Say It In Your Own Words: Defining Declarative Process Models Using Speech Recognition

Han van der Aa[1], Karl Johannes Balder[2], Fabrizio Maria Maggi[2], and Alexander Nolte[2,3]

[1] Humboldt-Universität zu Berlin, Germany
han.van.der.aa@hu-berlin.de
[2] University of Tartu, Estonia
{balder,f.m.maggi,alexander.nolte}@ut.ee
[3] Carnegie Mellon University, Pittsburgh, PA, USA

Abstract. In recent years declarative, constraint-based approaches have been proposed to model loosely-structured business processes, mediating between support and flexibility. A notable example is the Declare framework, equipped with a graphical declarative language whose semantics can be characterized with several logic-based formalisms. Up to now, the main problem hampering the use of Declare constraints in practice has been the difficulty with modeling constraints either with a formal notation, difficult to understand for users without a background in temporal logics, or with their graphical notation that has been proved to be unintuitive. In this work, we present and evaluate an analysis toolkit that tries to bypass this issue by providing the user with the possibility of modeling Declare constraints using his/her own way of expressing them. The toolkit contains a Declare modeler equipped with a speech recognition mechanism that takes as input a vocal statement from the user and converts it into the closest (set of) Declare constraint(s). The constraints that can be modeled with the tool cover the entire Multi-Perspective extension of Declare (MP-Declare) that can not only express control-flow properties, but also other perspectives like data and time. Even if we consider Declare as use case, the tool/evaluation presented in this paper can be considered, to the best of our knowledge, the first attempt to test the usability of speech recognition in business process modeling.

1 Introduction

Process models are an important means to capture information on organizational processes, serving as a basis for communication and often as the starting point for analysis and improvement [10]. For processes that are relatively structured, *imperative* process modeling notations, such as the Business Process Model and Notation (BPMN), are most commonly employed. However, other processes, in particular knowledge-intensive ones, are more flexible and, therefore, less structured. An important characteristic of such processes is that it is typically infeasible to specify the entire spectrum of allowed execution orders in advance [9], which severely limits the applicability of the imperative process modeling paradigm. Instead, such processes are better captured using *declarative* process models defined in process modeling languages like DECLARE whose semantics can be characterized with several logic-based formalisms. These models, indeed, do not require an explicit definition of all allowed execution orders, but rather use constraints to define the boundaries of the permissible process behavior [8].

Although their benefits are apparent, establishing declarative process models is known to be difficult, especially for domain experts that generally lack expertise in temporal logics, and in most of the cases find the graphical notation of DECLARE constraints unintuitive. Due to these barriers to declarative model creation, a preliminary approach has been presented in [1] that automatically extracts declarative process models from natural language texts, similarly to what many works focusing on the generation of imperative process models from natural language descriptions have largely investigated (cf., [2, 11, 18, 20]).

In this work, we go beyond the extraction of DECLARE constraints from natural language descriptions, and present and evaluate an interactive approach that takes as input a vocal statement from the user and employs speech recognition to convert it into the closest (set of) DECLARE constraint(s). The approach has been integrated in a declarative modeling and analysis toolkit. With this tool, the user is not required to have any experience in temporal logics nor to be familiar with the graphical notation of DECLARE constraints, but can express temporal properties using his/her own words. The temporal properties that can be modeled with the tool cover the entire Multi-Perspective (MP-DECLARE) language that can not only express control-flow properties, but also other perspectives like data, time and resource. An important contribution of the paper is that the evaluation presented in this paper can be considered, to the best of our knowledge, the first attempt to test the usability of speech recognition in business process modeling.

describe evaluation results

To summarize, we are able to provide the following contributions in comparison to the state of the art:

1. we greatly increase the coverage of DECLARE constraint types wrt. [1]
2. we handle more flexible inputs so that more expressions can be recognized by the speech recognition tool
3. constraints are now linked to each other, enabling the definition of actual declarative process models, rather than just individual constraints
4. no one-shot approach

what does this mean?

5. we ensure additional expressiveness by incorporating conditions from data, time, and resource perspectives
6. the generated process models can be directly used as input for, e.g., conformance checking, log generation, and process monitoring (being the modeler integrated in an analysis toolkit)
7. we test the usability of speech recognition in business process modeling

describe remainder of paper

Template	**LTL semantics**	**Activation**
responded existence	$\mathbf{G}(A \rightarrow (\mathbf{O}B \vee \mathbf{F}B))$	A
response	$\mathbf{G}(A \rightarrow \mathbf{F}B)$	A
alternate response	$\mathbf{G}(A \rightarrow \mathbf{X}(\neg A \mathbf{U} B))$	A
chain response	$\mathbf{G}(A \rightarrow \mathbf{X}B)$	A
precedence	$\mathbf{G}(B \rightarrow \mathbf{O}A)$	B
alternate precedence	$\mathbf{G}(B \rightarrow \mathbf{Y}(\neg B \mathbf{S} A))$	B
chain precedence	$\mathbf{G}(B \rightarrow \mathbf{Y}A)$	B
not responded existence	$\mathbf{G}(A \rightarrow \neg(\mathbf{O}B \vee \mathbf{F}B))$	A
not response	$\mathbf{G}(A \rightarrow \neg \mathbf{F}B)$	A
not precedence	$\mathbf{G}(B \rightarrow \neg \mathbf{O}A)$	B
not chain response	$\mathbf{G}(A \rightarrow \neg \mathbf{X}B)$	A
not chain precedence	$\mathbf{G}(B \rightarrow \neg \mathbf{Y}A)$	B

Table 1: Semantics for Declare templates

2 Background

2.1 Declarative Process Modeling

Declare DECLARE is a declarative process modeling language originally introduced by Pesic and van der Aalst in [19]. Instead of explicitly specifying the flow of the interactions among process activities, DECLARE describes a set of constraints that must be satisfied throughout the process execution. The possible orderings of activities are implicitly specified by constraints and anything that does not violate them is possible during execution. In comparison with procedural approaches that produce "closed" models, i.e., all that is not explicitly specified is forbidden, DECLARE models are "open" and tend to offer more possibilities for the execution. In this way, DECLARE enjoys flexibility and is very suitable for highly dynamic processes characterized by high complexity and variability due to the changeability of their execution environments.

A DECLARE model consists of a set of constraints applied to activities. Constraints, in turn, are based on templates. Templates are patterns that define parameterized classes of properties, and constraints are their concrete instantiations (we indicate template parameters with capital letters and concrete activities in their instantiations with lower case letters). They have a graphical representation understandable to the user and their semantics can be formalized using different logics [16], the main one being LTL over finite traces, making them verifiable and executable. Each constraint inherits the graphical representation and semantics from its template. Table 1 summarizes some Declare templates (the reader can refer to [3] for a full description of the language). Here, the **F**, **X**, **G**, and **U** LTL (future) operators have the following intuitive meaning: formula $\mathbf{F}\phi_1$ means that ϕ_1 holds sometime in the future, $\mathbf{X}\phi_1$ means that ϕ_1 holds in the next position, $\mathbf{G}\phi_1$ says that ϕ_1 holds forever in the future, and, lastly, $\phi_1 \mathbf{U} \phi_2$ means that sometime in the future ϕ_2 will hold and until that moment ϕ_1 holds (with ϕ_1 and ϕ_2 LTL formulas). The **O**, and **Y** LTL (past) operators have the following meaning: $\mathbf{O}\phi_1$ means that ϕ_1 holds sometime in the past, and $\mathbf{Y}\phi_1$ means that ϕ_1 holds in the previous position.

The major benefit of using templates is that analysts do not have to be aware of the underlying logic-based formalization to understand the models. They work with the graphical representation of templates, while the underlying formulas remain hidden. Consider, for example, the *response* constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$. This constraint indicates that if *a occurs*, b must eventually *follow*. Therefore, this constraint is satisfied for traces such as $\mathbf{t}_1 = \langle a, a, b, c\rangle$, $\mathbf{t}_2 = \langle b, b, c, d\rangle$, and $\mathbf{t}_3 = \langle a, b, c, b\rangle$, but not for $\mathbf{t}_4 = \langle a, b, a, c\rangle$ because, in this case, the second instance of a is not followed by a b. Note that, in $\mathbf{t}_2$, the considered response constraint is satisfied in a trivial way because a never occurs. In this case, we say that the constraint is *vacuously satisfied* [14]. In [6], the authors introduce the notion of *behavioral vacuity detection* according to which a constraint is non-vacuously satisfied in a trace when it is activated in that trace. An *activation* of a constraint in a trace is an event whose occurrence imposes, because of that constraint, some obligations on other events (targets) in the same trace. For example, a is an activation for the *response* constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$ and b is a target, because the execution of a forces b to be executed, eventually. In Table 1, for each template the corresponding activation is specified.

An activation of a constraint can be a *fulfillment* or a *violation* for that constraint. When a trace is perfectly compliant with respect to a constraint, every activation of the constraint in the trace leads to a fulfillment. Consider, again, the response constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$. In trace $\mathbf{t}_1$, the constraint is activated and fulfilled twice, whereas, in trace $\mathbf{t}_3$, the same constraint is activated and fulfilled only once. On the other hand, when a trace is not compliant with respect to a constraint, an activation of the constraint in the trace can lead to a fulfillment but also to a violation (at least one activation leads to a violation). In trace $\mathbf{t}_4$, for example, the response constraint $\mathbf{G}(a \rightarrow \mathbf{F}b)$ is activated twice, but the first activation leads to a fulfillment (eventually b occurs) and the second activation leads to a violation (b does not occur subsequently). An algorithm to discriminate between fulfillments and violations for a constraint in a trace is presented in [6].

Tools implementing process mining approaches based on Declare are presented in [15]. The tools are implemented as plug-ins of the process mining framework ProM.

Multi-Perspective Declare In this section, we illustrate a multi-perspective version of Declare (MP-Declare) introduced in [7]. This semantics is expressed in Metric First-Order Linear Temporal Logic (MFOTL) and is shown in Table 2. We describe here the semantics informally and we refer the interested reader to [7] for more details. To explain the semantics, we have to introduce some preliminary notions.

The first concept we use here is the one of *payload* of an event. Consider, for example, that the execution of an activity Submit Loan Application (S) is recorded in an event log and, after the execution of S at timestamp τ_S, the attributes *Salary* and *Amount* have values 12 500 and 55 000. In this case, we say that, when S occurs, two special relations are valid *event*(S) and

Template	**MFOTL Semantics**
responded existence	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow (\mathbf{O}_I(B \wedge \exists y.\varphi_c(x,y)) \vee \mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y)))))$
response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y))))$
alternate response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \mathbf{X}(\neg(A \wedge \varphi_a(x))\mathbf{U}_I(B \wedge \exists y.\varphi_c(x,y)))))$
chain response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \mathbf{X}_I(B \wedge \exists y.\varphi_c(x,y)))$
precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \mathbf{O}_I(A \wedge \exists y.\varphi_c(x,y)))$
alternate precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \mathbf{Y}(\neg(B \wedge \varphi_a(x))\mathbf{S}_I(A \wedge \exists y.\varphi_c(x,y))))$
chain precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \mathbf{Y}_I(A \wedge \exists y.\varphi_c(x,y)))$
not responded existence	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \neg(\mathbf{O}_I(B \wedge \exists y.\varphi_c(x,y)) \vee \mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y)))))$
not response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \neg\mathbf{F}_I(B \wedge \exists y.\varphi_c(x,y))))$
not precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \neg\mathbf{O}_I(A \wedge \exists y.\varphi_c(x,y)))$
not chain response	$\mathbf{G}(\forall x.((A \wedge \varphi_a(x)) \rightarrow \neg\mathbf{X}_I(B \wedge \exists y.\varphi_c(x,y)))$
not chain precedence	$\mathbf{G}(\forall x.((B \wedge \varphi_a(x)) \rightarrow \neg\mathbf{Y}_I(A \wedge \exists y.\varphi_c(x,y)))$

Table 2: Semantics for Multi-Perspective Declare constraints

$p_S(12\,500, 55\,000)$. In the following, we identify *event*(S) with the event itself S and we call $(12\,500, 55\,000)$, the *payload* of S.

Note that all the templates in MP-Declare in Table 2 have two parameters, an activation and a target (see also Table 1). The standard semantics of Declare is extended by requiring two additional conditions on data, i.e., the *activation condition* φ_a and the *correlation condition* φ_c, and a time condition. As an example, we consider the response constraint "activity Submit Loan Application is always eventually followed by activity Assess Application" having Submit Loan Application as activation and Assess Application as target. The activation condition is a relation (over the variables corresponding to the global attributes in the event log) that must be valid when the activation occurs. If the activation condition does not hold the constraint is not activated. The activation condition has the form $p_A(x) \wedge r_a(x)$, meaning that when A occurs with payload x, the relation r_a over x must hold. For example, we can say that whenever Submit Loan Application occurs, and the amount of the loan is higher than 50 000 euros and the applicant has a salary lower than 24 000 euros per year, eventually an assessment of the application must follow. In case Submit Loan Application occurs but the amount is lower than 50 000 euros or the applicant has a salary higher than 24 000 euros per year, the constraint is not activated.

The correlation condition is a relation that must be valid when the target occurs. It has the form $p_B(y) \wedge r_c(x, y)$, where r_c is a relation involving, again, variables corresponding to the (global) attributes in the event log but, in this case, relating the payload of A and the payload of B. A special type of correlation condition has the form $p_B(y) \wedge r_c(y)$, which we call *target condition*, since it does not involve attributes of the activation.

Finally, in MP-Declare, also a time condition can be specified through an interval ($I = [\tau_0, \tau_1)$) indicating the minimum and the maximum temporal distance allowed between the occurrence of the activation and the occurrence of the corresponding target.

2.2 From Natural Language to Declarative Models

A crucial component of our work involves the extraction of declarative constraints from natural language, referred to as a textual description in the remainder. This extraction step involves the identification of the described actions (a.k.a. activities), such as *create invoice* and *approve invoice* in s_1 of Table 3, as well as the identification of the extract relation described for these actions, e.g., corresponding to a precedence or response constraint.

Due to the inherent flexibility of natural language, this extraction step can be highly challenging. Its difficulty manifests itself in the sense that, on the one hand, the same declarative constraint can be expressed in a wide variety of textual descriptions, whereas, on the other hand, subtle textual differences can completely change the meaning of the described constraint. These two complimentary challenges can be illustrated as follows:

ID	**Description**
s_1	An invoice must be created before the invoice can be approved.
s_2	A bill shall be created prior to it being approved.
s_3	Invoice creation must precede its approval.
s_4	Approval of an invoice must be preceded by its creation.
s_5	Before an invoice is approved, it must be created.

Table 3: Different descriptions of *Precedence(create order, approve order)*

Variability of textual descriptions.. As shown in Table 3, the same declarative constraint can be described in a broad range of manners. Key types of differences occur due to the use of synonyms, of different grammatical structures, and due to order reversals.

- **Synonyms.** Synonyms are omnipresent in any unstructured or semi-structured natural language text. Their presence impacts two aspects of constraint descriptions. First, synonymous terms or phrases can be used to refer to what is semantically the same action, e.g., *create invoice* in s_1 and *create bill* in s_2. Second, they can be used to describe the inter-relations between actions in different manners. E.g., "*before the invoice can be approved*" in s_1 has the same implications for the declarative constraint as "*prior to it being approved*" in s_2.
- **Different grammatical structures.** Textual descriptions of constraints can have widely different grammatical structures. Given that such a description captures a number of actions and their inter-relations, a particularly important distinction is to be made between descriptions that use *verbs* to denote actions, e.g., "*An invoice must be created*" in s_1, versus those that use *nouns* to denote actions, e.g., "*invoice creation*" in s_3.
- **Description order.** Finally, constraint descriptions can differ in the order in which they describe the different components of binary constraints, i.e.,

whether they describe the constraint in a chronological fashion, e.g., s_1 to s_3, or the reverse order, such as s_4 and s_5.

To achieve our goal of supporting users in the elicitation of declarative process models, we do must not limit the user too much in terms of the input that they can provide. Rather, it has to be accommodating to the different manners in which its users may choose to describe constraints. However, a successful approach must be able to do this while simultaneously being able to recognize subtle distinctions between different constraints, as discussed next.

Subtle differences leading to different constraints.. Small textual differences can have a considerable impact on the semantics of a constraint description and, thus, on the constraints that should be extracted from them. To illustrate this, consider the descriptions in Table 4. In comparison to description s_6, the three other descriptions each differ by only a single word. However, as shown in the right-hand column, the described constraints vary greatly. This is due to the following factors:

ID	Description	Constraint
s_6	If an order is shipped, an invoice must be sent.	Resp.(ship order, send invoice)
s_7	If an order is shipped, an invoice can be sent.	Prec.(ship order, send invoice)
s_8	If an order is shipped, an invoice must be sent first.	Prec.(send invoice, ship order)
s_9	If an order is shipped, an invoice must not be sent.	NotScn.(ship order, send invoice)

Table 4: Subtle differences between constraint descriptions

– **Obligation.** The difference between the Response constraint of s_6 and the Precedence constraints described by s_7 lies in the obligation associated with the *send invoice* action. The former specifies that this action *must* occur, indicating an obligation, whereas the latter specify that it *can* occur. As such, the activation and the target of the constraint are swapped.
– **Order indicators.** As discussed before, constraints can be described in chronological and non-chronological orders. This difference is often signaled through small textual clues, typically through the use of temporal prepositions. This is seen for description s_8, where the use of *first* completely reverses the meaning of the described constraint.
– **Negation.** Finally, it should be clear that the presence of negation drastically changes the meaning of a constraint, as seen for s_9. **what to say about NotSuccession?**

State of the art. So far, only one approach has been developed to automatically extract declarative process constraints from natural language text. This approach, by Van der Aa et al. [1], is able to extract five types of Declare templates, Init, End, Precedence, Response, Succession, as well as their negated forms. The approach aims to handle the aforementioned extraction challenges, in particular based on a unique approach to the extraction of noun-based actions. Its evaluation results show that it is able to handle a reasonable variety in its input. Recognizing its potential, we build on this approach. Especially using its technique for activity extraction and employing it for the extraction of the aforementioned five constraint templates.

Nevertheless, due to its limitations, we extend the state-of-the-art approach in two main manners: (1) we generalized some of the pattern recognition mechanisms in order to handle more flexible input, (2) we cover eight additional constraint templates, and (3) we add support for the identification of multi-perspective constraints.

3 Interactive Elicitation of Declarative Models

Figure 1 provides an overview of the main components of the Speech2RuM approach. As shown, the user provides input through speech as well as interaction with the Graphical User Interface (GUI). Based on this input, users are able to construct a declarative process model through three main functions: (1) describing constraints in natural language using speech, (2) augmenting constraints with data- and time-perspectives, and (3) optionally editing and connecting the model's constraints.

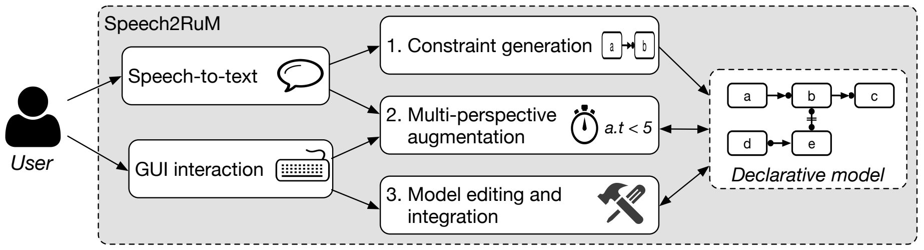

Fig. 1: Overview of the Speech2RuM approach

In this section, we outline our conceptual contributions with respect to the first two functions, i.e., for the transformation from natural language text into declarative constraints (Section 3.1) and into augmentation of constraints with multi-perspective constraints (Section 3.2). We cover the implementation of the approach and the user interaction it supports in Section 4.

3.1 Constraint generation

In this component, our approach aims to turn a constraint description, recorded using speech speech, into one or more declarative constraints. For this task, we enhanced the existing state-of-the-art approach for the extraction of declarative constraints from natural language text [1].

In particular, we take the result of the parsing step of that existing approach as input to the constraint generation approach. Given a sentence s, parsing yields a list A_s of actions described in the sentence and the inter-relations that exist between the actions, i.e., a mapping $rel_s : A_s \times A_s \to type$, with $type \in \{xor, and, dep\}$. As depicted in Figure 2, an action $a \in A_s$ consists of a verb and optional subjects and objects, which may, furthermore, all be augmented with additional information, such as whether or not an action is negated.

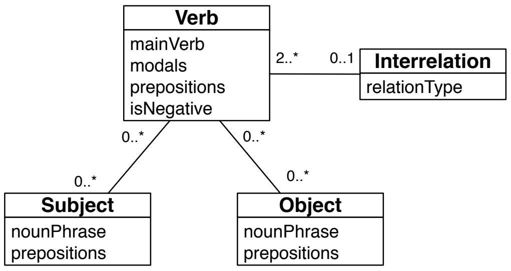

Fig. 2: Semantic components returned in the parsing step of [1]

By building on this parsing step and the existing constraint-generation approach, we have added support to handle the additional constraint types presented in Table 5. Based on a set of activities A_s and a relation rel_s extracted for a sentence s, these additional types are handled as follows:

– ***Participation* and *Absence.*** If A_s contains only one action, either an existence or absence constraint is established for the action, depending on whether it is negative or not.
– ***AtMostOne.*** If a *Participation* constraint is originally recognized, we subsequently check if s specifies a bound on its number of executions, i.e., by checking for phrases such as *at most once*, *not more than once*, *one time.*
– ***CoExistence.*** *CoExistence* relations are identified for sentences with two actions in an *and* relation, typically extracted from a coordinating conjunction like "*shipped and paid*". Furthermore, some notion of obligation should be present, in order to distinguish between actions that *can* happen together and ones that *should.*
– ***RespondedExistence.*** Responded existence constraints are similar to the originally extracted *Response*. Their extraction occurs when two actions are in a dependency relation, i.e., a_1 *dep* a_2 for which it holds that the target action, a_2, is indicated to be mandatory, e.g., using *must.* The key difference between *Response* and *RespondedExistence* is that the former includes a notion of order, i.e., a_1 precedes a_2, whereas no such order is specified for *RespondedExistence* constraints.
– ***NotCoExistence.*** This constraint type is identified for negated forms of descriptions of both *CoExistence* and *RespondedExistence* constraints. Semantically, in both cases, the description shall state that two actions should not appear for the same process instance. This is, e.g., seen in Table 5, where

"*If an application is accepted, it cannot be rejected*" is actually the negative form of a *RespondedExistence* description.

– ***ChainPrecedence* and *ChainResponse*.** These constraint types are specializations of *Precedence Response* constraints. Chain constraints are recognized by the presence of a temporal preposition that indicates immediacy. Generally, such a preposition is associated with the verb of either an action a_1 or a_2 in a relation a_1 *dep* a_2. For this, we consider the preposition *immediately* and several of its synonyms, i.e., *instantly*, *directly*, and *promptly*.

Constraint	**Example**
Participation	An invoice must be created
Absence?	Dogs are <u>not</u> allowed in the restaurant.
AtMostOne	An invoice should be paid <u>at most once</u>.
CoExistence	Order shipment and invoice payment <u>should occur together</u>.
RespondedExistence	If a product is produced, it <u>must be tested</u>.
NotCoExistence	If an application is accepted, it cannot be rejected.
ChainPrecedence	After an order is received, it may <u>immediately</u> be refused.
ChainResponse	After an order is received, it must <u>directly</u> be checked.

Table 5: Additional constraint types in Speech2RuM

Note that if a sentence does not yield constraints for these additional types, constraint generation proceeds to extract the six constraint types covered by the original approach.

3.2 Multi-Perspective Augmentation

Three types of conditions: activation, correlation, and time conditions.

we do not need to recognize which is being described

Activation conditions. Activation conditions define constraints on attribute values associated with the *activation* activity of a constraint. They denote specific requirements that must be met in order for a constraint to be applicable, e.g., stating that Response(receive loan application, check loan application) should only apply when the amount associated with the application is above 50 000EUR.

Our approach is able to handle input that follows the patterns shown in Table 6.

Correlation conditions.

Time conditions.

4 Implementation

Fabrizio

Type	**Pattern**	**Semantics**
Activation + Target	example greater	
Activation + Target	example lower	
Activation + Target	example negated	
Activation + Target	example categorical	
Correlation	exampl1	
Correlation	Example2	
Time		

Table 6: Condition patterns supported by Speech2RuM

5 User Evaluation

In order to improve the current implementation and assess the overall feasibility of the developed Speech2RuM approach, we conducted a qualitative case study. The threefold aim was to (1) identify means for improving the interface, (2) identify potential issues and demands of individuals with different backgrounds, and (3) identify potential usage scenarios. In the following we will first describe the study setup including the data we collected and our analysis procedure (Section 5.1) before discussing our findings (Section 5.2).

Han: we have to decide on a consistent way to refer to the tool, e.g., interfact/tool/implementation

I tried to stick to ”tool” but we can change it in any way you want.

5.1 Study setup

Participants. For our study we selected a total of eight participants with differing characteristics (referred to as P1 to P8 in Table 7). In particular, we selected two participants that only had experience related to business process management (tier 1), two participants with experience related to business process management and modeling temporal constraints (tier 2), two participants that had experience related to business process management and using DECLARE to model temporal constraints (tier 3), and two participants that had experience related to business process management and using a similar tool to model temporal constraints (tier 4). We chose this differentiation to gain insight about the usability of the tool for individuals with different levels of expertise and to assess potential issues and demands by individuals with different backgrounds.

Study setup. The study was conducted via Skype by a team consisting of a *facilitator* and an *observer*, with the facilitator guiding the participant and the observer serving in a supporting role. We used the production and shipping process in a bicycle factory[4] as a background for the study because it can be expected that every participant will be somewhat familiar with such a process. We also created a help document[5] that would explain different constraints that the tool could handle to the participant.

[4] The scenario can be found on Github: https://git.io/Jv1Hn
[5] The help document can be found on Github: https://git.io/Jv1Hc

Prior to the study, the facilitator sent the participants a link to the scenario, the help document, and the tool itself. He suggested for them to read the documents and install the tool prior to the study. At the beginning of the study the facilitator introduced the participant to the study procedure and to the tool by showing how to enter a single constraint using speech input. He also asked the participant to think aloud during the study. After any potential request for clarification, the participant shared her/his screen, the facilitator started the video recording, and began guiding the participant through the prepared scenario. The participants were then asked to carry out multiple tasks starting with reading prepared sentences related to the scenario to enter constraints before forming their own sentences based on predefined parameters and then creating constraints without prior guidance. During this time facilitator and observer noted down any issues the participant mentioned or any situations that appeared as s/he had issues using the tool based on the usability heuristics by Nielsen [17]. At the end of the study the facilitator conducted a short follow-up interview focusing on clarifying questions about the observation or issues the participant had mentioned during the test. He also asked the participant about what s/he *"liked about the tool"*, what s/he *"wished would be different"* and *"under which circumstances s/he would use the tool"*. After the study the participants were asked to complete a short post-questionnaire. The questionnaires consisted of multi-point Likert scales including the System Usability Scale (SUS) [5] and scales covering satisfaction, expectation confirmation, continuation intention and usefulness which were adapted from Bhattacherjee [4] to assess continued use of information systems[6]. The individual studies lasted between 14 and 35 minutes including interviews.

	all	**tier 1 (P6, P7)**	**tier 2 (P3, P8)**	**tier 3 (P2, P4)**	**tier 4 (P1, P5)**
SUS	73.13	71.25	81.25	67.50	72.50
Satisfaction	3.46	3.67	3.67	3.00	3.50
Expectation	3.69	3.67	3.50	3.50	4.00
Future intentions	2.61	2.33	2.00	2.00	3.67
Usefulness	3.00	2.63	3.13	2.75	3.50

Table 7: Questionnaire results represented as means across the different groups

Result analysis. To analyze the collected data, the research team first created an affinity diagram [13] based on the video recordings, observations, and follow-up interviews, by creating a single paper note for each issue that was reported. The team then clustered the notes based on emerging themes, which resulted in 139 notes divided over 21 distinct clusters. The results of the questionnaire served as an additional qualitative data point during the analysis (Table 7).

[6] The complete questionnaire can be found on Github: https://git.io/Jv1HC

5.2 Findings

In general our study revealed that the current interface was reasonably usable as evident by an average SUS score of 73.13 (table 7) which can be considered to be a good results compared to the general average of 68 [5]. Our study however also pointed towards multiple usability issues which were addressed to create an improved version of the interface (figure). These include the possibility to simply undo the last recording, editing the text that was recognized directly rather than modifying the constraints that the system recognized and adding different conditions (activation, correlation, time) via speech by selecting the respective record button. We also remove the previously included "cancel" button because it was often pressed by participants because they thought it would just stop and not stop and discard the previous recording.

Difference between participants with different backgrounds First it should be noted that all participants used some means to **write sentences down** before saying them out loud. They either wrote them *"on paper"* (P1), *"typed them on [their] computer"* (P2) or they noted down key parts that they wanted to say (observation of P7). Most participants also used somewhat **unnatural sentences** when entering constraints (*"ordering wheels must be executed before"*, P2). Some even explicitly tried to identify keywords (*"Is activity like a keyword?"*, P8) that the algorithm would pick up and translate into the desired constraint (*"Oh it didn't get this one. How about this? [adds "then" to wording]"*, P7). Only P1 used natural sentences from the start. This finding indicates that individuals need to learn about how to effectively use speech as a means of modeling constraints regardless of their previous experience related to BPM or DECLARE.

There were however also noticeable differences between individuals from different background starting with participants from tiers 1 and 2 **mainly relying on the visual process model** to assess whether the tool had understood them correctly while participants from tiers 3 and 4 **mainly relied on the textual representations** of the entered constraints. This was mainly evident by how they tried to fix potential errors. Participants from tiers 1 and 2 initially tried to edit the visual process model (observation of P6, *"I want to de-merge the visualization"*, P3) – which was not possible in the version of the tool we tested – while participants from tiers 3 and 4 directly started to edit the textual constraints (observation of P1 and P5). Only P2 attempted both. This difference might be related to participants from tiers 3 and 4 already being familiar with DECLARE while participants from tiers 1 and 2 were more familiar with visual process representations e.g. in the form of BPMN models. It thus appears reasonable that each group would mainly focus on the visualization that they are most familiar with.

Findings from our study also indicated that the way the **recognized text** was displayed (figure) might not have been ideal since participants only referred to it after they had entered the first constraint and then did not check it again for future constraints (observation of P3 and P6). This led to confusion in particular among less experienced participants who thought that they did not formulate the constraint correctly while the issue simply was that their verbalized sentence was not correctly recognized.

The previously discussed issues might also have led to **less experienced participants perceiving the tool as less useful and being less inclined to use the tool in the future** than experienced participants (table 7). Related to this finding it thus does appear counterintuitive that experienced participants (tiers 3 and 4) were less satisfied with the tool than less experienced participants (tiers 1 and 2). This finding can potentially be explained by the small sample size but also by P5 expressing that s/he would have *"expected more constraint types"* (P5) indicating that experienced participants might have had more expectations related to the functionality of the tool than less experienced participants.

Taking the aforementioned findings together it appears as if creating spoken constraints that would be turned into formal constraints might have **increased the complexity of entering constraints** in particular for less experienced participants. Writing down sentences before speaking them out and trying to identify keywords requires for users to formalize constraints in a way that might even be counterproductive when using speech input since our tool was created with the aim to understand natural sentences. Moreover it appears difficult for less experienced uses to assess the correctness of their entered constraints due to their potential lack of understanding DECLARE. It thus appears reasonable to consider saving the recorded text in addition to the created constraints which would allow experienced users to assess the entered constraints post-hoc.

Potential usage scenarios The participants mentioned multiple scenarios during which they would consider speech as useful for entering constraints. Most of them stated that using speech would **improve their efficiency** because they could *"quickly input stuff"* (P4) with P5 pointing out that it would be particularly useful for *"people that are not familiar with the editor"* (P5). P8 however had a different opinion stating that *"manually is quicker"* (P8). S/he did however acknowledge that speech might be useful *"when I forget the constraint name"* (P8). S/he thus indicated that **using natural language sentences** might be useful even when not using speech. Related to this participants suggested that speech input might be useful *"for designing"* (P3) or when reading *"from a textual description"* (P1). This finding also points towards the potential necessity for **post-processing** the entered constraints.

Finally P1 also mentioned that speech input would be useful for **mobile scenarios** such as *"using a tablet"* (P1) because entering text in a scenario like that is usually much more time consuming than when using a mechanical keyboard.

6 Related work

7 Conclusion

References

1. van der Aa, H., Di Ciccio, C., Leopold, H., Reijers, H.A.: Extracting declarative process models from natural language. In: International Conference on Advanced Information Systems Engineering. pp. 365–382. Springer (2019)
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 Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science - R&D 23(2), 99–113 (2009)
4. Bhattacherjee, A.: Understanding information systems continuance: an expectation-confirmation model. MIS quarterly pp. 351–370 (2001)
5. Brooke, J., et al.: Sus-a quick and dirty usability scale. Usability evaluation in industry 189(194), 4–7 (1996)
6. Burattin, A., Maggi, F.M., van der Aalst, W.M., Sperduti, A.: Techniques for a Posteriori Analysis of Declarative Processes. In: EDOC. pp. 41–50. Beijing (2012)
7. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. 65, 194–211 (2016)
8. van Der Aalst, W.M., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science-Research and Development 23(2), 99–113 (2009)
9. 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)
10. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A., et al.: Fundamentals of business process management, vol. 1. Springer (2013)
11. Friedrich, F., Mendling, J., Puhlmann, F.: Process model generation from natural language text. In: International Conference on Advanced Information Systems Engineering. pp. 482–496. Springer (2011)
12. Herbst, J., Karagiannis, D.: An inductive approach to the acquisition and adaptation of workflow models. In: Proceedings of the IJCAI. vol. 99, pp. 52–57. Citeseer (1999)
13. Holtzblatt, K., Wendell, J.B., Wood, S.: Rapid contextual design: a how-to guide to key techniques for user-centered design. Elsevier (2004)
14. Kupferman, O., Vardi, M.Y.: Vacuity Detection in Temporal Model Checking. International Journal on Software Tools for Technology Transfer 4, 224–233 (2003)
15. Maggi, F.M.: Declarative process mining with the Declare component of ProM. In: BPM (Demos) (2013)
16. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative Specification and Verification of Service Choreographies. ACM Transactions on the Web 4(1) (2010)
17. Nielsen, J.: Enhancing the explanatory power of usability heuristics. In: Proceedings of the SIGCHI conference on Human Factors in Computing Systems. pp. 152–158 (1994)

18. de Oliveira, J.P.M., Avila, D.T., dos Santos, R.I., Fantinato, M.: Assisting process modeling by identifying business process elements in natural language texts. In: Advances in Conceptual Modeling: ER 2017 Workshops AHA, MoBiD, MREBA, OntoCom, and QMMQ, Valencia, Spain, November 6–9, 2017, Proceedings. vol. 10651, p. 154. Springer (2017)
19. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: Full support for loosely-structured processes. In: IEEE International EDOC Conference 2007. pp. 287–300 (2007)
20. Schumacher, P., Minor, M., Schulte-Zurhausen, E.: Extracting and enriching workflows from text. In: 2013 IEEE 14th International Conference on Information Reuse & Integration (IRI). pp. 285–292. IEEE (2013)
21. 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)