Scenario-Based Model Checking of Declarative Process Models

Nicolai Schützenmeier, Martin Käppel, Myriel Fichtner and Stefan Jablonski
Institute for Computer Science, University of Bayreuth, Universitätsstraße 30, 95447 Bayreuth, Germany
{*nicolai.schuetzenmeier, martin.kaeppel, myriel.fichtner, stefan.jablonski*}*@uni-bayreuth.de*

Keywords: Business Process Management, Declarative Process Management, Declare, Model Verification.

Abstract: Modeling processes with *declarative process models*, i.e. sets of constraints which have to be satisfied throughout the whole process execution, allows for a great degree of flexibility in process execution. However, having a process specified by means of symbolic, textual or formal constraints comes along with the problem that it is often hard for humans to understand complicated interactions of constraints and overlook the entire process model without unintentionally neglecting important process details. Caused by these reasons, standard questions regarding process models, e.g. *"Can a running process instance still be completed successfully?"*, can often only be answered with great computational and temporal effort or even not at all. In this paper we present an efficient scenario-based approach for declarative process models, which supports process modelers in checking process models for important and common scenarios which regularly occur when modeling declarative processes. We implement our approach and show that the solutions for the scenarios can be computed within milliseconds even for real-life event logs. Furthermore, a user study conducted demonstrates that the error rate in understanding declarative process models is enormously reduced by using our implementation.

1 INTRODUCTION

In Business Process Management (BPM), two opposing classes of processes can be distinguished: *knowledge-intensive processes* (also called *decision-intensive processes*, *flexible processes* or *declarative processes*) and *routine processes* (Fahland et al., 2009; Fahland et al., 2010). While in routine processes the control flow is limited to a few a-priori defined variants, in knowledge-intensive processes the process participants are granted more influence on the control flow (based on their knowledge and decisions), resulting in a large set of execution variants.

This difference is also reflected in the associated process modeling languages. Knowledge-intensive processes are described by declarative process modeling languages (e.g. Declare (Pesic, 2008), DCR (Hildebrandt et al., 2013), GSM (Damaggio et al., 2011)), which describe a process by *restrictions* (so called *constraints*) over the behavior that must be satisfied throughout the whole process execution. In contrast imperative process modeling languages, such as Petri-nets (Aalst, 2011), BPMN[1], and UML[2], are well-suited for routine processes because they explicitly describe all feasible execution paths in a graph-based structure.

Model checking aims to verify that a given process model represents the intended behavior observed in reality and is an essential part of the process of model debugging and re-modeling (Dumas et al., 2013). In addition to automatic validations steps such as syntax validation (Mendling, 2009) and detection of unreachable paths, manual verification by domain experts or, more generally, humans involved in the execution of the process is a crucial part of the verification process (Dumas et al., 2013). This manual verification checks whether the process model adequately covers frequent or particularly critical scenarios of the underlying process and whether undesirable scenarios are appropriately prevented. These checks are performed by answering questions such as *"Does a certain state of the process execution reflect a situation that occurs in reality?"*, *"For a running process instance, what are all possible continuations or what activities I am allowed to perform next?"*, or *"Can a process instance still be (successfully) completed?"*.

For routine processes, these questions are usually easy to answer. For knowledge-intensive processes, however, this is anything but an easy task. There are many reasons for this: *(i)* The complexity of the constraints impose some edge cases that are easy to overlook (e.g. empty process instances) or lead to pos-

[1] www.bpmn.org
[2] www.uml.org

Schützenmeier, N., Käppel, M., Fichtner, M. and Jablonski, S.
Scenario-Based Model Checking of Declarative Process Models.
DOI: 10.5220/0011856400003467
In *Proceedings of the 25th International Conference on Enterprise Information Systems (ICEIS 2023) - Volume 2*, pages 406-417
ISBN: 978-989-758-648-4; ISSN: 2184-4992

sible misinterpretations (e.g. unnecessary execution of activities) (Haisjackl et al., 2016). *(ii)* The constraints cannot be considered in isolation from each other, since they all have to be fulfilled at the same time and influence each other. Therefore, we need to take a holistic perspective, which is complicated due to explicit and implicit dependencies (so called *hidden dependencies*) between constraints (De Smedt et al., 2016; De Smedt et al., 2018). *(iii)* The large degrees of freedom in process execution lead to an almost unmanageable number of possible process execution variants. These issues, combined with a less intuitive representation in terms of constraints, make the verification of declarative process models to a very tedious, strenuous and error-prone task. This finding is supported by numerous studies in the scientific literature, e.g. (Haisjackl et al., 2016).

In this paper, we identify three core functionalities that support the simulation and verification of different scenarios in knowledge-intensive processes and, thus, support domain experts and process participants to verify their models in a scenario-driven way. We conduct a user study that shows that these core functionalities, with appropriate tool support, lead to significant time savings, reduce the mental effort of the people involved and significantly lower the error rate.

The remainder of the paper is structured as follows. First, we introduce some basic terminology (cf. Section 3) and delimit our work from existing work (cf. Section 2). In Section 4 we propose three core functionalities, while in Section 5 we evaluate our approach from different angles. Finally, we provide some concluding remarks and suggest paths for future work.

2 RELATED WORK

Despite the well-known advantages of declarative process modeling languages for so-called flexible processes (Reichert and Weber, 2012), the corresponding declarative process models, which generally consist of several constraints, are notoriously difficult for humans to understand (Haisjackl et al., 2016; Nagel and Delfmann, 2021). The main reason for this is the mutual interaction of the constraints, which makes it very hard even for experts to understand the whole process model without overlooking important details. Another point of (Haisjackl et al., 2016) was the observation that individuals use the composition of graphical and text-based elements to interpret declarative models represented in Declare (Pesic, 2008), which is a widespread declarative modeling language. The idea of hybrid process representations combining both graphical and text-based elements did not address this issue and led to similar difficulties (Andaloussi et al., 2019a; Andaloussi et al., 2019b).

Recently approaches aimed at investigating declarative process models and helping people to understand them have been made. (De Smedt et al., 2018; De Smedt et al., 2016) address the issue of detecting so called *hidden dependencies* in declarative process models. Hidden dependencies are the result of the interaction of multiple constraints and are therefore not explicitly modeled in the model, making it difficult for humans to find them. The detected hidden dependencies are added to the models in the form of graphical and textual annotations to improve their understandability (De Smedt et al., 2018). (Di Ciccio et al., 2017; Corea et al., 2022) examine the set of constraints of declarative process models for redundancies and *inconsistencies*, which do not make a single execution trace satisfiable and hence the model useless. A possible solution to resolve the detected inconsistencies is proposed by (Corea et al., 2021). In (Schützenmeier et al., 2022), the authors provide a comparison approach for Declare process models that allows Declare models to be compared with each other in order to identify common properties and differences. All of these approaches are useful for making declarative process models more comprehensible. Nevertheless, they do not provide support for concrete scenarios at runtime, e.g. *”What are all possible continuations at a running process instance?”*.

(Bauer et al., 2011) presents a runtime verification technique for linear temporal logic (LTL) that enables to check traces for fulfillment. Since declarative process modeling languages like Declare (Pesic, 2008) are based on linear temporal logic over finite traces (LTL$_f$) (Montali et al., 2010), this approach can help to verify Declare models at runtime. Nevertheless, concrete scenarios as mentioned above cannot be handled either. In (Schützenmeier et al., 2019), the authors present a method to extract Declare constraints from LTL formulas by transforming common Declare templates into a standardized form called positive normal form, with the aim of reducing the complexity of Declare models.

While finite state automatons have proven to be well-suited for in-depth analysis of single-perspective declarative process models (Schützenmeier et al., 2022), they are hardly used in the multi-perspective case due to the associated state explosion (Käppel et al., 2021). There, mostly techniques that rely on SAT solving are employed. In such techniques, however, the search space used for the analysis (i.e. maximum trace length) must be defined beforehand in order to be able to work efficiently. Unfortunately, this prevents analyses in which a maximum trace length is not known in advance. However, multi-perspective declarative process models are not within the scope of our approach, and that is why we can rely on finite state automatons.

3 PRELIMINARIES

In this section, we introduce the theoretical concepts necessary for understanding the rest of the paper.

3.1 Events, Traces and Declarative Process Models

When a process instance is executed, this execution is recorded in a *(process) event log*. There we encounter *events* and *traces* (Aalst, 2011).

Definition 1. *An* ***event*** *encapsulates the execution of an activity (i.e. a well-defined step in a business process) in a particular process instance.*

Definition 2. *A* ***trace*** *is a finite sequence* $\sigma = \langle e_1, ..., e_n \rangle$ *of events that belong to the same process instance and are ordered by their execution time, where* $n := |\sigma|$ *denotes the* ***trace length*** *of* σ*.*

As we are interested in control-flow, we represent events by the name of the activity they encapsulate.

In declarative process modeling languages, a process is described by restrictions over the behavior that must be satisfied throughout the whole process execution. Formally, a *declarative process model* is a pair $P = (\mathcal{A}, \mathcal{T})$ where $\mathcal{A}$ is a finite set of *activities* and $\mathcal{T}$ is a finite set of *constraints*. Often these constraints are specified in terms of logical expressions, e.g. in the modeling language Declare the Linear Temporal Logic on finite traces is used for this purpose. In general, the semantics of such a declarative process model is interpreted on finite traces. Since business processes are assumed to terminate, the restriction to finite traces is not a limitation.

3.2 Automata Theory

We build our approach on the foundations of the theory of finite state automatons. It has been shown that the representation of single-perspective declarative process models as finite state automatons is well-suited for theoretical in-depth investigations and simulation tasks (Schützenmeier et al., 2022). Therefore, for most declarative process modeling languages algorithms exist to convert their process models into finite state automatons (Di Ciccio et al., 2017).

Definition 3. *A* ***deterministic finite-state automaton (FSA)*** *is a quintuple* $M = (\Sigma, Q, q_0, \delta, F)$ *where* Σ *is a finite (non-empty) set of symbols,* Q *is a finite (non-empty) set of states,* $q_0 \in Q$ *is an initial state,* $\delta : Q \times \Sigma \to Q$ *is the state-transition function, and* $F \subseteq Q$ *is the set of final states.*

A concatenation $\omega = a_1 \dots a_n$ of finitely many symbols $a_i \in \Sigma$ is called *word*. We call n the *length* of ω. The word of length 0 is called the *empty word* and is denoted by ε. All words that can be constructed with symbols from an alphabet Σ are denoted by $\Sigma^* := \{a_1 \dots a_n \mid n \in \mathbb{N}_{\geq 1}, a_i \in \Sigma\} \cup \{\varepsilon\}$. In order to enable the processing of words instead of single symbols, we extend the state-transition function as follows:

Definition 4. *For an FSA* $M = (\Sigma, Q, q_0, \delta, F)$ *we define the* ***extended state-transition function*** $\hat{\delta} : Q \times \Sigma^* \to Q$*,*

$$(s, \omega) \mapsto \begin{cases} s & \omega = \varepsilon \\ \delta(s, \omega) & \omega \in \Sigma \\ \delta(\hat{\delta}(s, a), b) & \omega = ab, \text{ with } a \in \Sigma, b \in \Sigma^*. \end{cases}$$

The set of all words accepted by an automaton M is called *language of* M, i.e. $\mathcal{L}(M) := \{\omega \in \Sigma^* \mid \hat{\delta}(\omega) \in F\}$.

In general, two different (non-isomorphic) automatons $M \neq M'$ can accept the same language, i.e. $\mathcal{L}(M) = \mathcal{L}(M')$. By applying a minimization algorithm to M (e.g. the Hopcroft algorithm (Hopcroft, 1971)), an automaton M_{min} with a minimal number of states can be constructed that accepts the same language as M, i.e. $\mathcal{L}(M_{min}) = \mathcal{L}(M)$. Minimizing an FSA reduces memory and speeds up various computations.

3.3 Linking Declarative Process Models and Finite State Automatons

As mentioned above, we can transform a declarative process model $P = (\mathcal{A}, \mathcal{T})$ into an FSA M_P (also called *process automaton*). Concrete transformations can be found in (Westergaard et al., 2013; Schützenmeier et al., 2022). In this paper, we use the transformation presented in (Schützenmeier et al., 2022) because this transformation directly yields a minimized FSA.

In this process automaton, the alphabet Σ is the set of activities $\mathcal{A}$ of the process P. Therefore, traces are represented as words. For example, a trace $\sigma = \langle a, a, c, d \rangle$ is represented as a word $\omega = aacd$. An accepted word represents a valid trace, i.e. a trace that satisfies all constraints $\tau \in \mathcal{T}$. Consequently, M_P accepts exactly those words whose corresponding traces are not forbidden by $\mathcal{T}$, i.e. the language of M_P is the set of all valid traces that can be derived from P.

4 CORE FUNCTIONALITIES (CFs)

Simulating all allowed paths of a process model allows both estimating the impact of changes and verifying the process model. When all process paths are known, it its possible to determine the extent to which the process model captures the desired behavior of the underlying process. However, the large number of possible execution paths in flexible processes combined with a potentially unbounded trace length make the simulation of all possible traces to an almost impossible task. On the other hand, a rigid maximum trace length for simulation prevents answering relevant questions such as *"Can this process instance be finished?"* since it is only possible to check whether the process instance can be finished within this defined scope. Apart from that, simulation is computationally and time intensive and cannot be performed for each aspect that needs to be checked during process validation. Therefore a more targeted simulation technique is needed that simulates only those parts that are relevant to assess a certain aspect. These aspects usually refer to frequent or particularly critical scenarios of the process. Formally, a *scenario* describes a (hypothetical) sequence of already executed activities in a process instance that has not yet been completed. Consequently, a scenario can be described by a trace. In scenario-based simulation, we either analyze the scenario or simulate possible continuations of the process instance (i.e., we play through the scenario and simulate how the process model covers this scenario). The simulation of a scenario can be reduced to a few repetitive core functionalities (CFs):

- check the validity of a trace (*CF1*),
- check if a scenario leads to a successful completion of the process instance (*CF2*),
- simulate continuations for a given scenario (*CF3*).

These CFs can be combined arbitrarily to perform more advanced analysis. In the following subsections, we describe these three CFs with their respective intended scope. We also show that CF2 outperforms classical approaches with regard to check process instance completion.

In the following, we assume that a declarative process model $P = (\mathcal{A}, \mathcal{T})$ has already been transformed into a finite state automaton $M_P = (\mathcal{A}, Q, q_0, \delta, F)$.

4.1 Trace Validity

A common task in validating a process model is to check whether a given trace t is valid, i.e., satisfies all constraints. If it does not, it is necessary to determine which constraints are violated. If the trace is mistakenly not valid, the process analyst receives indications which constraints may need to be adjusted. Moreover, in the case of a desired behavior, this analysis provides a well-understandable explanation for the behavior exhibited. This functionality can be realized by checking if the corresponding process automaton M_P accepts t, i.e. $\hat{\delta}(q_0, t) \in F$. For the constraint violation check, we consider each constraint $c \in \mathcal{T}$ as a separate process model $P_c = (\mathcal{A}, \{c\})$ and check whether t satisfies P_c.

4.2 Trace Completion

A second question is whether a process execution that has not yet been completed can still be completed. And if so, what are the valid continuations of minimal length? For answering this question, FSA based simulation has several advantages over classical simulation approaches: as long as the execution does not lead to a dead state, it is still possible to find a valid completion of the trace, i.e. a continuation that satisfies all constraints of the process model. Another advantage of using minimization techniques is the existence of a unique "dead state" (Hopcroft et al., 2007):

Theorem 1. *Let $M = (\Sigma, Q, q_0, \delta, F)$ be a minimized FSA. Then there is a unique state $q_d \in Q$:*

1. $q_d \notin F$
2. $\forall a \in \Sigma : \delta(q_d, a) = q_d$.

In other words, state q_d cannot be left for all symbols $a \in \Sigma$ and as $q_d \notin F$, i.e. q_d is not an accepting state, there is no possible path to an accepting state starting in state q_d. We will use this fact in our CFs to exclude certain execution traces.

However, if the execution reaches a dead state, there is no possibility for a valid continuation of the trace. In other words: a trace can be completed successfully, iff there is a path that leads to an accepting state. Otherwise, the trace cannot be completed successfully. To answer this question, we consider the automaton as a weighted graph $G = (V, E)$ whose nodes V are the states of M_P and whose edges E are the state transitions, with each edge $e \in E$ associated with the symbols for the respective transition as weights. This enables us to apply common graph algorithms for searching paths. For a given trace t, we start a modified breadth-first search from the node representing the state $\hat{\delta}(q_0, t)$, which returns the set

$\Gamma_{\hat{\delta}(q_0,t)}$ of all paths of minimal length l from $\hat{\delta}(q_0,t)$ to a node representing an accepting state. Our breadth-first search stops when it reaches either a node representing an accepting state or a dead state. In this context, minimal length means that an edge is traversed at most once. If we found such a path, we can assume that the trace can be completed successfully. Finally, we construct the traces (i.e. words) from the paths by replacing two consecutive nodes i and j with the weights of the edge that connects i and j. This procedure is exemplary illustrated in Fig. 1. To ensure minimal trace length, we avoid visiting a node multiple times. Therefore, we ignore recurrent edges as well as iterating cycles more than once. The corresponding procedure is shown in Algorithm 1. Note that the algorithm terminates if all nodes have been visited. This is always the case, since our automaton does not contain any unreachable states. In practical application, a concrete scenario can thus be modeled as a trace and then it can be checked whether this scenario leads to a successful process completion and which options exist for this. In the case that either paths are missing or unintentionally allowed, the model must be adjusted. The resulting set of traces can be analysed in various ways, for instance, whether a certain activity must always be executed.

Algorithm 1: *checkCompletion*

```
   Input: Process Automaton
          $P_M = (\mathcal{A}, Q, q_0, \delta, F)$, trace $t$
   Output: Set $S$ of completed traces of
           minimal length, if $t$ can be
           completed successfully, otherwise
           False
1  if $\hat{\delta}(q_0,t) \neq q_d$ then
2      $\Gamma_{\hat{\delta}(q_0,t)}$ ←breadth-first_search($\delta(q_0,t)$)
3      $S$ ← constructWords($\Gamma_{\hat{\delta}(q_0,t)}$)
4      return $S$
5  else
6      return False
7  end
```

4.3 Simulate *n* Steps

A third basic task simulates continuations of maximum length n for a trace t. Thus, we look for all valid traces and those that can still become valid (i.e. traces that are currently invalid but may eventually become valid again as more activities are executed) of length $\leq |t|+n$. This analysis tells us which activities can be executed next (or more generally in the next n steps) and which paths are feasible. Depending on the choice of n, we can perform a more specific analysis, such as a determination of the next feasible activity or a classical simulation of all paths up to length $|t|+n$.

Therefore, as in CF2, we transform the process automaton M_P into a weighted graph. Then we apply a modified breadth-first search of depth n, starting from the node representing the current state $\hat{\delta}(q_0,t)$, which outputs all paths ending in a node representing an accepting state. Finally, we construct all words from the found paths, as in CF2. Algorithm 2 shows the detailed procedure of this functionality.

Algorithm 2: *simulateNSteps*

```
   Input: Process Automaton
          $P_M = (\mathcal{A}, Q, q_0, \delta, F)$, trace $t$, number
          of steps $n$
   Output: Set $S$ of traces with length $\leq |t|+n$
           that are valid or can become valid
           again
1  $P_{\hat{\delta}(q_0,t)}$ ←breadth-first_search($\hat{\delta}(q_0,t), n$)
2  $S$ ← constructWords($P_{\hat{\delta}(q_0,t)}$)
3  return $S$
```

5 EVALUATION

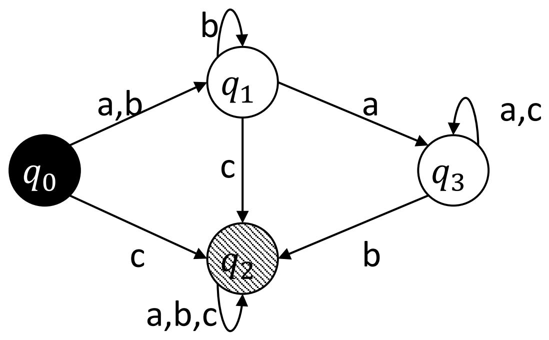
We implemented our approach as a prototype Java web application. The code and a runnable jar file can be accessed online[3]. This tool contains the CFs and all methods required to convert a declarative process model into the required automaton representation. Moreover, an intuitive graphical user interface allows it to be used by a wide range of users even without expert knowledge in declarative process management. To test the effectiveness of our approach, we perform a profound evaluation of three different aspects: *(i)* the theoretical worst-case runtime of our algorithms, *(ii)* the real-world runtime when used with real-life declarative process models from practice, and *(iii)* the benefits to the people involved in process model verification in terms of time savings, error reduction, and mental effort reduction.

5.1 Runtime Complexity

To determine the worst-case runtime of our approach, we determine the runtime complexity of the three core functionalities (CFs).

The runtime of CF1 depends only on the trace length n of the trace t that is checked for validity, and not on the process automaton M_P. The calculation

[3]https://github.com/NicolaiSchuetzenmeier/Scenario-based-Model-Checking



Paths	Derived words
$q_0 - q_2$	c
$q_0 - q_1 - q_2$	ac, bc
$q_0 - q_1 - q_3 - q_2$	aab, bab

Figure 1: Searching all paths from q_0 to q_2 of minimal length (i.e. without reflexive transitions) and deriving the corresponding words.

takes n steps because for each symbol of t the transition function of M_P must be evaluated. Hence, the runtime of CF1 is $O(n)$ (depending on the trace length n), i.e. linear.

In the case of CF2, finding minimal valid continuations of a trace t, we need to determine the runtime of three subsequent steps: *(i)* we need to process t such that the automaton represents the current state, *(ii)* find all paths of minimal length leading from the current state (i.e. $\hat{\delta}(q_0,t)$) to an accepting state, *(iii)* derive all words from the found paths. The runtime for the first step is identical to the runtime for CF1. For *(ii)* the corresponding graph $G = (V,E)$ to the process automaton M_P is traversed by a breadth-first search that has a time complexity of $O(|V|+|E|)$. The third part *(iii)* constructs all valid continuations of t from the found paths. Therefore, we denote m as the number of distinct activities that occur in the process model. In the worst-case, there are then $m^{2|V|}$ different continuations of t because in general any activity could be possible at any step (e.g. in a process model that accepts any trace). Since this CF avoids loops from being traversed more than once and also recurrent edges being traversed more than once, the length of the trace is limited to a maximum of $2|V|$. Consequently, the total runtime complexity (i.e. the sum of the complexity of the substeps) of the computation is exponential.

The runtime of CF3 differs only in the time required to construct the words from the found paths. While in CF2 the maximal trace length is limited to $2|V|$, we now have a user-defined arbitrary trace length of l.

Although two of the CFs have exponential runtimes, we will see in the next section (cf. Sec. 5.2) that the performance is nonetheless applicable and efficient for most real-world calculations, since the trace length and the size of the automaton are in a manageable range.

5.2 Empirical Evaluation

To evaluate the practical applicability of our approach, we conducted runtime experiments with the real-life event logs of the Business Process Intelligence Challenge (BPIC) extracted from the *4TU Center for Research Data*[4]. From these logs, we automatically created Declare models using the MINERful miner (Di Ciccio et al., 2015; Alman et al., 2020).

For our evaluation, we used the available BPIC event logs of the last five years, i.e., the log of the *BPIC 2017 (financial industry)*, *BPIC 2018 (fund process)*, *BPIC 2019 (purchase process)* and the *BPIC'20 (reimbursement process)*. The event logs of the *BPIC'20* contain sub-logs, which are labelled accordingly. For each event log, we mined a Declare model. As mining parameters, we chose the mining parameters as suggested in (Di Ciccio et al., 2017), namely a *support* factor of 75% (minimum number of cases in which a rule has to be fulfilled in), as well as *confidence* and *interest* factors of 75% (support scaled by the ratio of cases in which the activation occurs, respectively support scaled by the ratio of cases both the activation and reaction occur). An exception was made for the *BPIC 2017* event log, as these parameters yielded to many (several hundreds) constraints here - instead, the parameters of 95%, 95%, 95% for support, confidence and interest were chosen. Table 1 shows the sizes of the resulting Declare models and the corresponding calculated automatons. All obtained models and automatons can be found online[5]. Investigating the mined Declare models, we found that the two models for *BPIC'20(domestic)* and *BPIC'20(international)* accept exactly one trace. This is due to the fact that the mined constraints are too strict. Therefore, we excluded these two models from our evaluation. The mined process model for

[4]https://data.4tu.nl

[5]https://github.com/NicolaiSchuetzenmeier/Scenario-based-Model-Checking

BPIC'20(prepaid) was broken, i.e. it did not accept a single trace. Hence, this model was also omitted from our evaluation.

To evaluate the three CFs, we performed several calculations on the corresponding automatons. CF1 and CF2 were calculated within milliseconds for all mined process models and all trace lengths up to length $n = 20$. We chose this trace length because the average trace length of the traces in the event logs considered was significantly smaller. Hence, a length of $n = 20$ covers all sensible applications. The efficient runtime is obvious, as for these CFs only one trace has to be simulated in the corresponding automaton.

For CF3, we started with the empty trace for all mined models and computed all valid traces up to length $n = 20$. The corresponding runtimes in seconds and the number of found traces are shown in Table 2. Note that for all runtimes < 1 second, we used 1 as a default value. The experiments were run on a Windows 10 64 Bit system equipped with an Intel Core i7-7500U CPU @2.70GHZ 32GB memory and SSD drive. For reliable time measurements, we ensured that no parallel experiments were run on the workstation.

We observe that almost all computations - even with millions of traces (e.g. *BPIC 2019*) - take place within a few (milli)seconds, which underlines the practical applicability of our approach. *BPIC'20(request)* shows outlier behavior because there is exactly one accepted trace for each trace length (cf. Table 1). The relatively large runtimes for *BPIC 2018* and *BPIC'20(travelpermit)* are due to the fact that the corresponding process models are very flexible, i.e. they allow a very large number (couple of millions) of execution traces. This fact can be understood at automaton level regarding the attribute *#non-dead-transitions* in Table 2. This attribute describes the number of transitions that do not yield into the dead state, i.e. the transitions that are responsible for the number of accepted words. The corresponding automatons of *BPIC 2018* and *BPIC'20(travelpermit)* have the highest number of non-dead-transitions. For this reason, the exponential runtime (see Section 5.1) is severer than for the other event logs. Note that it was therefore not possible to calculate the accepted traces of *BPIC'20(travelpermit)* for trace lengths ≥ 16, which explains the missing values in Table 2. Since the average trace length in the original event log of *BPIC'20(travelpermit)* is much smaller ($\varnothing = 12.2$), our calculations up to length 14 are representative.

5.3 User Study

In the following, we briefly describe the overall procedure of the study and the rationale behind its construction. For a detailed description of materials, questionnaires, and instrumentation we refer the reader to the supplementary material[6]. To verify that a process model captures the desired behavior of an underlying process, the process model has to be completely understood by users. Especially when confronted with declarative models, users face challenges in understanding the logic defined by a set of constraints.

In our study, we evaluate the extent to which using the CFs with the proposed tool supports domain experts to verify declarative process models. For this purpose, we designed an experimental study in which participants perform tasks with and without using the tool. Thereby, the tasks refer to sub-steps that are performed during the validation of a declarative process model. We followed previous studies that evaluate the comprehensibility of process models from various perspectives with different focuses, e.g., (Fichtner et al., 2022; Figl et al., 2009; Jošt et al., 2016; Mendling et al., 2007). The comprehensibility of a process model is linked to its usability (Houy et al., 2014), while according to ISO 9241-11, *effectiveness* and *efficiency* are two key aspects of usability. Effectiveness indicates the ability of a participant to correctly solve a task. We measure this by the number of errors made on a task. Efficiency is derived from the resources (i.e. time and additional material) required by the user to solve the task. We measure this by stopping the time, asking to what extent the auxiliary material was used, and having the participants rate their mental effort.

Inspired by the results of related work, we formulate the following hypotheses:

- *Hypothesis 1:* When using a tool that offers the CFs, the success increases.
- *Hypothesis 2:* Subjects benefit from the tool support through the CFs regardless of their prior knowledge.
- *Hypothesis 3:* Subjects who initially work with the tool subsequently perform better when working without the tool than subjects who work on such tasks without the tool for the first time.

Study Setup, Materials, and Task Design. We have created two task sets S_1 and S_2 which are equal in structure and level of difficulty. A task set consists of a process model and three corresponding tasks 1, 2,

[6]https://github.com/NicolaiSchuetzenmeier/Scenario-based-Model-Checking

Table 1: Overview of considered Declare models and characteristics of the corresponding process automatons.

event log	#constraints	#activities	#states	#non-dead-transitions
BPIC 2017	154	9	14	20
BPIC 2018	178	10	21	55
BPIC 2019	14	4	5	8
BPIC'20(domestic)	32	5	7	5
BPIC'20(international)	116	9	11	9
BPIC'20(travelpermit)	70	9	13	37
BPIC'20(prepaid)	55	6	1	0
BPIC'20(request)	32	5	7	6

Table 2: Runtimes (in seconds) and number of calculated traces for mined process models in dependency of the trace length.

trace length n		BPIC17	BPIC18	BPIC19	BPIC20(travelpermit)	BPI20(request)
10	runtime	≈ 1	≈ 1	≈ 1	≈ 1	≈ 1
	#traces	0	0	2,304	9,140	6
12	runtime	≈ 1	≈ 1	≈ 1	4	≈ 1
	#traces	13	13	11,264	235,444	8
14	runtime	≈ 1	8	≈ 1	992	≈ 1
	#traces	153	537	53,248	5,543,860	10
16	runtime	≈ 1	59	≈ 1		≈ 1
	#traces	867	10,951	245,760	$> 10^6$	12
18	runtime	≈ 1	509	2		≈ 1
	#traces	3375	167,790	1,114,112	$> 10^6$	14
20	runtime	≈ 1	31,923	16		≈ 1
	#traces	10,382	2,242,933	4,980,736	$> 10^6$	16

Table 3: Prior knowledge and demographics of the study participants.

Variable	Scale	Mean/Count	SD/Percentage
Age	min / max / avg	24.0 / 69.0 / 37.21	13.93
Gender	male / female / diverse	19 / 10 / 0	65.52% / 34.48% / 0.0%
Highest level of education	primary / lower sec. / secondary / high school / university	0 / 0 / 4 / 3 / 22	0.0% / 0.0% / 13.79% / 10.34% / 75.86%
Working experience	min / max / avg	0.0 / 48.0 / 9.9	12.05
Prior knowledge	yes / no	18 / 11	62.07% / 62.07%
Read process models	yes / no	17 / 1	94.44% / 5.56%
Create process models	yes / no	11 / 7	61.11% / 38.89%
Read declarative process models	yes / no	6 / 12	33.33% / 66.67%
Create declarative process models	yes / no	4 / 14	22.22% / 77.78%
Years of exp. PM	min / max / avg	0.0 / 31.0 / 3.45	7.32
Years of exp. declarative PM	min / max / avg	0.0 / 28.0 / 2.31	6.7

and 3. 2 and 3 are divided into independent subtasks (2*a*, 2*b*, 3*a*, 3*b*, 3*c*). The (sub-) tasks correspond to the respective CFs and can be solved with them (see Table 4). In some cases, the tasks can be solved in different ways, i.e. the correct result can be achieved by different CFs and/or a combination of them. The process models used in the study are Declare models consisting of four and five constraints respectively. The comparatively small size of the models is motivated by three points: *(i)* this is the typical size of running examples in scientific literature, e.g. (Schützenmeier et al., 2021; Corea et al., 2022), *(ii)* a study in the form of a face-to-face interview should not exceed 45 minutes to achieve sufficient acceptance (Tscheulin and Helmig, 2004), *(iii)* our preliminary study has shown that larger declarative process models can no longer

Table 4: Subtasks of task sets S_1 and S_2 and the corresponding CFs that can be used for solving.

Subtask	Category	Core functionalities
1	Trace Validity	1
2a	Trace Completion	2, (3)
2b	Trace Completion	2
3a	Trace Simulation	3
3b	Trace Simulation	3
3c	Trace Simulation	3, (2)

be verified by people with little or no prior knowledge.

To avoid any bias (i.e. misinterpretation) due to implicit contextual knowledge, the process model describes an abstract process that is not inspired by a real process. Therefore, activities were named using only single letters. To avoid process-specific terms preventing participants with little or no prior knowledge from successfully completing a task, we replaced the terms *process model*, *constraint*, *alphabet*, and *trace* with more common terms *set of rules*, *rule*, *character*, and *character sequence*. For consistency, these terms have also been replaced in the corresponding tool. We also provide an explanation of the Declare constraints used in natural text, since their meaning cannot be assumed to be known. Participants (regardless of their prior knowledge) could refer to this note at any time during the study. For instance, to a given process model we asked in task $2a$ from task set S_1: *List all valid character sequences of length 4 that begin with "DE"*. The process models, all other tasks, and additional material used can be found in the supplementary material.

We would like to emphasize that this study does not evaluate the tool (i.e. its usability and graphical representation), but rather the usefulness of the identified CFs. The chosen study design is sufficient for this purpose: even when using the tool, participants must find a solution on their own, including identifying the appropriate CF in each case. Since the tool only provides these CFs, its impact is limited to making the problem-solving process more structured and goal-oriented. Thereby, its introduction of the CFs relieves the work that would otherwise have to be done manually. However, the facilitation of manual work has no effect on problem-solving skills. Thus, the study measures the extent to which the CFs support the verification of a process model. This is emphasized in our setting by the fact that we took no action to optimize the user interface.

Study Procedure. The study consists of three parts: *(i)* inquiring descriptive characteristics, *(ii)* solving a task set with or without tool support and rating mental effort, *(iii)* solving the other task set with/or without tool support and again rating mental effort. We randomly determine whether participants begin with or without the tool and which one of the two task sets is completed first.

First, the descriptive characteristics are collected during an interview to ensure complete coverage. This is followed by a brief introduction to the topic of the study. Here, the main terms (set of rules, rule, characters, character sequence) are explained and the subjects are given the opportunity to ask questions. The subjects are also informed that the aids (scratch sheet, overview of the rules and their explanation) can be used at any time.

As well as randomly determining which task set participants should start with and whether they start with or without the tool, the order of the three tasks within a task set is randomised. This decision avoids the occurrence of learning and order effects. In contrast, within a task, the order of the subtasks is fixed, i.e. subtask a) is processed first, then b), then c). The time of each subtask execution is recorded. We measure the number of errors indirectly by the points that can be achieved in a subtask. Points are awarded for each correct part of the answer, while missing parts brings no points, and errors are penalized with point deduction. The maximum point value to be achieved differs per subtask (from two to eight), while subject cannot achieve less than zero points. After each subtask execution, subjects receive situational feedback from the study conductor. Finally, after completing a task set, the study participants receive a hard-copy questionnaire to assess their perceived mental effort during task execution on a scale between zero and 220 (Zijlstra and Van Doorn, 1985).

Immediately before participants start working with the tool, they receive a brief introduction. The study conductor teaches different functionalities and shows how traces can be entered, how analyses can be started, and where the results are displayed. In case of technical problems with the tool, help is offered by the study conductor. However, this support is limited to questions about using the tool only and excludes answering questions about the task or its solution. Due to the study design, which aims to evaluate the usefulness of the CFs rather than the tool, this does not bias the results. After completing the tasks with the tool, the participant is asked whether he or she used the explanation of the constraints.

Results. Our study was conducted with 29 participants. Detailed descriptive statistics of the participants can be found in Table 3. We tested our hypotheses and were able to confirm all of them: For all subtasks except task $3c$, we observed significant time savings regardless of participants' prior knowledge (see Fig. 2, Fig. 5, Fig. 6). The error rate was also significantly reduced (as evidenced by a higher point score), especially for the first subtasks, which were almost always solved without errors when using the tool. However, in $3c$, where the participants had to find a character sequence in which a certain character occurs exactly twice, there is an outlier behavior. Since the process models in both task sets allow the

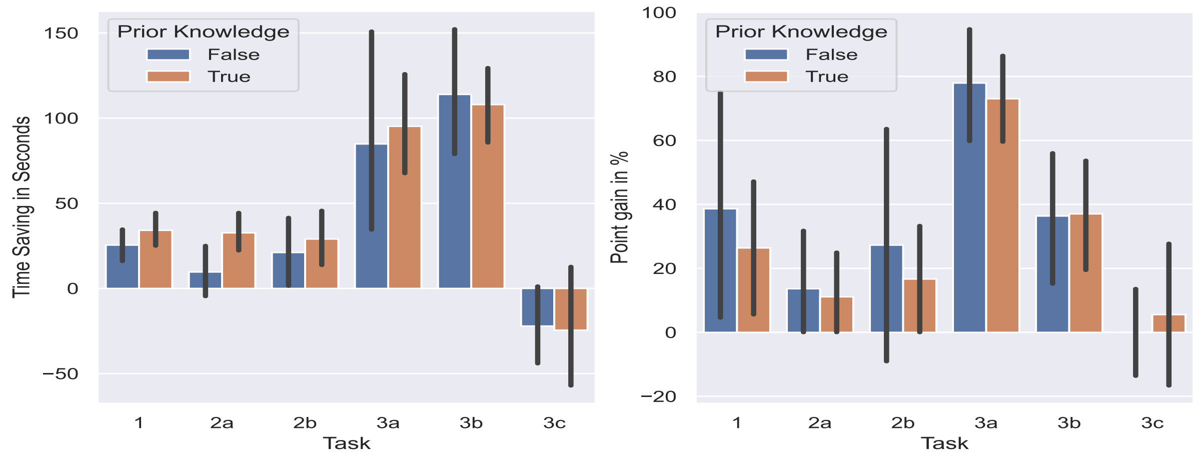


Figure 2: Time and point benefits by using the tool divided according to prior knowledge.

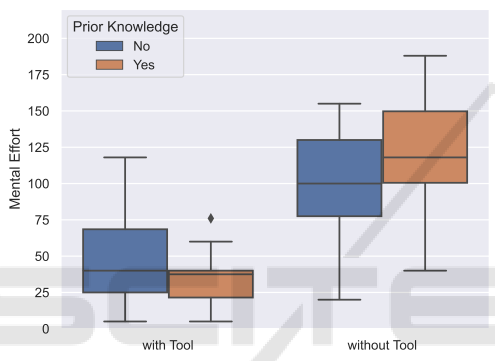


Figure 3: Mental effort for the processing of the subtasks with and without tool support, divided according to prior knowledge (scale from zero (low) to 220 (very high)).

occurrence directly as the first two characters of the sequence, an obvious first try led directly to the correct solution. Therefore, this task could be solved just as fast or in some cases even more quickly without tool support.

Regardless of their prior knowledge, subjects mostly rated the mental effort of working without tool support as strenuous (see Fig. 3). Compared to other studies (Fichtner et al., 2022; Orendt et al., 2016), a score of 150 (on the scale up to 220) is associated with great effort. In general, participants with no prior knowledge rated their mental effort significantly higher than subjects with sufficient prior knowledge. In both groups, the mental effort when using the tool shifted to an acceptable range (< 80). We also see that mental effort decreases more for subjects with prior knowledge when using the tool. This can be explained by the fact that the mental effort in this group of participants is caused more by the numerous steps to be performed and less by solving the problem itself. Moreover, we can observe that 24 out of 29 subjects indicated that they no longer needed explanations of the rules when working with the tool.

In testing hypothesis 3, we found that with the exception of subtasks 1 and $2b$, we observe a small learning effect, i.e. subjects who initially work with the tool perform eventually better when they work without the tool. The same effect is observed for the time required, where, with the exception of subtask $3b$, less time was required for manual solving. We have visualized this result in Fig. 4.

Threats for Validity. Explanatory power is limited due to small sample size. However, our sample size is in line with comparable studies in this field and is sufficient to check our hypotheses. Nevertheless, the small sample size combined with the imbalance of some descriptive features (e.g. occupation, highest education level) prevents a more fine-grained evaluation, e.g. to what extent the results differ between different occupational groups or age groups. Moreover, the non-optimized graphical user interface of the tool introduced a small bias, as some participants had overseen certain results. However, this does not affect the overall significance of our study, but only the strength of the observed effect.

6 CONCLUSION AND OUTLOOK

In this paper we presented a scenario-based model checking approach for declarative process models. We discussed essential scenarios and challenges a process modeler is confronted with throughout the phases of process modeling and model checking. Our approach, which is based on automata theory, is able to solve all the described challenges in a reasonable runtime, which we have proven in several scenarios calculated on real time applications. The results of an

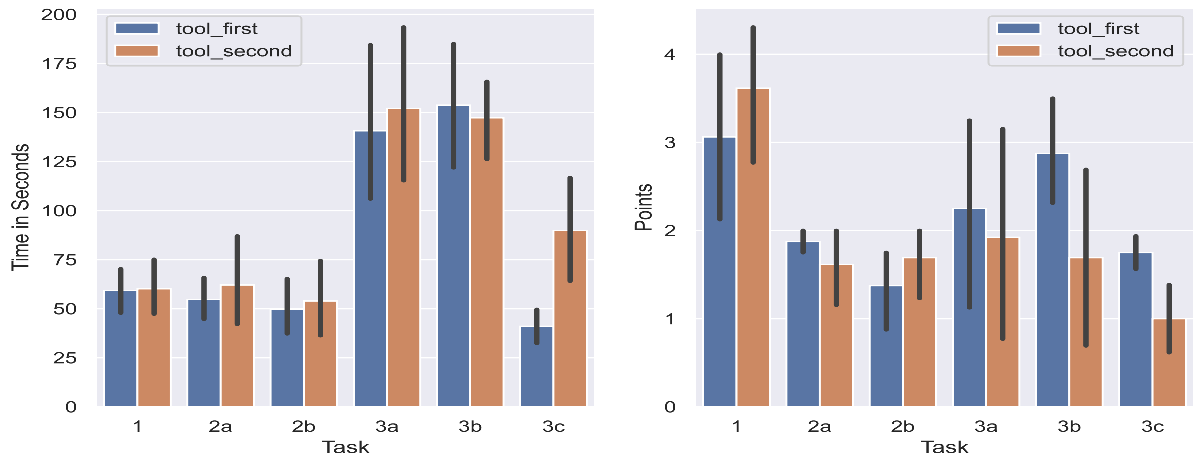


Figure 4: *Left:* Time spent manually completing subtasks when initially working with the tool compared to time spent when initially working without the tool. *Right:* Points achieved for the manual processing of the subtasks when initially working with the tool compared to the points achieved when initially working without the tool (more points are better and indicate a lower error rate).

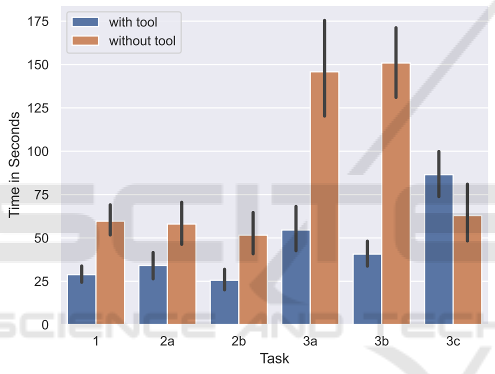


Figure 5: Time spent for processing the subtasks with and without tool support.

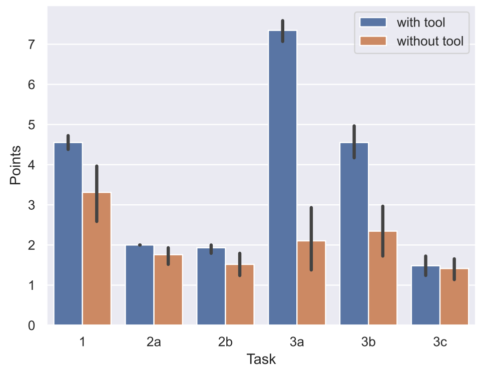


Figure 6: Points achieved per subtasks with and without tool support (more points are better and indicate a lower error rate).

elaborate user study underlined the practical usefulness of our approach.

In future work, we aim to adjust the graphical interface of the developed tool in order to enhance user friendliness and to create the opportunity to conduct meaningful user studies with a greater number of participants. We also want to focus on extending our approach to other process perspectives, e.g. the *data-flow perspective*, which describes data objects that are involved in a process and their occurring usages. In order to do so multi-perspective process modeling languages like MP-Declare will have to be investigated and included in our approach.

REFERENCES

Aalst, W. M. P. v. d. (2011). *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer.

Alman, A., Di Ciccio, C., Haas, D., Maggi, F. M., and Nolte, A. (2020). Rule mining with rum. In *2nd International Conference on Process Mining, Italy*.

Andaloussi, A., Buch-Lorentsen, J., Lopez, H. A., Slaats, T., and Weber, B. (2019a). Exploring the modeling of declarative processes using a hybrid approach. In *Proc. of 38th Int. Conf. on Conceptual Modeling 2019*, pages 162–170. Springer.

Andaloussi, A., Burattin, A., Slaats, T., Petersen, A., Hildebrandt, T., and Weber, B. (2019b). *Exploring the Understandability of a Hybrid Process Design Artifact Based on DCR Graphs*, pages 69–84. Springer Cham.

Bauer, A., Leucker, M., and Schallhart, C. (2011). Runtime verification for ltl and tltl. *ACM Trans. Softw. Eng. Methodol.*

Corea, C., Grant, J., and Thimm, M. (2022). Measuring inconsistency in declarative process specifications. In *BPM*.

Corea, C., Nagel, S., Mendling, J., and Delfmann, P. (2021). Interactive and minimal repair of declarative process models. In *Business Process Management Forum*.

Damaggio, E., Hull, R., and Vaculin, R. (2011). On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. *Information Systems*.

De Smedt, J., De Weerdt, J., Serral, E., and Vanthienen, J. (2016). Improving understandability of declarative process models by revealing hidden dependencies. In *Advanced Information Systems Engineering*. Springer.

De Smedt, J., De Weerdt, J., Serral, E., and Vanthienen, J. (2018). Discovering hidden dependencies in constraint-based declarative process models for improving understandability. *Inf. Syst.*, 74(Part).

Di Ciccio, C., Maggi, F. M., Montali, M., and Mendling, J. (2017). Resolving inconsistencies and redundancies in declarative process models. *Inf. Systems*, 64.

Di Ciccio, C., Schouten, M. H., de Leoni, M., and Mendling, J. (2015). Declarative process discovery with minerful in prom. In *BPM (Demos)*.

Dumas, M., La Rosa, M., Mendling, J., and Reijers, H. A. (2013). *Fundamentals of Business Process Management*. Springer, Berlin.

Fahland, D., Lübke, D., Mendling, J., Reijers, H., Weber, B., Weidlich, M., and Zugal, S. (2009). Declarative versus imperative process modeling languages: The issue of understandability. In *Enterprise, Business-Process and Information Systems Modeling*. Springer.

Fahland, D., Mendling, J., Reijers, H. A., Weber, B., Weidlich, M., and Zugal, S. (2010). Declarative versus imperative process modeling languages: The issue of maintainability. In *Business Process Management Workshops*. Springer.

Fichtner, M., Fichtner, U. A., and Jablonski, S. (2022). An experimental study of intuitive representations of process task annotations. In *International Conference on Cooperative Information Systems*. Springer.

Figl, K., Mendling, J., and Strembeck, M. (2009). Towards a usability assessment of process modeling languages. In *8th GI-Workshop Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten (EPK), CEUR-WS*, volume 554. Citeseer.

Haisjackl, C., Barba, I., Zugal, S., Soffer, P., Hadar, I., Reichert, M., Pinggera, J., and Weber, B. (2016). Understanding declare models: strategies, pitfalls, empirical results. *Software & Systems Modeling*, 15(2).

Hildebrandt, T. T., Mukkamala, R. R., Slaats, T., and Zanitti, F. (2013). Contracts for cross-organizational workflows as timed dynamic condition response graphs. *J. Log. Algebr. Program.*, 82(5-7).

Hopcroft, J. (1971). An n log n algorithm for minimizing states in a finite automaton. In Kohavi, Z. and Paz, A., editors, *Theory of Machines and Computations*.

Hopcroft, J., Motwani, R., and Ullman, J. (2007). *Introduction to Automata Theory, Languages, and Computation*. Pearson/Addison Wesley.

Houy, C., Fettke, P., and Loos, P. (2014). On the theoretical foundations of research into the understandability of business process models. In *European Conference on Information Systems*.

Jošt, G., Huber, J., Heričko, M., and Polančič, G. (2016). An empirical investigation of intuitive understandability of process diagrams. *Computer Standards & Interfaces*.

Käppel, M., Ackermann, L., Schönig, S., and Jablonski, S. (2021). Language-independent look-ahead for checking multi-perspective declarative process models. *Softw. Syst. Model.*, 20(5).

Mendling, J. (2009). *Empirical Studies in Process Model Verification*. Springer-Verlag, Berlin, Heidelberg.

Mendling, J., Reijers, H. A., and Cardoso, J. (2007). What makes process models understandable? In *International Conference on Business Process Management*, pages 48–63. Springer.

Montali, M., Pesic, M., van der Aalst, W. M. P., Chesani, F., Mello, P., and Storari, S. (2010). Declarative Specification and Verification of Service Choreographies. *ACM Transactions on the Web*, 4(1).

Nagel, S. and Delfmann, P. (2021). Investigating inconsistency understanding to support interactive inconsistency resolution in declarative process models. In *European Conference on Information Systems*.

Orendt, E. M., Fichtner, M., and Henrich, D. (2016). Robot programming by non-experts: Intuitiveness and robustness of one-shot robot programming. In *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*.

Pesic, M. (2008). *Constraint-based workflow management systems : shifting control to users*. PhD thesis, Industrial Engineering and Innovation Sciences.

Reichert, M. and Weber, B. (2012). *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer Berlin Heidelberg.

Schützenmeier, N., Käppel, M., Ackermann, L., Jablonski, S., and Petter, S. (2022). Automaton-based comparison of declare process models. *Software and Systems Modeling*.

Schützenmeier, N., Käppel, M., Petter, S., and Jablonski, S. (2021). Upper-bounded model checking for declarative process models. In *PoEM*. Springer.

Schützenmeier, N., Käppel, M., Petter, S., Schönig, S., and Jablonski, S. (2019). Detection of declarative process constraints in LTL formulas. In *EOMAS*, volume 366 of *LNBIP*. Springer.

Tscheulin, D. K. and Helmig, B. (2004). *A-Z*. Gabler, Wiesbaden.

Westergaard, M., Stahl, C., and Reijers, H. (2013). *UnconstrainedMiner : efficient discovery of generalized declarative process models*. BPM reports. BPMcenter. org.

Zijlstra, F. and Van Doorn, L. (1985). *The construction of a scale to measure perceived effort*. University of Technology.