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.

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 imperial process modeling paradigm. Therefore, such processes are better captured using *declarative* process models. These models 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 process models is known to be difficult, especially for domain experts that lack modeling expertise [11,21], and can be highly time-consuming [12]. Due to these barriers to model creation, processes are often documented using natural language instead []. Recognizing this, several works have been developed that automatically extract process models from natural language texts (cf., [2, 11, 18, 20]). The vast majority of these focus on the generation of imperative process descriptions, whereas only one preliminary approach [1] targets the extraction of declarative process models.

In this work, we aim to support the elicitation of declarative process models based on natural language input. In particular, we present an interactive approach that employs speech recognition and is integrated in a declarative modeling and analysis toolkit. In this manner, we are able to provide the following contributions in comparison to the state of the art [1]:

1. greatly increase coverage of declare constraint types covered

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. handling more flexible input
3. no one-shot approach
4. additional expressiveness by incorporating conditions from data, time, and resource perspectives
5. constraints are now linked to each other, enabling the definition of actual declarative process models, rather than just individual constraints
6. directly use the generated process models as input for, e.g., conformance checking, event log generation, and process monitoring

describe evaluation results
describe remainder of paper

2 Background

2.1 Declarative Process Modeling

2.2 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 $\mathbf{F}$, $\mathbf{X}$, $\mathbf{G}$, and $\mathbf{U}$ 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 $\mathbf{O}$, and $\mathbf{Y}$ 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.

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

2.3 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 $p_S(12\,500, 55\,000)$. 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.

In this paper, we aim at discovering constraints that correlate an activation and a target condition. For example, we can find 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, then eventually Assess Application must follow, and the assessment type will be *Complex* and the cost of the assessment higher than 100 euros.

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.

2.4 From Natural Language to Declarative Models

describe the goal of such an extraction, which is largely extracting action and determining the interrelations that hold among them

The extraction of declarative constraints from text is highly challenging due to the inherent flexibility of natural language. This 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 sources of such differences include:

- **Use of synonyms.** The use of synonyms is omnipresent in any unstructured or semi-structured natural language text. The use of synonyms 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 interrelations 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.** s1 vs s3
- **Order reversals.** s4 and s5

recap why this makes it then hard if all natural language descriptions that an approach needs to consider would be structured in the same manner, there would be no challenge. However, an approach that actually supports the elicitation of declarative constraints 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.
- Negation
- Order indicators
- Propositions
- Mandatory versus optional.

Table 4: Subtle differences between constraint descriptions

- state-of-the-art extraction approach
which challenges it handles and which constraints it covers
- our delta w.r.t. state of the art

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. In the remainder of this section, we will describe each of these three components in detail.

do we need to mention how to handle speech or do we put implementation details somewhere else?

3.1 Constraint generation

In this component, our approach aims to turn a recorded sentence (transformed from speech to text using a standard software library [?]) into one or more declarative constraints. For this task, we enhanced the existing state-of-the-art

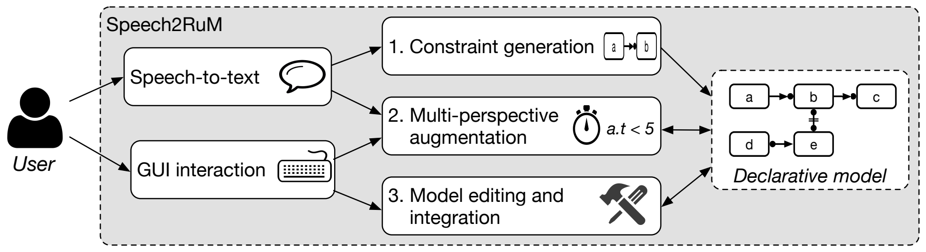

Fig. 1: Overview of the Speech2RuM approach

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 $rel_s : A_s \times A_s \rightarrow type$, 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.

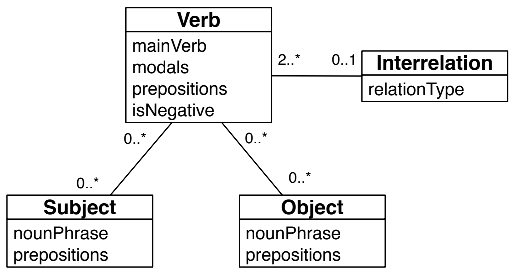

Fig. 2: Semantic components returned in the parsing step of [1]

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* .** When a sentence has two actions in A_s that are in a relation of type *and* (and without any *dep* relations) are transformed into either a *CoExistence* or a *NotCoExistence* constraint, depending on whether there is a negation present.
– ***NotCoExistence* .**
– ***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.
– ***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 not allowed in the restaurant.
AtMostOne	An invoice should be paid at most once.
CoExistence	Order shipment and invoice payment should occur together.
NotCoExistence	If an application is accepted, it cannot be rejected.
RespondedExistence	If a product is produced, it must be tested.
ChainPrecedence	After an order is received, it may immediately be refused.
ChainResponse	After an order is received, it must directly 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

3.3 Model Editing and Integration

4 User Evaluation

In order to improve the current implementation and assess the overall feasibility of the developed user interface we conducted a qualitative case study. The 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 and our analysis procedure (section 4.1) before discussing our findings (section 4.2).

4.1 Study setup

For our study we selected a total of eight participants (table 6). We selected two participants that only had experience related to business process management (tier 1), two participants that had 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.

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 be familiar with a production and shopping process. We also created a help document[5] that would explain different constraints that the tool could handle to the participant.

Prior to the study, the facilitator sent a link to the scenario, the help document and a link for the tool the participants. 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 potential clarifying questions 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"*,

[4] The scenario can be found here: `https://github.com/alexandernolte/BPM2020_SpeechToRuM/blob/master/study-materials/study-scenario.pdf`

[5] The help document can be found here: `https://github.com/alexandernolte/BPM2020_SpeechToRuM/blob/master/study-materials/help-document.pdf`

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 individual studies lasted between 14 and 35 minutes including interviews. 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].

	all	**tier 1 (P6, P7)**	**tier 2 (P3, P8)**	**tier 3 (P2, P4)**	**tier 4 (P1, P5)**
SUS	75.00	71.25	70.00	83.75	72.50
Satisfaction	3.62	3.67	3.67	3.67	3.50
Expectation	3.00	3.67	3.50	3.33	4.00
Future intentions	2.10	2.33	2.00	3.00	3.67
Usefulness	3.14	2.63	3.13	3.50	3.50

Table 6: Questionnaire results

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. They then clustered the notes based on emerging themes which resulted in 139 notes in 21 distinct clusters. The questionnaire served as an additional qualitative data point during the analysis (table 6).

4.2 Findings

Our findings overall... Overall usability is sufficient / good (SUS) Minor changes to the interface that were mainly related to the usability (e.g. longer pauses during a sentence would lead to the speech recording to stop and the placement of new elements was somewhat confusing because they were not highlighted) Placement of the recognized text appeared to be an issue... People saw it correct once and then did not check again... that was particularly problematic for newcomers because of the following... Help document needs improvement for newcomers (P7 had troubles identifying suitable constraint)

Difference between users with different backgrounds Usage differences between the different tiers: - all used pre-formulated sentences (either typed or written) - most used somewhat unnatural sentences (e.g. activity ... takes place before activity ...)... tried to identify keywords (P7 by trial and error)... only P1 used natural speech - newcomers had issues realizing that constrains were

[6] The complete questionnaire can be found here:

incorrectly formulated because their sentence was not correctly understood... thought it was because the transition from text to constraint did not work - newcomers relied more on the model (T1 and 2) while participants that knew DECLARE relied more on the text - newcomers found it least useful and usable despite them being satisfied - professionals found it the most useful and had the highest future intentions to use it - pro user expected more constraint types (P5)

Potential usage scenarios Usage scenarios: - ...

Overall: - generally not ”natural” enough yet - additional layer of complexity (from natural language to what user thinks is understandable to the machine) hard for novices to check correctness (unfamiliar with constraints)... esp. P7 –¿ save recorded text as well (annotation)...

5 Related work

6 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)