Matching of Events and Activities - An Approach Using Declarative Modeling Constraints

Thomas Baier[1](✉), Claudio Di Ciccio[2], Jan Mendling[2], and Mathias Weske[1]

[1] Hasso Plattner Institute at the University of Potsdam,
Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, Germany
{thomas.baier,mathias.weske}@hpi.de
[2] Wirtschaftsuniversität Wien, Welthandelsplatz 1, 1020 Vienna, Austria
{claudio.di.ciccio,jan.mendling}@wu.ac.at

Abstract. Nowadays, business processes are increasingly supported by IT services that produce massive amounts of event data during the execution of a process. This event data can be used to analyze the process using process mining techniques to discover the real process, measure conformance to a given process model, or to enhance existing models with performance information. Mapping the produced events to activities of a given process model is essential for conformance checking, annotation and understanding of process mining results. In order to accomplish this mapping with low manual effort, we developed a semi-automatic approach that maps events to activities using the solution of a corresponding constraint satisfaction problem. The approach extracts Declare constraints from both the log and the model to build matching constraints to efficiently reduce the number of possible mappings. The evaluation with an industry process model collection and simulated event logs demonstrates the effectiveness of the approach and its robustness towards non-conforming execution logs.

Keywords: Process mining · Event mapping · Business process intelligence · Constraint satisfaction

1 Introduction

Organizations often support the execution of business processes with IT systems that log each step of participants or systems. Individual entries in such logs represent the execution of services, the submission of a form, or other related tasks that in combination realize a business process. To improve business processes and to align IT process execution with existing business goals, a precise understanding of processes execution is necessary. Using the event data logged by IT systems, process mining techniques help organizations to have a more profound awareness of their processes, in terms of discovering and enhancing process models, or checking the conformance of the execution to the specification [2]. Yet,

K. Gaaloul et al. (Eds.): BPMDS 2015 and EMMSAD 2015, LNBIP 214, pp. 119–134, 2015.
DOI: 10.1007/978-3-319-19237-6_8

these process mining techniques face an important challenge: the mapping of log entries produced by IT systems to the corresponding process activities in the process models has to be known. A discovered process model can only be fully understood when the presented results use the terminology that is known to the business analysts. However, such a mapping is often not existing because (i) the logging mechanism of IT systems captures fine-granular steps on a technical level and (ii) especially with legacy systems, the way in which events are recorded is rarely customizable. In fact, it is often a tedious task to reconstruct a mapping from cryptic names in a database to the activities in a process model.

In this paper, we offer means to help the analyst identify the mapping between a process model and events in an event log, in a semi-automated fashion. Defining such a mapping is generally hard to do manually, due to its combinatorial complexity. While there exist automatic techniques such as [5] or [6], these approaches have limitations on their applicability. For example, [5] requires event names that are processable using linguistic techniques, which are not always provided. The work presented in [6] overcomes this limitation, yet it is able to handle only 1:1 relations between events and activities, and requires pre-processing to handle 1:N relationships. The approach presented in this paper overcomes these limitations by using declarative constraints, in order to turn the matching problem into a constraint satisfaction problem. In this way, we not only lift the limitations of [6], but also drastically narrow down the effort for an analyst. Our approach also informs research into Declare, as it has been mainly used for the modeling of discovered processes from event logs [8,21]. Here, we also devise techniques to derive Declare constraints from an existing imperative process model, in order to reason about possible matches between events and activities, through the comparison of Declare constraints inferred from the event log and the process model.

The remainder of this paper is structured as follows. Section 2 starts by further illustrating the problem with an example and stating the formal definition of the mapping problem and the required formal concepts. Having laid the foundations, the matching technique is introduced in Section 3. In Section 4, the proposed approach is evaluated using an industry process model collection and simulated event logs. Related work is discussed in Section 5 and Section 6 concludes the work.

2 Preliminaries

This section gives a running example to illustrate the problem and introduces the main concepts used for the mapping approach. We will formally introduce the notion of a process model, an event log, and the used Declare rules.

2.1 Illustrating Example

Starting with an example, Table 1 shows an exemplary event log with 5 traces, which have been produced by an IT system supporting the order process depicted in Fig. 1. Obviously, it is not straightforward to interpret the given event log, because the event labels are cryptic database field names, which cannot be easily matched to the names of the activities in the process model. It can be seen that for some of the activities multiple events are being logged, while for others only one type of events can be observed. Looking at the events of the "Change order" activity, not necessarily all events belonging to an activity are generated when the activity is executed. Once the mapping is established as shown in Tab. 2, we can use the event log to check conformance between the model and the log. For example, we are able to detect that there is a case in the log, in which the customer has already been notified before the products were shipped. It is critical for organizations to detect, and accordingly react to such non-conforming behavior [2]. Moreover, using process discovery techniques, a new process model that reflects the actual as-is process, including all deviations, can be automatically created using the known terminology.

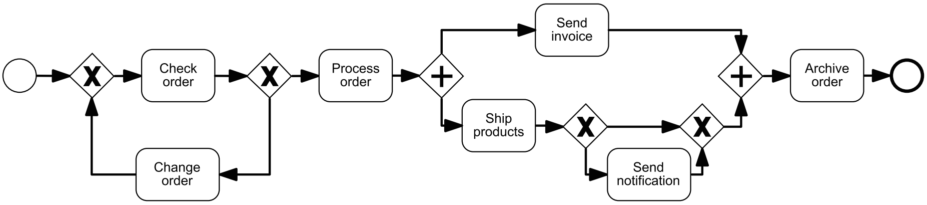


Fig. 1. Order process model in BPMN

Table 1. Event log (L) of order process (M)

	Label sequence
t_1	⟨ O_CHK_S, O_PR_S, O_PR_E, I_SM_E, P_SP_E, O_ARC_S, O_ARC_E ⟩
t_2	⟨ O_CHK_S, O_RC_SB, O_RC_E, O_CHK_S, O_PR_S, O_PR_E, P_SP_E, P_NOT_E, I_SM_E O_ARC_S, O_ARC_E ⟩
t_3	⟨ O_CHK_S, O_PR_S, O_PR_E, P_SP_E, P_NOT_E, I_SM_E, O_ARC_S, O_ARC_E ⟩
t_4	⟨ O_CHK_S, O_RC_SA, O_RC_E, O_CHK_S, O_PR_S, O_PR_E, P_NOT_E, I_SM_E, P_SP_E, O_ARC_S, O_ARC_E ⟩
t_5	⟨ O_CHK_S, O_PR_S, O_PR_E, P_SP_E, P_NOT_E, I_SM_E, O_ARC_S, O_ARC_E ⟩

Table 2. Mapping Map

Activity	**Event label**
Check order	O_CHK_S
Change order	O_RC_SA, O_RC_SB, O_RC_E
Process order	O_PR_S, O_PR_E
Send invoice	I_SM_E
Ship products	P_SP_E
Send notification	P_NOT_E
Archive order	O_ARC_S, O_ARC_E

2.2 Process Model and Event Log

Let S be a finite set of states, and A be a set of activities. A process model $M = (S, s_I, s_F, A, T)$ is a transition system that defines the allowed sequences of activity executions. Here, $T \in (S \times A \times S)$ is a transition relation modeling the allowed activities in a given state that result in a succeeding state. For example $(s_1, a, s_2) \in T$ implies that we can perform activity a in state s_1 and reach state s_2. A model has an initial state $s_I \in S$ and a final state $s_F \in S$. The function $\tau : M \rightarrow \mathcal{P}(A^*)$ captures all execution sequences starting with the initial state s_I and ending in the final state s_F that are allowed in T. Note that the number of execution sequences is infinite if the model contains loops. An execution sequence is also referred to as a process instance. For example, the model $M = (\{s_1, s_2, s_3\}, s_1, s_3, \{a, b, c\}, \{(s_1, a, s_2), (s_2, b, s_2), (s_2, c, s_3)\})$ has the execution sequences $\tau(M) = \{\langle a, c\rangle, \langle a, b, c\rangle, \langle a, b, b, c\rangle, \ldots\}$.

An IT system that supports process executions typically records events for each process instance in an event log [2]. Note that the relation of event instances to process instances might not be trivial in every practical setting. Yet, there exist approaches relating event instances to process instances that use event correlation (see, e.g., [22]). In this work, we therefore assume that this relation is already given. We abstract events as symbols of an alphabet E, which is often referred to as the set of event classes. Each process instance is represented as a sequence of events and also referred to as trace $t \in E^*$. For example, $\langle o, p, o, q\rangle$ is a trace with four consecutive events and three different event classes, $o, p, q \in E$. An event log L is a multiset of traces.

Confronted with a process model M and an event log L, the challenge is to derive the mapping relation between the activities $a \in A$ and the event classes $e \in E$. In this paper, we assume a 1:N relation as events are typically on a more fine-granuar level than activities. Thus, we are looking for the surjective function $Map : E \rightarrow A$ that maps event classes to their corresponding activities.

2.3 Declare

Having a process model and an event log, the approach presented in this paper will use Declare to describe their behavior. Declare [3] is natively a declarative process modeling language. It models workflows by means of temporal rules.[1] Such rules are meant to impose specific conditions on the occurrence of tasks in process instances. The rationale is, that every behavior in the process enactment is allowed, as long as it does not violate the specified rules. Due to this, declarative models are said to be "open", in contrast with the "closed" fashion of classical procedural models [21]. The Declare standard provides a predefined library of templates, listing default restrictions that can be imposed on the process control-flow. For instance, *Participation*(a) is a Declare rule expressed on activity a. It states that a must occur in every trace. *NotCoExistence*(a, b) constrains a and b, and imposes that a and b never occur together in the same trace. *Participation*(a) expresses a condition on the execution of a single activity. It is thus said to be an *existence rule*, as opposed to *relation rules*, such

[1] In literature, they are called "constraints". Nevertheless, we prefer not to make use of such term, in order to avoid the conflict with "constraints" in the context of constraint satisfaction problems (CSPs).

Table 3. Declare templates

Rule	Explanation	Cat.	Positive and negative examples			
Participation(a)	a occurs at least *once*	$\mathcal{C}_E$	✓ bcac	✓ bcaac	× bcc	× c
Init(a)	a is the *first* to occur	$\mathcal{C}_E$	✓ acc	✓ abac	× cc	× bac
End(a)	a is the *last* to occur	$\mathcal{C}_E$	✓ bca	✓ baca	× bc	× bac
Precedence(a, b)	b occurs only if preceded by a	$\mathcal{C}_{\overrightarrow{R}}$	✓ cacbb	✓ acc	× ccbb	× bacc
AlternatePrecedence(a, b)	Each time b occurs, it is preceded by a and no other b can recur in between	$\mathcal{C}_{\overrightarrow{R}}$	✓ cacba	✓ abcaacb	× cacbba	× acbb
ChainPrecedence(a, b)	Each time b occurs, then a occurs immediately beforehand	$\mathcal{C}_{\overrightarrow{R}}$	✓ abca	✓ abaabc	× bca	× bacb
CoExistence(a, b)	If b occurs, then a occurs, and viceversa	$\mathcal{C}_R$	✓ cacbb	✓ bcca	× cac	× bcc
Succession(a, b)	a occurs if and only if it is followed by b	$\mathcal{C}_{\overrightarrow{R}}$	✓ cacbb	✓ accb	× bac	× bcca
AlternateSuccession(a, b)	a and b if and only if the latter follows the former, and they alternate each other in the trace	$\mathcal{C}_{\overrightarrow{R}}$	✓ cacbab	✓ abcabc	× caacbb	× bac
ChainSuccession(a, b)	a and b occur if and only if the latter immediately follows the former	$\mathcal{C}_{\overrightarrow{R}}$	✓ cabab	✓ ccc	× cacb	× cbac
NotSuccession(a, b)	a can never occur before b	$\mathcal{C}_{\overrightarrow{R}}$	✓ bbcaa	✓ cbbca	× aacbb	× abb
NotCoExistence(a, b)	a and b never occur together	$\mathcal{C}_R$	✓ cccbbb	✓ ccac	× accbb	× bcac

as *NotCoExistence*(a, b), which indeed constrains pairs of activities. In the following, existence templates will be denoted as $\mathcal{C}_E$, and $\mathcal{C}_E(x)$ is the rule that applies template $\mathcal{C}_E$ to activity $x \in A$. Relation rules will instead be denoted as $\mathcal{C}_R$. $\mathcal{C}_R(x, y)$ applies template $\mathcal{C}_R$ to $x, y \in A$. *Precedence*(a, b) is the relation rule establishing that, if b occurs in the trace, then it must be *preceded* by at least one occurrence of a. In addition to relation rules, it imposes a condition on the *ordering* in which constrained activities can occur. Therefore, *Precedence*(a, b) falls under the category of *ordering relation* rules. Templates of such category will be denoted as $\mathcal{C}_{\overrightarrow{R}}$. $\mathcal{C}_{\overrightarrow{R}}(x, y)$ indicates an *ordering relation* rule applied to $x, y \in A$. In particular, $\mathcal{C}_{\overrightarrow{R}}(x, y)$ always specifies the order in which the occurrences of x and y are considered: x first, y afterwards (henceforth, *order direction*).

Table 3 lists the set of Declare rules that are mentioned in the remainder of the paper, along with the *category* (i.e., either $\mathcal{C}_E$, $\mathcal{C}_R$ or $\mathcal{C}_{\overrightarrow{R}}$) to which they belong. For every rule, two examples of complying traces and two examples of violating traces are provided. The complete list of rules can be found in [3].

Declare rules, when discovered from event logs, are usually associated to a reliability metric, namely *support* [9,21]. Support is a normalized value, ranging from 0 to 1, which measures to what extent traces are compliant with a rule. A support of 0 stands for a rule which is always violated. Conversely, a value of 1 is assigned to the support of rules which always hold true. According to the measurement introduced by the work of [9], the analysis of a trace $t_1 = \langle \mathsf{b}, \mathsf{a}, \mathsf{c}, \mathsf{b}, \mathsf{a}, \mathsf{b}, \mathsf{b}, \mathsf{c} \rangle$ would, e.g., lead to a support of 1 to $\mathit{Participation}(\mathsf{a})$, 0 to $\mathit{NotCoExistence}(\mathsf{a}, \mathsf{b})$, and 0.75 to $\mathit{Precedence}(\mathsf{a}, \mathsf{b})$, as 3 b's out of 4 are preceded by an occurrence of a. Considering an event log, which consists of t_1 and $t_2 = \langle \mathsf{c}, \mathsf{c}, \mathsf{a}, \mathsf{c}, \mathsf{b} \rangle$, the support of $\mathit{Participation}(\mathsf{a})$ and $\mathit{NotCoExistence}(\mathsf{a}, \mathsf{b})$ would remain equal to 1 and 0, respectively, whereas the support of $\mathit{Precedence}(\mathsf{a}, \mathsf{b})$ would be 0.8 (4 b's out of 5 are preceded by an occurrence of a). [9] provides further details on the computation of support values for each rule. This metric is usually utilized to prune out those rules which are associated to a value below a user-defined threshold.

3 Mapping Event Log and Process Model

This section introduces the approach for the mapping of events to given activities in a process model. The approach consists of three phases. The first one builds and solves a constraint satisfaction problem, to reduce the number of possible mappings between activities and events. The result of this phase is a set of potential event-activity mappings. During the second phase, the analyst is guided to select the correct mapping from the derived potential mappings. Finally, the last phase is used to automatically transform one or many event logs to reflect the activities in the process model. In the following sections, we will elaborate on each of the three phases.

3.1 Reduction of the Potential Set of Event–Activity Mappings

The first phase of our approach deals with the definition of a constraint satisfaction problem (CSP), which is used to restrain the possible mappings of events and activities. A CSP is a triple $CSP = (X, D, C)$ where $X = \langle x_1, x_2, \ldots, x_n \rangle$ is an n-tuple of variables with the corresponding domains specified in the n-tuple $D = \langle D_1, D_2, \ldots, D_n \rangle$ such that $x_i \in D_i$ [13]. $C = \langle c_1, c_2, \ldots, c_t \rangle$ is a t-tuple of constraints. We use predicate logic to express the constraints used in this paper. The set of solutions to a CSP is denoted as $S = \{S_1, S_2, \ldots, S_m\}$, where each solution $S_k = \langle s_1, s_2, \ldots, s_n \rangle$ is an n-tuple with $k \in 1..m$, $s_i \in D_i$ and such that every constraint in C is satisfied.

To build the CSP, first, the activities and event labels need to be mapped to the set of variables and their domains. Therefore, a bijective function $var : E \rightarrow X$ is defined, which assigns each event label to a variable with the natural numbers $1..|A|$ as domain. Furthermore, a bijective function $val : A \rightarrow 1..|A|$ is defined, which assigns each activity a natural number in the range from 1 to the number of activities. Table 4 and Table 5 show the mapping *var* and the mapping *val* respectively for the example given in Section 2.

Table 4. Mapping *var*

Event $e \in E$	O_CHK_S	O_RC_SA	O_RC_SB	O_RC_E	O_PR_S	O_PR_E	I_SM_E	P_SP_E	P_NOT_E	O_ARC_S	O_ARC_E
Variable $var(e) \in X$	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}

Table 5. Mapping *val*

Activity $a \in A$	Check order	Change order	Process order	Send invoice	Ship Products	Send notification	Archive order
Value $val(a) \in 1..\lvert A\rvert$	1	2	3	4	5	6	7

With the variables and domains defined, the solutions to the CSP reflect all possible mappings between events and activities, i.e., for n activities and m events there are potentially n^m solutions. For the example given in section 2.1 these are $7^{11} = 1,977,326,743$ possible mappings. Yet, this also includes solutions where not all activities are assigned to an event or solutions where all events are mapped to one single activity. As these solutions are not desired, we first restrict the set of solutions to those that assign each activity to at least one event. Note that we assume that the execution of each activity in the process model is being logged by the supporting IT system. Thus, those activities that are not recorded, are not considered in the processing. We assume that each event in the given log relates to exactly one activity in the process model, whereas one activity can relate to multiple events. Thus, we are using the `NVALUE` constraint, available in many constraint problem solvers (cf. [13]). This constraint ensures that each value in the domain of the variables is assigned at least once. Still, the complexity of the matching problem remains very high. In the following, we present an approach to tackle this complexity issue by combining the information available in the log with knowledge on the process model structure.

To be able to reduce the number of possible mappings, we look at Declare rules describing the behavior of event logs and process models as defined in Section 2.3. The techniques described in [9] are utilized to derive the described rules from event logs. In order to infer Declare rules from process models, we build upon the following assumption: if an event log is given, such that at least one trace is recorded for each legal path in the process model, then the Declare rules which are discovered out of such log, reflect the behavior of the original process model.[2] Hence, we can generate an event log from the process model using the simulation techniques described in [23], and thereafter apply the discovery algorithm of [9] to derive the Declare rules. We denote the set of all Declare rules inferred from the event log as $\mathcal{B}_L$, and the set of Declare rules discovered from the process model as $\mathcal{B}_M$. Next, we prune all discovered rules having a support lower than a given minimal threshold β. From our experience, a minimal support of $\beta = 0.9$ has turned out to be the most effective choice. Experimental findings

[2] Without loss of generality, loops can be unraveled and treated as an optional path that is traversable multiple times.

reported in the use cases of [8] confirm this assumption. Yet, the value of β can be redefined by the user if needed. From the ordering rules ($\mathcal{C}_{\overrightarrow{R}}$), only the rule with the highest support for each pair of events / activities is kept. In case there is no ordering rule with a support above β for a given pair of events / activities, we add the pair to the set of interleaving events / activities, denoted as $\mathcal{I}$.

Having the Declare rules from both the model and the event log as well as the set of interleaving pairs of events / activities, we can define a number of constraints to reduce the number of possible solutions of the CSP. These will be introduced in the following equations. For each equation, e_1, $e_2 \in E$ denote two different event classes, i.e., $e_1 \neq e_2$. In the same manner, a_1, $a_2 \in A$ denote two different activities, i.e., $a_1 \neq a_2$. The constraint introduced in Equation (1) ensures that events for which an *Init* rule exists, are only mapped to activities for which an *Init* rule exists. Equation (2) and 3 work in the same manner for *End* and *Participation* rules.

$$\mathit{Init}(e_1) \wedge (\mathit{Map}(e_1) = a_1) \implies \mathit{Init}(a_1) \tag{1}$$

$$\mathit{End}(e_1) \wedge (\mathit{Map}(e_1) = a_1) \implies \mathit{End}(a_1) \tag{2}$$

$$\mathit{Participation}(e_1) \wedge (\mathit{Map}(e_1) = a_1) \implies \mathit{Participation}(a_1) \tag{3}$$

The CSP constraints derived from *CoExistence* and *NotCoExistence* rules found in the event log, are similar to those derived from the existence rules, but look at pairs of events and activities. If two event classes that are co-existing (not co-existing) are matched to two different activities, these activities should also be co-existing (not co-existing).

$$\begin{aligned} \mathit{NotCoExistence}(e_1, e_2) \ &\wedge (\mathit{Map}(e_1) = a_1) \wedge (\mathit{Map}(e_2) = a_2) \\ &\implies \mathit{NotCoExistence}(a_1, a_2) \end{aligned} \tag{4}$$

$$\begin{aligned} \mathit{CoExistence}(e_1, e_2) \ &\wedge (\mathit{Map}(e_1) = a_1) \wedge (\mathit{Map}(e_2) = a_2) \\ &\implies \mathit{CoExistence}(a_1, a_2) \end{aligned} \tag{5}$$

In contrast to this, it cannot be assumed that *events* in an ordering relation necessarily map to *activities* in the same ordering relation. This is due to the fact that, for a pair of parallel activities, the log may contain a dominant ordering of the corresponding events. For instance in the order process example of Section 2.1, events I_SM and P_NOT_E are in *ChainSuccession* because P_NOT_E always occurs directly before I_SM. Yet, their corresponding activities "Send invoice" and "Send notification" are in interleaving order in the process model. Such a situation is still coherent, with respect to the model. Therefore, we specify in Equation (6) that if two events, for which an ordering rule exists, are mapped to two different activities, then these two activities either have to be in an ordering relation enforcing the same order direction, or this pair of activities has to be in the set of interleaving activities.

$$\mathcal{C}_{\overrightarrow{R}}(e_1, e_2) \wedge \mathit{Map}(e_1) = a_1 \wedge \mathit{Map}(e_2) = a_2 \implies \mathcal{C}_{\overrightarrow{R}}(a_1, a_2) \vee (a_1, a_2) \in \mathcal{I} \tag{6}$$

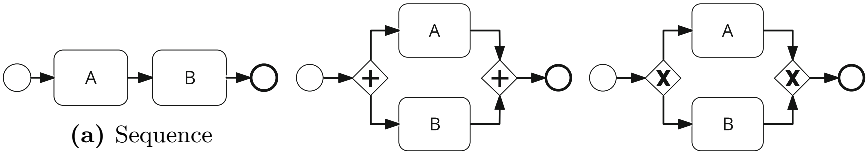
Regarding the pairs of events for which no ordering rule exceeds β, Equation (7) ensures that if a pair of interleaving events is mapped to a pair of activities, these activities also have to be interleaving.

$$(e_1, e_2) \in \mathcal{I} \wedge (Map\,(e_1) = a_1) \wedge (Map\,(e_2) = a_2) \implies (a_1, a_2) \in \mathcal{I} \qquad (7)$$

Having the constraint definitions in equations 1-7, we add a constraint $c_i, i \in 1..|\mathcal{B}_L|$ for each Declare rule derived from the event log to the CSP. For example, constraint $Init(O\_CHK\_S)$ is derived from the event log. In the set of inferred Declare constraints from the process model there is only one *Init* rule, namely *Init*(*Check order*). Using the mappings defined in Table 4 and Table 5 we can derive the corresponding constraint for the CSP: $c_1 \equiv x_1 = 1$. Having defined all constraints, the CSP can be solved to retrieve all possible mappings. If the CSP returns multiple solutions, the analyst has to choose the correct one. The next section shows how the analyst is supported in this selection.

3.2 Selection of the Correct Event–Activity Mapping

The previous section introduced the approach for automatic matching of event labels and activities. This section discusses why there are often multiple solutions to the defined constraint satisfaction problem and introduces means to guide the user through the set of potential mappings returned by the CSP solver.



(a) Sequence

(b) Concurrency

(c) Choice

Fig. 2. Process model fragments leading to multiple solutions

Consider the trace $t_1 = \langle k, l, m, n \rangle$ and the simple sequence of activities a and b shown in Fig. 2a. When matching t_1 and the sequence model, the corresponding CSP returns three solutions. In all three solutions k is matched to a, and n is matched to b. For l and m it cannot be said whether they belong to a or b without further knowledge. It may be that both belong to A, or both belong to b, or l belongs to a and m belongs to b. The only mapping that can be excluded, is that l belongs to b and m belongs to a at the same time. If we want to match t_1 to the model shown in Fig. 2b, actually every combination of mappings is possible, besides those where all events are mapped to only one of the activities. For the matching with the process model depicted in Fig. 2c, we add the trace $t_2 = \langle p, q, r, s \rangle$. In this case the CSP returns two solutions: Either k, l, m, n belong to activity a and the rest to b, or the other way around.

Such ambiguous mappings, i.e., cases in which the CSP has multiple solutions, cannot be automatically resolved and require a domain expert to decide the mapping for the concerned events and activities. Nonetheless, this decision can be supported by the mapping approach. To aid the analyst with the disambiguation of multiple potential mappings, we introduce a questioning approach, which is inspired by the work of La Rosa et al. [24], in which the user is guided through the configuration of a process model using a questionnaire procedure. The analyst is presented one event label at a time, along with the possible activities to which this event label can be mapped. Once the analyst decides which of the candidate activities belongs to the event label, this mapping is converted into a new constraint that is added to the CSP. Consecutively, the CSP is solved again. In case there are still multiple solutions, the analyst is asked to make another decision for a different event label. This procedure is repeated until the CSP yields a single solution. The goal is to pose as few questions to the analyst as possible. To achieve this goal, we look into all solutions and choose the event label that is assigned to the highest number of different activities.

3.3 Transformation of the Event Log

Having defined the procedure to build a CSP and iteratively resolved any ambiguities, the next step is to use the selected solution of the CSP as mapping Map to transform the event log. Mapping Map is used to iterate over all traces in the event log and replace each event e_i with the activity returned by $Map(e_i)$. This resulting event log, where each event carries the label of its corresponding activity, is processed using the activity clustering approach described in [4] in order to correctly reflect activity instances. The transformed event log can then be used as input for any process mining technique.

4 Evaluation and Discussion

4.1 Evaluation

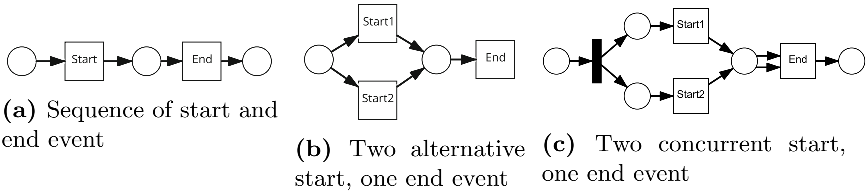
For the purpose of evaluation, the approach presented in this paper was implemented as a plug-in in the process mining framework ProM[3]. The Petri net notation has been chosen as modeling language for the implementation of the approach, because it has well-defined semantics and can be verified for correctness [1]. Furthermore, most of the common modeling languages, as e.g. BPMN and EPC, can be transformed into Petri nets [20]. As solver for the constraint satisfaction problem, the java library CHOCO[4] has been used.

To evaluate our approach with real life business processes, we used the *BIT process library, Release 2009*, which has been analyzed by Fahland et al. in [12] and is openly available to academic research. The process model collection contains models of financial services, telecommunications, and other domains. First, the models were transformed into Petri nets and only 1-bounded models that are free of life locks and deadlocks, and do not contain disconnected activities, have

[3] See http://processmining.org

[4] See http://www.emn.fr/z-info/choco-solver/

been kept. This restriction is due to the fact that models with such characteristics cannot be simulated. Moreover, some of the larger process models needed to be filtered out, as the resulting CSP could not be solved by the CSP solver due to memory shortage. This is mainly a limitation of the used CSP solver. Yet, for most of these processes, there is little behavioral distinction between activities or pairs of activities, e.g. all activities are interleaving to each other. Thus, it is not possible to match these processes without further knowledge. After the filtering step, 595 models remained with which we tested our approach. For these process models, event logs were generated by simulating the process activities' enactment through event generators. Such event generators followed the patterns (event models) illustrated in Fig. 3. Figure 3a shows a simple model with one start and one end transition, demonstrating a typical pattern found in many systems. For each activity that is assigned to this event model, a start and an end transition are generated for each execution of that activity. The second event model, depicted in Fig. 3b, generates for each execution either an event "Start1" or an event "Start2" and always an end event. Thus, there are two alternative starts for such an activity, e.g. it could be started by an incoming mail or by a telephone call. The event model presented in Fig. 3c also has two different start transitions, but in contrast to the model in Fig. 3b, both start events always occur with no restriction on their order. For the simulation of the process models, each activity is randomly assigned to one of these three event models, or it is left as is, generating only a single event. All generated event logs contain 1.000 traces and are limited to 1.000 events per trace as a stop condition for process models containing loops.



(a) Sequence of start and end event

(b) Two alternative start, one end event

(c) Two concurrent start, one end event

Fig. 3. Different event models used to generate events

In reality, event logs rarely completely comply to the defined process models due to noise and misbehavior. Thus, we generated for each process model five sets of simulated logs, in which we randomly inserted noise by shuffling, duplicating and removing events for a different percentage of traces. Figure 4 shows the results of this experiment. For 22 % of the processes, the mapping between events and activities can be established without asking any question, regardless the amount of noise injected. Looking at logs with no noise or where only 25 % of the traces contain noise, another 17 % of the processes require only one or two questions. With more noise in the event logs, this number continuously shrinks and more questions are needed for these processes. Yet, Fig. 4 shows that even

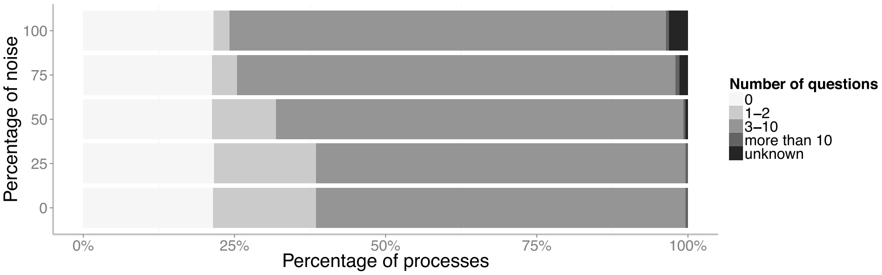


Fig. 4. Number of necessary questions with respect to noise

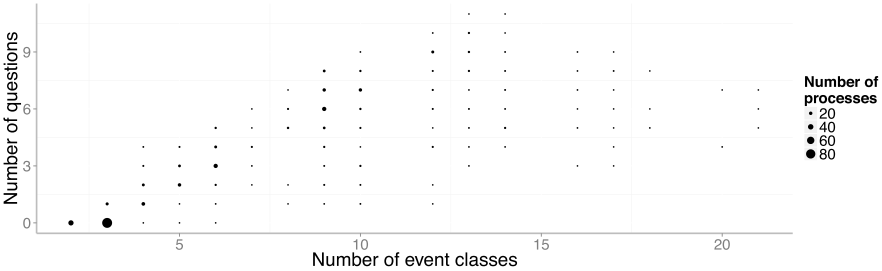


Fig. 5. Number of necessary questions with respect to number of event classes

when all traces contain noise, 97 % of the processes require less than 11 questions to build the correct mapping. Furthermore, it can be seen that with increasing noise a few processes cannot be processed by the CSP solver due to resource problems. This is mainly an implementation issue of the used CSP solver.

Figure 5 depicts the number of necessary questions with respect to the number of different events in a log without any noise. While one can see that there is a slight trend towards more questions with increasing numbers of event classes, it can also be seen that this trend also reverses for larger numbers of event classes. For example, it can be seen that there are processes with 20 different events, requiring only 5 questions. Thus, the number of required questions is rather independent of the number of activities. In fact, it depends on the structure of the process model and how well activities and pairs of activities can be distinguished from each other by their behavior.

4.2 Discussion

In the light of our experimental results, the approach turns out to be promising, especially with regards to resilience to noise. It requires in most of the cases only little manual intervention. Still, there are some processes that could not be handled, mainly due to massive parallelism and resulting memory shortage. Future work should investigate how these processes can be handled or, at least, automatically discovered. Moreover, it might be beneficial to combine this approach with the linguistic matching presented in [5], for cases in which the event labels carry more useful information than cryptic database field names. Finally, it is our plan to investigate, how the approach can be extended to support N:M relations, namely cases in which a single event class can be related to multiple activities – e.g. events representing shared functionalities. In the N:M case, the already very large search space for the matching problem grows drastically and other techniques might be necessary to handle this. Yet, event logs containing shared functionalities could be handled with the approach presented in this paper using preprocessing that essentially removes such events. If applicable, the approach presented in [5] could be used for the detection of shared functionalities.

5 Related Work

Related research can be subdivided into approaches working on event logs and approaches working on process models. Looking at approaches focusing on event logs, there are several approaches aiming at the abstraction of events to activities. Günther et al. introduce in [14] an approach that clusters events to activities using a distance function based on time or sequence position. Due to performance issues with this approach, a new means of abstraction on the level of event classes is introduced by Günther et al. in [16]. These event classes are clustered globally based on co-occurrence of related terms, yielding better performance but lower accuracy. A similar approach introducing semantic relatedness, N:M relations, and context dependence is defined by Li et al. in [19]. Another approach that uses pattern recognition and machine learning techniques for abstraction is introduced by Cook et al. in [7]. Together with the fuzzy miner, Günther and van der Aalst present an approach to abstract a mined process model by removing and clustering less frequent behavior [15]. While all these approaches aim at a mapping of events to activities, they are designed to automatically construct activities and not to match events to activities that have already been defined a-priori. In [5] and [6], we introduced approaches that aim at the mapping of events to pre-defined activities. Nevertheless, the approach in [5] still required more manual work as the precision of matchings is not sufficiently high. In contrast, the approach presented in this paper requires only very little manual effort to match events to pre-defined activities. The approach presented in [6] only works with 1:1 relations between events and activities and requires pre-processing for 1:N relations. Furthermore, it is only able to capture behavior from traces that can be replayed on the model. This is resolved by the work of this paper.

Another branch of related approaches working on event logs are those dealing with event correlation to group events belonging to the same process instance, as e.g. the work by Perez et al. in [22]. Yet, these approaches work on a more coarse-grained level as they focus on the relation to process instances rather than to activities. In fact, we assume that the correlation of events to process instances is either already given, or can be established by an approach like [22].

Our work is also related to automatic matching for process models. While matching has been partially addressed in various works on process similarity [10], there are only a few papers that cover this topic as their major focus. The work on the ICoP framework defines a generic approach for process model matching [25]. This framework is extended with semantic concepts and probabilistic optimization in [17,18]. Further, general concepts from ontology matching are adopted in [11]. The implications of different abstraction levels for finding correspondences is covered in [26]. However, all these works focus on finding matches between two process models, not between events and activities.

Our approach adopts MINERful [9] for the computation of constraints' support. MINERful is a declarative process miner, based on a two-phase technique. During the first step, it creates a so-called knowledge base, containing statistics about the occurrences of events in the log. The second step computes the support for constraints by querying such knowledge base. It is proven to be among the fastest Declare miners [9].

6 Conclusion

In this paper we introduce a novel technique for the mapping of events to activities, which can be used as a preprocessing step to enable business process intelligence techniques (e.g., process mining). The approach uses Declare rules derived from existing business process models and from event logs generated by IT systems, to establish a connection between conceptual process models and operational execution data. The key contribution of this approach is the establishment of a relation between events and a given set of activities in a process model using behavioral knowledge captured by Declare rules. Thereby, also 1:N relations can be handled. As shown in the evaluation in Section 4, the newly introduced matching technique performs well and requires little manual intervention. It also reveals to be robust towards noise.

References

1. van der Aalst, W.M.P.: Verification of workflow nets. In: Azéma, P., Balbo, G. (eds.) ICATPN, LNCS, vol. 1248, pp. 407–426. Springer (1997)
2. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes, 1st edn. Springer (2011)
3. van der Aalst, W.M.P., Pesic, M.: DecSerFlow: Towards a Truly Declarative Service Flow Language. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 1–23. Springer, Heidelberg (2006)
4. Baier, T., Mendling, J.: Bridging Abstraction Layers in Process Mining: Event to Activity Mapping. In: Nurcan, S., Proper, H.A., Soffer, P., Krogstie, J., Schmidt, R., Halpin, T., Bider, I. (eds.) BPMDS 2013 and EMMSAD 2013. LNBIP, vol. 147, pp. 109–123. Springer, Heidelberg (2013)
5. Baier, T., Mendling, J., Weske, M.: Bridging abstraction layers in process mining. Information Systems **46**, 123–139 (2014)
6. Baier, T., Rogge-Solti, A., Weske, M., Mendling, J.: Matching of Events and Activities - An Approach Based on Constraint Satisfaction. In: Frank, U., Loucopoulos, P., Pastor, Ó., Petrounias, I. (eds.) PoEM 2014. LNBIP, vol. 197, pp. 58–72. Springer, Heidelberg (2014)

7. Cook, D.J., Krishnan, N.C., Rashidi, P.: Activity discovery and activity recognition: A new partnership. IEEE T. Cybernetics **43**(3), 820–828 (2013)
8. Di Ciccio, C., Mecella, M.: Mining artful processes from knowledge workers' emails. IEEE Internet Computing **17**(5), 10–20 (2013)
9. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Trans. Manage. Inf. Syst. 5(4), 24:1–24:37 (2015)
10. Dijkman, R.M., Dumas, M., van Dongen, B.F., Käärik, R., Mendling, J.: Similarity of Business Process Models: Metrics and Evaluation. Information Systems **36**(2), 498–516 (2011)
11. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer (2007)
12. Fahland, D., Favre, C., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Analysis on demand: Instantaneous soundness checking of industrial business process models. Data & Knowledge Engineering **70**(5), 448–466 (2011)
13. Freuder, E., Mackworth, A.: Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2, ch. Constraint satisfaction: An emerging paradigm, pp. 13–27. Elsevier (2006)
14. Günther, C.W., van der Aalst, W.M.P.: Mining activity clusters from low-level event logs. In: BETA Working Paper Series. vol. WP 165. Eindhoven University of Technology (2006)
15. Günther, C.W., van der Aalst, W.M.P.: Fuzzy Mining – Adaptive Process Simplification Based on Multi-perspective Metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 328–343. Springer, Heidelberg (2007)
16. Günther, C.W., Rozinat, A., van der Aalst, W.M.P.: Activity Mining by Global Trace Segmentation. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 128–139. Springer, Heidelberg (2010)
17. Klinkmüller, C., Weber, I., Mendling, J., Leopold, H., Ludwig, A.: Increasing Recall of Process Model Matching by Improved Activity Label Matching. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 211–218. Springer, Heidelberg (2013)
18. Leopold, H., Niepert, M., Weidlich, M., Mendling, J., Dijkman, R., Stuckenschmidt, H.: Probabilistic Optimization of Semantic Process Model Matching. In: Barros, A., Gal, A., Kindler, E. (eds.) BPM 2012. LNCS, vol. 7481, pp. 319–334. Springer, Heidelberg (2012)
19. Li, J., Bose, R.P.J.C., van der Aalst, W.M.P.: Mining Context-Dependent and Interactive Business Process Maps Using Execution Patterns. In: Muehlen, M., Su, J. (eds.) BPM 2010 Workshops. LNBIP, vol. 66, pp. 109–121. Springer, Heidelberg (2011)
20. Lohmann, N., Verbeek, E., Dijkman, R.M.: Petri net transformations for business processes - a survey. Petri Nets and Other Models of Concurrency **2**, 46–63 (2009)
21. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient Discovery of Understandable Declarative Process Models from Event Logs. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 270–285. Springer, Heidelberg (2012)
22. Pérez-Castillo, R., Weber, B., de Guzmán, I.G.R., Piattini, M., Pinggera, J.: Assessing event correlation in non-process-aware information systems. Software and System Modeling **13**(3), 1117–1139 (2014)
23. Rogge-Solti, A., Weske, M.: Prediction of Remaining Service Execution Time Using Stochastic Petri Nets with Arbitrary Firing Delays. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 389–403. Springer, Heidelberg (2013)

24. La Rosa, M., Lux, J., Seidel, S., Dumas, M., ter Hofstede, A.H.M.: Questionnaire-driven Configuration of Reference Process Models. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 424–438. Springer, Heidelberg (2007)
25. Weidlich, M., Dijkman, R., Mendling, J.: The ICoP Framework: Identification of Correspondences between Process Models. In: Pernici, B. (ed.) CAiSE 2010. LNCS, vol. 6051, pp. 483–498. Springer, Heidelberg (2010)
26. Weidlich, M., Dijkman, R., Weske, M.: Behaviour Equivalence and Compatibility of Business Process Models with Complex Correspondences. ComJnl (2012)