely-used NLP techniques to identify semantic components that are of specific importance to the extraction of declarative constraints. Figure 2 provides an overview of those components, which are identified as follows.

Verbs. Verbs convey *actions*, *occurrences*, or *states* in natural language texts, such as "*to create*", "*to occur*", and "*to be*". So-called *part-of-speech taggers* are well-suited to identify them. They assign a tag indicating a word category to each word in a natural language text [20]. An example of such a tagger is the widely employed *Stanford parser* [21]. To illustrate the usefulness of taggers for the identification of verbs, consider the tags assigned by the Stanford Parser on the words of sentence S_5:

> "A/DT manager/NN must/MD create/**VB** a/DT claim/NN before/IN it/PRP can/MD be/**VB** accepted/**VBN** or/CC rejected/**VBN** ./."

Since the explanation of all tags goes beyond the scope of this paper, we focus here on the tags that indicate verbs. In particular, the /VB tag indicates the *base form* of a verb and the /VBN tags indicate the *past participle* of a verb. Therefore, by considering part-of-speech tags we can extract three main verbs from the description: "*create*", "*accepted*", and "*rejected*".

Subjects and Objects. The *subject* of a verb indicates the entity that performs the action. In a process context, the subject typically corresponds to the *actor* executing an activity (e.g., a

manager who creates a claim). Similarly, the *object* of a verb indicates the entity that is acted upon. In a process context, the object of a verb often corresponds to the *business object* that is affected by an activity, e.g. the *claim* business object being created.

For the identification of subjects and objects, we employ NLP techniques that compute *dependency grammars* for natural language texts. Those grammars capture the grammatical relationships among the words in a sentence using dependency relations. Typed dependency relations, such as the *Stanford relations* [7], describe grammatical relations of a certain type that exist between two words. For instance, the relation *nsubj*(*create*, *manager*) denotes that "*manager*" is the subject performing the verb "*create*". Table 3 provides an overview of the dependency relations obtained for constraint S_5.

The most important Stanford relation used to extract subjects is the aforementioned *nsubj* relation. To extract objects, we consider two relations: *dobj* and *nsubjpass*. The *dobj* relation indicates the direct object of a verb, as in the relation *dobj*(*create*, *claim*) from Table 3. The *nsubjpass* relation refers to the *synthetic subject* in a passive clause. However, from a process perspective, this synthetic subject actually fulfils the same role as an object. For example, the phrase "*A claim is created*" contains the relation *nsubjpass*(*create*, *claim*), which is equivalent to the *dobj* relation for active phrases.

Specifiers. We augment the extracted verbs, subjects, and objects with various specifiers. In particular, we focus on the extraction of modal verbs, negations, and prepositions. By employing the previously introduced grammatical dependencies, we identify these as follows:

Modal verbs. Modal verbs are auxiliary verbs that are used to indicate if something is certain, probable, or possible (or not). The principal modal verbs in English are: can/could, may/might, must, will/would and shall/should. Modal verbs play a key role in our approach, because they are an important means to distinguish among various declarative constraints, such as Response and Succession (challenge C4). Therefore, we explicitly extract modal verbs and associate them with the related main verbs. For example, in sentence S_5 we use the *aux* relation to relate "*must*" to the verb "*created*".

Negation. As indicated in challenge C5, the recognition of negations in constraint descriptions is important because failing to do so can lead to the identification of a constraint that is completely the opposite of what is intended.
To identify negated verbs, we use the *neg* dependency relation. For example, for constraint description S_9, "*An invoice cannot be paid before it is received*", the relation

Table 3: Grammatical dependencies for constraint S_5

Relation	**Meaning**	**Relation**	**Meaning**
det(*manager, A*)	determinant	*nsubjpass*(*accepted, it*)	passive subject
nsubj(*create, manager*)	subject	*aux*(*accepted, can*)	auxiliary verb
aux(*create, must*)	auxiliary verb	*auxpass*(*accepted, be*)	passive auxiliary
det(*claim, a*)	determinant	*advcl*(*create, accepted*)	adverbial clause
dobj(*create, claim*)	object	*cc*(*accepted, or*)	coordination
mark(*accepted, before*)	marker	*conj*(*accepted, rejected*)	conjunct

neg(*paid*, *not*) indicates the negation. By extracting this information, we are able to properly identify negative constraints.

Prepositions. Prepositions are terms used to specify a relationship between a noun and other parts of a sentence. In the context of constraint descriptions, prepositions are commonly used to indicate ordering relations that exist between activities. For instance, in constraint description S_5, the term "*before*" is used to describe a relation between "*a claim is approved*" and "*must create a claim*." In particular, it denotes that the latter part must occur *before* a claim's approval. Thus, we augment nouns with information on their prepositions. To that end, we primarily use the *mark* and *advmod* Stanford dependencies, such as the *mark*(*accepted*, *before*) included in Table 3.

Interrelations. Finally, our approach extracts information regarding two types of interrelations that can exist among verbs in a constraint description: adverbial clauses and conjunctions.

Adverbial clauses. Adverbial clauses are dependent clauses that modify other entities in a text. In the case of constraint descriptions, these can be used to describe relations that exist among verbs. For example, Table 3 shows the relation *advcl(create, accepted)*, which indicates that the phrase containing the former term ("*A manager must create a claim*" represents a specifier for the phrase containing the latter term "*it can be accepted*"). In particular, it specifies a temporal relation between the two phrases. In the remainder, we will denote a relation between two verbs v_1 and v_2 as *rel*(v_1,v_2).

Coordinating conjunctions. Conjunctions indicate relations between two entities, which are important when handling multi-constraint descriptions (C6). Common conjunctions are indicated by "*and*" and "*or*". For example, in constraint S_5, the term "*or*" indicates that there is a relation between "*accepted*" and "*rejected*". By identifying this relation, we recognize that "*rejected*" is in the same adverbial relations as "*accepted*", i.e., *rel*(*created*, *rejected*) is derived from the adverbial relation *rel*(*created*, *accepted*) and the conjunctive relation *conj*(*accepted*,*rejected*).

After this linguistic processing step, which covers all the linguistic components just discussed, we use the extracted components to identify the activities contained in a constraint description.

3.2 Step 2: Activity Extraction

Activities denote core constructs in any process description, thus also for constraints in declarative process modeling. An activity in a process description generally refers to an *action* that is performed on some *business object*, optionally by a specified *actor*. For example, in constraint S_1, we observe the activities "*create claim*" and "*approve claim*". As indicated by challenge C3, to properly extract activities, we need to consider both *verb-based* and *noun-based* activities.

Verb-based activities. For the extraction of verb-based activities, we employ the state-of-the-art technique used by Friedrich et al. [15], which utilizes the semantic components identified in the previous step. The verb corresponds to the action, the verb's subject to the actor, and the verb's object to the activity's business object. For example, in Section 3.1, we analysed the verb "*created*", having "*manager*" as a subject and "*a claim*" as the object. This translates to the *create claim* activity, performed by a manager.

Noun-based activities. A noun-based activity is a noun phrase in a constraint description that corresponds to a process activity, such as "*creation of a claim*" and "*its approval*" in constraint S_3. The main concern regarding their extraction is the ability to recognize the situations in which they occur. Therefore, we focus on the identification of verbs that describe the flow of a process, rather than activities. In sentence S_2, for instance, the verb "*precedes*" describes a flow relation between its subject, "*creation of the claim*", and its object, "*its approval*". To recognize such cases, we decided to use an established set of so-called *temporal verbs* [5, p.39]. This taxonomy includes verbs such as: *to start*, *to proceed*, *to end*, *to finish*, *to follow*, *to happen*. We augment this list with some of their synonyms, such as *to occur*, *to complete*, and *to finish*. When our approach recognizes that an extracted verb corresponds to a temporal verb from the aforementioned list, the approach generates activities based on the subject and object of the verb (if any). For example, the approach would extract the *claim creation claim approval* activities from sentence S_3. Furthermore, for noun-based activities stemming from verbs with both a subject and an object, we extract a semantic relation between those components. For example, in S_3, we recognize that a relation exists between its activities, i.e., the approach extracts *rel*(*claim creation*, *claim approval*).

3.3 Step 3: Constraint Generation

In the final step of our approach, we generate declarative constraints based on the extracted activities and their semantic interrelations. Algorithm 1 shows the main flow of this generation step. The algorithm takes as input a relation *rel*(*a*,*b*) that exist between two extracted activities. These relations result from the verb interrelations, as identified in Section 3.1, and from the noun-based activity extraction, described in Section 3.2.

Notably, our approach can handle multi-constraint descriptions (challenge C6) because we apply the algorithm to all relations *rel*(*a*,*b*) identified in a description. For instance, we identify two relations in sentence S_5, namely *rel*(*create claim*, *accept claim*) and *rel*(*create claim*,*reject claim*). As a result, our approach generates two declarative constraints out of a single sentence.

In the remainder of this section, we will describe the core aspects of our algorithm. We examine (i) the generation of unary constraints (lines 3–8), (ii) the determination of the temporal order in binary constraints (lines 9–11), (iii) the differentiation among precedence, response, and succession constraints (lines 12–18), and (iv) the identification of negated constraints (lines 19–20).

Unary constraint generation. Unary relations correspond to the Init(*a*) and End(*a*) constraint templates, which identify the start and end points in a process. To identify those types of constraints, we recognize that their constraint descriptions describe activities from the viewpoint of the process itself, rather than from the viewpoint of an actor in the actor. Consider, for instance, the sentence S_0: "*The process starts when a claim is received.*" In this description, the subject, i.e., the actor that performs the *start* activity, is the *process*, rather than, for instance, a *department* or a *manager*.

To generate the unary Init and End constraints, we identify the involvement of such *process subjects*. We achieve this by checking the actor of an activity against a set of process-related terms (employed by Friedrich et al. [15]), which includes terms such as "*process*", "*instance*", and "*case*". When such a process subject occurs as the *actor* in a description, we establish that the description contains a sort of *meta-action* related to the process flow (lines

Algorithm 1 Declarative constraint generation

```
1:  function GenerateConstraint
2:      Input: rel(a, b)                                   ▷ Relation between actions a and b
3:      metaAct ← ⊥; otherAct ← ⊥;                         ▷ Placeholders for unary constraints
4:      if isMetaAction(a) then metaAct ← a, otherAct ← b;  ▷ Check for meta-activities
5:      if isMetaAction(b) then metaAct ← b, otherAct ← a;
6:      if metaAct ≠ ⊥ then                                ▷ A unary constraint should be generated
7:          if hasStartVerb(metaAct) then return Init(otherAct)
8:          if hasEndVerb(metaAct) then return End(otherAct)
9:      if reversedDirection(rel(a, b)) then               ▷ Check temporal order of binary constraint
10:         act1 ← b; act2 ← a
11:     else act1 ← a; act2 ← b
12:     template ← ⊥
13:     if isMandatory(act1) ∧ ¬isMandatory(act2) then     ▷ Determine binary constraint template
14:         constraint ← Precedence(act1,act2)
15:     if ¬isMandatory(act1) ∧ isMandatory(act2) then
16:         constraint ← Response(act1,act2)
17:     if isMandatory(act1) ∧ isMandatory(act2) then
18:         constraint ← Succession(act1,act2)
19:     if hasNegation(act1, act2) then                    ▷ Check if template should be negated
20:         constraint ← NotSuccession(act1,act2)
21:     return constraint
```

3–5). If a meta-action is detected, we then determine if the constraint describes Init or End by comparing the verb to synonyms of "*start*" and "*end*" in WordNet [25] (lines 7–8). As a result, our approach generates the constraints Init(*receive claim*) for description S_0.

Binary constraint direction. A crucial aspect for the correct extraction of binary constraints, such as Response(*a*,*b*) and Precedence(*a*,*b*), is to recognize which activity in a description corresponds to *a* and which to *b*. Such a recognition is not trivial because, considering two activities, the corresponding propositions can be put in different points of the sentence though keeping the same temporal order. For instance, description S_1, "*A claim must be created before it can be approved*", describes activity *a* (i.e., *create claim*) prior to *b* (*approve claim*). In S_2, propositions are switched but the same constraint is described: "*Before a claim is approved, it must be created*". We shall name the switch of propositions with respect to the temporal order as *reversed direction*. To properly recognize constraint directions, we consider three aspects that can be present in textual constraint descriptions: *temporal prepositions*, *temporal verbs*, and *tenses*.

Temporal prepositions. The prepositions that we extracted in step 1 can represent valuable indicators. In particular, prepositions can correspond to a subset of *temporal prepositions*, which indicate different temporal relations that exist between components of a sentence, in our context between activities. For this, we build on an established classification of temporal prepositions.[1] In our approach, we consider the *preceding* and *following* subclasses. The former contains prepositions such as "*ahead of*", "*before*", and "*prior to*", whereas the latter includes "*after*", "*beyond*", and "*subsequent to*".

[1] See: www.clres.com/db/classes/ClassTemporal.php

To illustrate the use of temporal prepositions to our aim, consider again sentence S_2. In step 1 of our approach, it identifies "*before*" as a preposition of "*accepted*" and an interrelation between "*accepted*" and "*created*". Given that the preposition "*before*" belongs to the class of *preceding* prepositions, our approach is able to recognize that "*accepted*" should be preceded by its related verbs, i.e. by "*create*". Therefore, it accurately identifies that *a* corresponds to *create claim* and *b* to *accept claim*.

Temporal verbs and tenses. For descriptions involving noun-based activities, the *process verbs* identified in Section 3.2 can provide valuable indicators of reversed directions. Consider, for example, constraint description S_3, "*Creation of a claim should precede its approval.*" In this text, the verb "*precede*" corresponds to a temporal verb from the taxonomy described in Section 3.2. This verb specifies that "*claim creation*" should occur before the "*claim approval*". Conversely, if the text had used the verb "*follow*" rather than "*precede*", our approach would have identified the reversed direction.
The way in which temporal verbs are used plays a crucial role here. Most importantly, we differentiate between active forms, e.g. "*precede*," and passive forms, e.g. "*is preceded by*." As an illustration, consider a sentence S'_3 that reads "*Creation of a claim is preceded by its approval.*" The replacement of an active verb (as seen in S_3) with its infinitive form yields exactly the opposite temporal order between the two activities. The identification and selection of the proper one is included in lines 9–11. Once it has been determined, we finally need to identify the correct constraint *template*.

Binary constraint templates. For the binary constraint templates Precedence, Succession, and Response, a key aspect of their accurate elicitation is to recognize what elements distinguishes them. All three constraint templates denote a temporal order between activities *a* and *b*. However, the key difference is in the restrictions that the constraints impose. Precedence(*a*,*b*) states that *b* can only occur if *a* has already been performed. Response(*a*,*b*) indicates the opposite, namely that if *a* has occurred, then *b* must follow at some stage. Finally, Succession(*a*,*b*) lies at the intersection of the previous constraints, and enforces that neither *a* nor *b* can occur independently from each other.

To identify these different templates in a textual constraint description, we identify indicators of restrictiveness in the text. In particular, we set out to determine whether the tasks involved in a constraint are mandatory or optional (as indicated in lines 12–18). Specifically, we consider the use of modal verbs, as discussed in Section 3.1. The modal verbs *can*, *could*, *may*, and *might* indicate that something is optional, whereas *must*, *will*, *would*, *shall*, and *should* generally specify that an activity is mandatory. Consider, for instance, the difference between the constraint descriptions S_6 and S_7 from the running example. These descriptions differ because S_6 states that "*an invoice can be sent*" while S_7 states that "*an invoice must be sent*". For these constraints, the replacement of the *optional* modal "*can*" with the *mandatory* modal "*must*" means that S_7 describes a Response constraint, rather than Precedence. We use the presence of modals as follows to determine the template of the constraint.

Precedence(*a*,*b*). If *a* has an associated mandatory modal, whereas *b* is associated with an optional modal, then this yields the constraint Precedence(*a*,*b*) because *a* is a mandatory prerequisite for the optional follow-up *b*. This is, for instance, seen for constraint S_1, which states that "*A claim must* [mandatory] *be created, before it can* [optional] *be accepted.*"

Table 4: Overview of the test collection

Source	Init	End	Precedence	Response	Succession	Negation	Multi-cons.	Total
Declare [26]	1	0	0	3	0	0	0	4
DCR graphs [18, 32]	1	1	12	7	2	2	6	22
Declare mining [11, 24]	2	0	4	4	3	0	1	12
General descriptions [29]	7	3	32	5	2	2	20	49
Syntax variation	0	0	15	0	0	0	0	16
Full collection	11	4	63	19	7	4	27	103

Response(*a*,*b*). If *b* is accompanied by a mandatory modal whereas *a* either has an optional or no modal, then the approach recognizes a Response(*a*,*b*) constraint, such as seen for constraint S_7: "*When an order is shipped* [no modal]*, an invoice must* [mandatory] *be sent.*"

Succession(*a*,*b*). Finally, using the same reasoning, when both activities are accompanied by mandatory modal verbs, a Succession(*a*,*b*) constraint is generated.

Negation. Finally, we also consider the identification of negation in constraints (lines 19–20). In Section 3.1 we described how we identified negated verbs based on dependency relations. For these cases, we return a NotSuccession constraint, as it is the negative form of Succession, hence, a fortiori, of Response and Precedence. For example, for constraint description S_9 ("*An invoice cannot be paid before it is received*"), our approach recognizes that this is a negative constraint. As a result, our approach returns NotSuccession(*pay invoice*, *receive invoice*).

4 Evaluation

To demonstrate the capabilities of our extraction approach, we conduct a quantitative evaluation by comparing automatically extracted declarative constraints to a manually created gold standard. The evaluation goal is to learn how well the automated approach approximates constraints created manually. If our extraction approach can automatically generate definitions that closely resemble those created manually by experts, it seems fair to regard our approach as a viable and efficient alternative to an otherwise time-consuming and complex manual endeavor. The data collection and prototype used in this evaluation are both publicly available.[2]

4.1 Test Collection

To compose the test collection used for our evaluation, we obtained textual constraint descriptions from various industrial and academic sources. The sources can be divided into two sets: (1) sources that specifically describe declarative process models, e.g., works on Declare [26] and DCR graphs [18,32], and (2) sources that provide more general, i.e., not specifically declarative, process descriptions, stemming from Sanchez et al. [29]. From these sources, we derived the constraint descriptions that relate to the declarative constraint types included in our scope.

The final collection consists of 103 constraint descriptions, of which Table 4 present the main characteristics. All further details of the collection are publicly available at the

[2] See: https://github.com/hanvanderaa/declareextraction

aforementioned link. The descriptions included in the collection vary highly in terms of the syntax used to describe constraints. On the one hand, this is because the descriptions stem from different sources and, thus, from different authors. On the other hand, the syntactic differences were intentionally created. A researcher, who is independent of the team involved in the development of the proposed approach, established 15 variations of the constraint PRECEDENCE(*create claim*,*approve claim*), some of these variations are used as part of the running example in Table 2. Due to these factors, we believe that the test collection is well-suited to achieve a high external validity of the evaluation results.

4.2 Setup

To be able to compare the generated and manually extracted declarative constraints, we conceptualize the extraction of declarative constraints as a *template* or *slot filling* problem [20]. As a result, each declarative constraint consists of a number of slots, e.g., the constraint SUCCESSION(*a*,*b*), contains three slots: (1) a *template* slot, denoting that the template of the constraint is SUCCESSION, (2) a slot for task *a* (3) a slot for task *b*. When a constraint description contains multiple constraints, as seen for constraint S_5, the number of slots to be identified increases alongside the number of constraints. Given the nature of a slot filling problem, we can quantify the performance of our approach by computing the well-known *precision* and *recall* metrics. Using A to denote the set of slots filled by our approach and G for the slots filled in the gold standard, precision reflects the fraction of slots that our automated approach filled correctly according to the gold standard ($|A \cap G|/|A|$). Recall represents the fraction of slots filled in the gold standard that were also correctly filled by our approach ($|A \cap G|/|G|$). Furthermore, we report the F_1-score as the harmonic mean of precision and recall.

Table 5: Overview of the evaluation results

Template	**Cases**	**Precision**	**Recall**	F_1**-score**
INIT	11	0.75	0.82	0.78
END	4	0.88	0.88	0.88
PRECEDENCE	63	0.78	0.71	0.74
RESPONSE	19	0.80	0.77	0.75
SUCCESSION	7	0.68	0.68	0.68
Negation	4	1.00	1.00	1.00
Single-constraint	74	0.79	0.82	0.80
Multi-constraint	27	0.74	0.61	0.67
Type recognition	132	0.86	0.86	0.86
Overall	103	0.77	0.72	0.74

4.3 Results

Table 5 provides an overview of the most important evaluation results. The table shows that our approach achieves an overall precision of 0.77 and a recall of 0.72, yielding an F_1-score of 0.74. In the following, we investigate the results in detail by considering how well our approach deals with the various challenges posed in Section 2.

C1, C2: Synonymous terms and description order. Challenges C1 and C2 both relate to the variety of natural language descriptions that can be used to describe a constraint. The ability of our approach to deal with these challenges is, among others, reflected in the results achieved for the 15 constraints we introduced to test syntax variation. For those 15 descriptions, which all describe the same constraint, our approach achieves precision and recall scores of 0.88. This demonstrates that by building on NLP techniques such as parsers, our approach is able to handle linguistic variation well.

C3: Noun-based activities. As described in Section 3.2, we have developed a novel approach to extract activities from texts that also extracts noun-based activities. The value of this extraction approach can be determined by comparing our approach's results to those results that would be obtained if we just relied on existing techniques, i.e., if we just identified activities using the extraction approach from [15]. By employing that activity extraction technique, a precision of 0.77 and recall of 0.66 are achieved. By using our approach the recall thus increases from 0.66 to 0.72. This illustrates that the additional consideration of noun-based activities helps to improve performance.

C4: Constraint restrictiveness. As shown in Table 5, our approach identifies constraint templates correctly in over 86% of the cases. This means that it is well able to differentiate among the different levels of restrictiveness for constraints. An in-depth analysis of the results also reveals that the template identification errors are largely related to other problems of the approach. For instance, for cases when the activities are not extracted properly, also the constraint template is not identified correctly.

C5: Negation. As shown in Table 5, our approach successfully identifies the negation present in constraint descriptions and, furthermore, does not indicate negation when it is not present (i.e., no false negatives). Nevertheless, it should be taken into account that only 4 cases in the test collection contain negated constraints.

C6: Multi-constraint descriptions. Finally, we observe that the ability of our approach to deal with multi-constraint descriptions varies. In Table 5, it becomes clear that our approach achieves a lower recall for multi-constraint descriptions than for single-constraint ones (0.61 versus 0.82). However, we also observe that our approach is able to deal with descriptions containing multiple constraints of a single template rather well. Its performance for descriptions that pertain contain multiple templates, e.g. a Response and a Succession, is considerably lower. These descriptions are generally longer and, as a result, typically more problematic for the NLP techniques on which we build.

5 Related Work

Textual documents, such as work instructions and process descriptions, represent a valuable source of information in the context of business process management [3]. Their value has led to the recent development of a variety of analysis techniques that focus on process information contained in text, such as techniques that compare process models against textual descriptions [1, 29], as well as techniques that consider textual descriptions for process querying [22, 23], process matching [6, 33], and conformance checking [2]. Closely related to the goal of this paper are existing techniques that focus on the extraction of imperative process models from natural language texts [15, 16, 27], of which the technique by Friedrich et al. [15] can be regarded as the state-of-the-art in this context [27]. Finally, recent research has focused on the identification and extraction of rules from regulatory documents [12, 34].

Our proposed approach provides unprecedented contributions related to the extraction of *declarative* process models from text, specifically the identification of constraint order (Challenge C2), the extraction of noun-based activities (C3), the identification of constraint types (C4), and the handling of multi-constraint descriptions (C6). Though existing techniques are not suitable to fully address them, the integration of the aforementioned advancements [12, 34] draws plans for our future work, as discussed in the conclusion.

6 Conclusion

This paper presented the first approach for the automated extraction of declarative process models from textual descriptions. Our approach builds on and significantly extends existing NLP techniques in order to identify process-related information in constraint descriptions and transform these into declarative constraints. A quantitative evaluation demonstrates that our approach achieves a high accuracy when compared to manually established constraints. Therefore, our approach provides automated support for an otherwise complex and tedious manual task, making the declarative modeling paradigm more accessible to a wider audience.

In future work, we aim to extend our approach in several directions. First, we aim to enhance our technique towards a richer repertoire of templates, considering subsumed constraint types such as *alternate* and *chain* templates, as well as by including branched constraints [8]. Second, we intend to augment the control-flow based constraints with other perspectives by focusing on the extraction of data-based and temporal declarative constraints. Finally, we will go beyond the consideration of individual sentences in isolation. Among others, this will involve (i) the extraction of constraints that span multiple sentences, (ii) identifying which sentences actually contain constraints, and (iii) relating extracted constraints to each other. Natural language processing techniques developed in the contexts of requirements engineering [19, 28], imperative process model extraction [15], and semantic matching [1] may all help in this regard. As a result, this will enable the extraction of declarative models from full documents, such as process documentations and normative texts.

Acknowledgments. This work has received funding from the EU H2020 programme under MSCA-RISE agreement 645751 (RISE_BPM) and the Alexander von Humboldt Foundation.

References

1. Van der Aa, H., Leopold, H., Reijers, H.A.: Comparing textual descriptions to process models: The automatic detection of inconsistencies. Information Systems **64**, 447–460 (2017)
2. Van der Aa, H., Leopold, H., Reijers, H.A.: Checking process compliance against natural language specifications using behavioral spaces. Information Systems **78**, 83–95 (2018)
3. Van der Aa, H., Leopold, H., van de Weerd, I., Reijers, H.A.: Causes and consequences of fragmented process information: Insights from a case study. In: 23rd Americas Conference on Information Systems, AMCIS (2017)
4. Van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science - R&D **23**(2), 99–113 (2009)
5. Androutsopoulos, I.: Exploring Time, Tense and Aspect in Natural Language Database Interfaces, vol. 6. John Benjamins Publishing (2002)
6. Baier, T., Mendling, J.: Bridging abstraction layers in process mining by automated matching of events and activities. In: Business Process Management, pp. 17–32. Springer (2013)
7. De Marneffe, M.C., Manning, C.D.: The stanford typed dependencies representation. In: Workshop on Cross-Framework and Cross-Domain Parser Evaluation. pp. 1–8 (2008)
8. Di Ciccio, C., Maggi, F.M., Mendling, J.: Efficient discovery of Target-Branched Declare constraints. Information Systems **56**, 258–283 (March 2016)
9. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. Information Systems **64**, 425–446 (March 2017)
10. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-intensive processes: characteristics, requirements and analysis of contemporary approaches. Journal on Data Semantics **4**(1), 29–57 (2015)

11. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Trans. Manage. Inf. Syst. **5**(4), 24:1–24:37 (January 2015)
12. Dragoni, M., Villata, S., Rizzi, W., Governatori, G.: Combining natural language processing approaches for rule extraction from legal documents. In: AICOL. pp. 287–300. Springer (2017)
13. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)
14. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE. pp. 411–420. ACM (1999)
15. Friedrich, F., Mendling, J., Puhlmann, F.: Process model generation from natural language text. In: Advanced Information Systems Engineering. pp. 482–496. Springer (2011)
16. Gonçalves, J.C.d.A., Santoro, F.M., Baiao, F.A.: Business process mining from group stories. In: CSCWD. pp. 161–166. IEEE (2009)
17. Herbst, J., Karagiannis, D.: An inductive approach to the acquisition and adaptation of workflow models. In: IJCAI. vol. 99, pp. 52–57. Citeseer (1999)
18. Hildebrandt, T., Mukkamala, R.R., Slaats, T.: Designing a cross-organizational case management system using dynamic condition response graphs. In: EDOC. pp. 161–170 (2011)
19. Ilieva, M., Ormandjieva, O.: Automatic transition of natural language software requirements specification into formal presentation. In: NLDB. pp. 392–397. Springer (2005)
20. Jurafsky, D., Martin, J.H.: Speech & language processing. Pearson Education India (2000)
21. Klein, D., Manning, C.D.: Accurate unlexicalized parsing. In: ACL. pp. 423–430 (2003)
22. Leopold, H., van der Aa, H., Pittke, F., Raffel, M., Mendling, J., Reijers, H.A.: Searching textual and model-based process descriptions based on a unified data format. SoSym pp. 1–16 (2017)
23. Leopold, H., van der Aa, H., Reijers, H.A.: Identifying candidate tasks for robotic process automation in textual process descriptions. In: Enterprise, Business-Process and Information Systems Modeling, pp. 67–81. Springer (2018)
24. Maggi, F.M., Di Ciccio, C., Di Francescomarino, C., Kala, T.: Parallel algorithms for the automated discovery of declarative process models. Information Systems (2017)
25. Miller, G.A.: WordNet: a lexical database for english. Communications of the ACM **38**(11), 39–41 (1995)
26. Pesic, M., Van der Aalst, W.M.: A declarative approach for flexible business processes management. In: International conference on business process management. pp. 169–180. Springer (2006)
27. Riefer, M., Ternis, S.F., Thaler, T.: Mining process models from natural language text: A state-of-the-art analysis. In: MKWI. Universität Illmenau (2016)
28. Saint-Dizier, P.: Mining incoherent requirements in technical specifications. In: NLDB. pp. 71–83. Springer (2017)
29. Sànchez-Ferreres, J., van der Aa, H., Carmona, J., Padró, L.: Aligning textual and model-based process descriptions. Data & Knowledge Engineering (2018)
30. Selway, M., Grossmann, G., Mayer, W., Stumptner, M.: Formalising natural language specifications using a cognitive linguistic/configuration based approach. Information Systems **54**, 191–208 (2015)
31. Sinha, A., Paradkar, A.: Use cases to process specifications in business process modeling notation. In: IEEE International Conference on Web Services. pp. 473–480. IEEE (2010)
32. Slaats, T., Mukkamala, R.R., Hildebrandt, T., Marquard, M.: Exformatics declarative case management workflows as DCR graphs. In: BPM, pp. 339–354. Springer (2013)
33. Weidlich, M., Sheetrit, E., Branco, M.C., Gal, A.: Matching business process models using positional passage-based language models. In: ER. pp. 130–137. Springer (2013)
34. Winter, K., Rinderle-Ma, S.: Detecting constraints and their relations from regulatory documents using nlp techniques. In: OTM. pp. 261–278. Springer (2018)