Article type: Focus Article

Replaying History on Process Models for Conformance Checking and Performance Analysis

Wil van der Aalst, Arya Adriansyah, and Boudewijn van Dongen

Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, The Netherlands.

Keywords

process mining, conformance checking, automated process discovery, Petri nets, business intelligence

Abstract

Process mining techniques use event data to *discover* process models, to *check the conformance* of predefined process models, and to *extend* such models with information about bottlenecks, decisions, and resource usage. These techniques are driven by observed events rather than hand-made models. Event logs are used to learn and enrich process models. By replaying history on the model, it is possible to establish a *precise relationship between events and model elements*. This relationship can be used to check conformance and to analyze performance. For example, it is possible to diagnose deviations from the modeled behavior. The severity of each deviation can be quantified. Moreover, the relationship established during replay and the timestamps in the event log can be combined to show bottlenecks. These examples illustrate the importance of maintaining a proper alignment between event log and process model. Therefore, we elaborate on the realization of such alignments and their application to conformance checking and performance analysis.

Introduction

Process mining is an emerging research discipline that sits between computational intelligence and data mining on the one hand, and process modeling and analysis on the other hand (1). Starting point for process mining is an *event log*. All process mining techniques assume that it is possible to *sequentially* record *events* such that each event refers to an *activity* (i.e., a well-defined step in the process) and is related to a particular *case* (i.e., a process instance). Event logs may store additional information such as the *resource* (i.e., person or device) executing or initiating an activity, the *timestamp* of an event, or *data elements* recorded with an event (e.g., the size of an order). Event logs can be used to discover, monitor and improve processes based on facts rather than fiction. There are three main types of process mining:

- *Discovery*: take an event log and produce a model without using any other a-priori information. There are dozens of techniques to extract a process model from raw event data. For example, the classical α algorithm is able to discover a Petri net by identifying basic process patterns in an event log (4). For many organizations it is surprising to see that existing techniques are indeed able to discover real processes based on merely example executions recorded in event logs. Process discovery is often used as a starting point for other types of analysis.
- *Conformance*: an existing process model is compared with an event log of the same process. The comparison shows where the real process deviates from the modeled process. Moreover, it is possible to quantify the level of conformance and differences can be diagnosed. Conformance checking can be used to check if reality, as recorded in the log, conforms to the model and vice versa. There are various applications for this (compliance checking, auditing, Six Sigma, etc.) (25).
- *Enhancement*: take an event log and process model and extend or improve the model using the observed events. Whereas conformance checking measures the alignment between model and reality, this third type of process mining aims at changing or extending the a-priori model. For instance, by using timestamps in the event log one can extend the model to show bottlenecks, service levels, throughput times, and frequencies (1).

Process mining provides a bridge between data mining (including computational intelligence, machine learning, and knowledge discovery) and model-driven process management (process modeling, business process management, simulation, and verification). Data mining techniques (16) tend to search for relatively simple patterns in large datasets. These patterns may be association rules, decision trees, clusters, frequent sequences, etc. However, patterns discovered through traditional data mining techniques do not describe end-to-end processes involving concurrency, resource sharing, etc. Model-driven process management techniques (31) focus on such end-to-end processes but do not use event data, i.e., models are typically made by hand and are not related to the actual observed behavior. Only in recent years it has become possible to bridge this gap (1).

Over the last decade, event data have become readily available and process mining techniques have matured. Moreover, managements trends related to process improvement, e.g., Six Sigma, TQM (Total Quality Management), CPI (Continuous Process Improvement), and CPM (Corporate Performance Management) can benefit from process mining (1; 18). Whereas these managements trends aim at improving operational performance, e.g., reducing flow time and defects, organizations are also putting more emphasis on corporate governance, risk, and compliance. Major corporate and accounting scandals including those affecting Enron, Tyco, Adelphia, Peregrine, and WorldCom have fueled interest in more rigorous auditing practices. Legislations such as the Sarbanes-Oxley Act (SOX) of 2002 and the Basel II Accord of 2004 were enacted in response to such scandals. The recent financial crisis also underscores the importance of verifying that organizations operate "within their boundaries". Process mining techniques offer a means to more rigorously check compliance and ascertain the validity and reliability of information about an organization's core processes (1; 18).

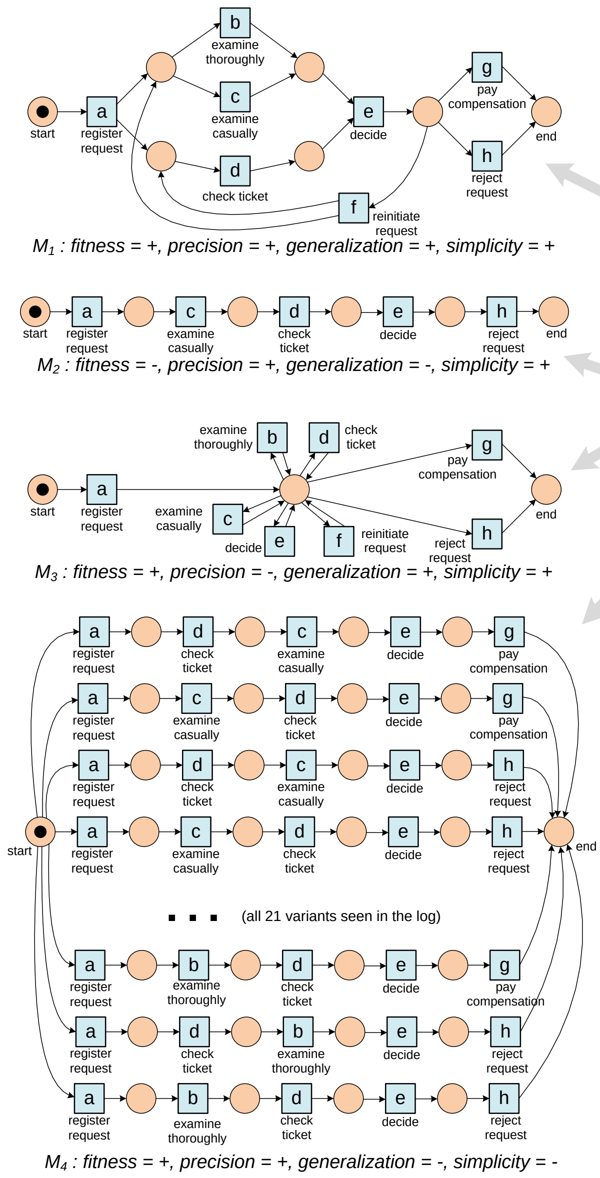
The right-hand side of Fig. 1 shows an abstraction of an event log containing 1391 cases, i.e., instances of some reimbursement process. Each case is represented by a trace of events and each event corresponds to an activity executed for a specific case. For example, Fig. 1 shows that there are 455 process instances having a trace $acdeh$. Activities are represented by a single character: a = *register request*, b = *examine thoroughly*, c = *examine casually*, d = *check ticket*, e = *decide*, f = *reinitiate request*, g = *pay compensation*, and h = *reject request*. Hence, trace $acdeh$ models a reimbursement request that was rejected after a registration, examination, check, and decision step. 455 cases followed this path consisting of five steps, i.e., the first line in the table corresponds to $455 \times 5 = 2275$ events. The whole log consists of 7539 events.

The event log in Fig. 1 only shows activity names. As mentioned before, events may have much more properties. For example, the a event of one of the 455 cases following $acdeh$ may refer to the name of the customer, the employee registering the request, the timestamp, etc. Although the activity name is just one of many potential attributes of an event, we use the more compact representation shown in Fig. 1 for clarity.

The α algorithm (4) mentioned earlier constructs process model M_1 from the event log shown in Fig. 1. The model is represented as a Petri net and shows that all cases start with a (register request) and end with g (pay compensation) or h (reject request). The examination (i.e., b or c) is concurrent with the checking of the ticket (d). Subsequently, a decision is made (e) with three possible outcomes. One of these three outcomes is the reinitiation of the request (f) which triggers a new examination and check. Model M_1 describes the observed behavior well, e.g., M_1 is able to replay all 1391 cases in the event log.

Figure 1 shows three additional models. These models also aim to describe the process that produced the event log. Model M_2 is only able to replay the 455 cases that followed $acdeh$ and none of the 936 other cases fits completely. Model M_3 is able to replay the whole log (i.e., all 1391 cases), but also allows for traces dissimilar to the observed behavior, e.g., traces such as $aeeeeeh$ are possible according to M_3. Model M_4 simply enumerates all 21 traces observed. This model is too complex and is also "overfitting" the observed behavior, i.e., unseen but likely behavior is excluded.

The four process models may have been discovered using a process discovery algorithm or may also have been made by hand. In either case, it is interesting to check conformance of model and log. Assuming that M_1 is the model that best fits reality as observed in the event log, we can use the relation between M_1 and the 7539 events in the log as the starting point for all kinds of analysis. For example, when events have timestamps, we can replay the log and add performance related information to M_1. For example, all places (circles) in M_1 can be annotated with the observed average waiting time extracted from the event log.


start
a register request
b examine thoroughly
c examine casually
d check ticket
e decide
f reinitiate request
g pay compensation
h reject request
end
M₁ : fitness = +, precision = +, generalization = +, simplicity = +
M₂ : fitness = -, precision = +, generalization = -, simplicity = +
M₃ : fitness = +, precision = -, generalization = +, simplicity = +
(all 21 variants seen in the log)
M₄ : fitness = +, precision = +, generalization = -, simplicity = -

event log

#	trace
455	acdeh
191	abdeg
177	adceh
144	abdeh
111	acdeg
82	adceg
56	adbeh
47	acdefdbeh
38	adbeg
33	acdefbdeh
14	acdefbdeg
11	acdefdbeg
9	adcefcdeh
8	adcefdbeh
5	adcefbdeg
3	acdefbdefdbeg
2	adcefdbeg
2	adcefbdefbdeg
1	adcefdbefbdeh
1	adbefbdefdbeg
1	adcefdbefcdefdbeg
1391	

Figure 1: One event log L (right) and four process models M_1, M_2, M_3, and M_4 (left).

Aligning Event Log and Process Model

Conformance checking and performance analysis require an *alignment* of event log and process model, i.e., events in the event log need to be related to model elements and vice versa. Such an alignment shows how the event log can be replayed on the process model. This is far from trivial as the log may deviate from the model and not all activities may have been modeled or recorded. To discuss these complications we need to introduce some notations.

The event log in Fig. 1 contains 1391 cases. Each case is described by a trace, i.e., a sequence of activity names. Different cases may have the same trace. Therefore, an event log is a *multiset* of traces (rather than a set). A_L denotes the set of *activities* that may be recorded in the log. $\sigma \in A_L^*$ is a *trace*, i.e., a sequence of events. $L \in \mathbb{B}(A_L^*)$ is an *event log*, i.e., a multiset of traces. For example, $L = [acdeh, acdeh, abdeg]$ is an event log with three cases, two of which follow the same trace.

Each of the four process models in Fig. 1 is represented as a Petri net. Other representations frequently used are UML activity diagrams, BPMN (Business Process Modeling Notation), EPCs (Event-driven Process Chains) models, YAWL models, etc. (17; 31). It is essential that such models are able to represent common process patterns in a compact and intuitive manner. For example, process models should be able to represent concurrency (e.g., in M_1 checking and examination activities can be done in parallel). However, in this article we abstract from the actual process notation and focus on the behavior described by the model. Therefore, we can use basic *transition systems* as a placeholder for more advanced modeling languages (Petri nets, UML, BPMN, EPCs, etc.).

A *process model* $M = (S, S_I, S_F, A_M, T)$ is a transition system over a set of activities A_M with states S, initial states $S_I \subseteq S$, final states $S_F \subseteq S$, and transitions $T \subseteq S \times A_M \times S$. The set of activities in the model A_M may differ from the set of activities in the event log A_L. The transition system starts in a state in S_I and moves from one state to another according to the transition relation T, e.g., $(s_1, a, s_2) \in T$ means that in state s_1 the transition system can move to state s_2 while producing an event labeled a. Eventually, a final state in S_F should be reached in order to complete.

$\beta(M) \subseteq A_M^*$ is the set of all *full execution sequences*, i.e., possible traces starting in a state in S_I and ending in a state in S_F. Consider for example $M = (\{s_1, s_2, s_3, s_4\}, \{s_1\}, \{s_4\}, \{a, b, c, d\}, \{(s_1, a, s_2), (s_2, b, s_3), (s_3, c, s_2), (s_2, d, s_4)\})$. $\beta(M) = \{ad, abcd, abcbcd, abcbcbcd, \ldots\}$. All modeling languages with executable semantics can be described in this way.

An important assumption of the approach presented in this article is that the relationship between activities in the model and events in the event log is known, i.e., both events and activities carry labels. This way it is possible to relate events to activities. However, as we will show later, there may be multiple activities carrying the same label and there may be a partial match between an event and activity (defined by a so-called distance function δ).

To establish an alignment between process model and event log we need to relate "moves" in the log to "moves" in the model. However, it may be the case that some of the moves in the log cannot be mimicked by the model and vice versa. We explicitly denote "no move" by $\perp$. For convenience, we introduce the set $A_L^{\perp} = A_L \cup \{\perp\}$ where $x \in A_L$ refers to "move x in log" and $\perp$ refers to "no move in log". Similarly, we use $A_M^{\perp} = A_M \cup \{\perp\}$ where $y \in A_M$ refers to "move y in model" and $\perp$ refers to "no move in model".

One step in an alignment is represented by a pair $(x, y) \in A_L^{\perp} \times A_M^{\perp}$ such that

- (x, y) is a *move in log* if $x \in A_L$ and $y = \perp$,
- (x, y) is a *move in model* if $x = \perp$ and $y \in A_M$,
- (x, y) is a *move in both* if $x \in A_L$ and $y \in A_M$,
- (x, y) is an *illegal move* $x = \perp$ and $y = \perp$.

$A_{LM} = \{(x, y) \in A_L^{\perp} \times A_M^{\perp} \mid x \in A_L \ \vee \ y \in A_M\}$ is the set of all *legal moves*.

Let $\sigma_L \in L$ be a trace in event log L and let $\sigma_M \in \beta(M)$ be a full execution sequence of model M. An *alignment* of σ_L and σ_M is a sequence $\gamma \in {A_{LM}}^*$ such that the projection on the first element (ignoring $\perp$) yields σ_L and the projection on the second element (again ignoring $\perp$) yields σ_M. Two examples of alignments:

$$\gamma_1 = \begin{array}{|c|c|c|c|c|} a & c & d & e & h \\ \hline a & c & d & e & h \end{array} \quad \text{and} \quad \gamma_2 = \begin{array}{|c|c|c|c|c|c|c|} a & b & \perp & d & e & g & \perp \\ \hline a & \perp & c & d & e & \perp & h \end{array}$$

Note that we represent the moves vertically, e.g., the first move of γ_1 is (a, a) indicating that both the log and the model make an a move. γ_1 is an alignment of $\sigma_L = acdeh$ and $\sigma_M = acdeh$. γ_2 is an alignment of $\sigma_L = abdeg$ and $\sigma_M = acdeh$. Two additional alignments for $\sigma_L = abdeg$ and $\sigma_M = acdeh$:

$$\gamma_3 = \begin{array}{|c|c|c|c|c|c|c|c|} a & b & \perp & d & e & \perp & \perp & g \\ \hline \perp & a & c & d & \perp & e & h & \perp \end{array} \quad \text{and} \quad \gamma_4 = \begin{array}{|c|c|c|c|c|c|c|} a & \perp & b & d & e & g & \perp \\ \hline a & \perp & c & d & e & \perp & h \end{array}$$

Note that in γ_3 there is first a "move in log" $(a, \perp)$ followed by a "move in both" (b, a). Clearly, this is not an optimal alignment because there is no need to just move in the log and then to move both without agreement on the step to be made: the log makes a b move whereas the model makes an a move. γ_4 is not a correct alignment because it contains an illegal move $(\perp, \perp)$ (see second position). After removing the illegal move, we obtain a proper alignment.

γ_1 shows the ideal situation where model and log can mimic each other perfectly. Such an alignment is not possible for $\sigma_L = abdeg$ and $\sigma_M = acdeh$. Instead, one needs to resort to alignments such as γ_2 and γ_3. Of these two alignments γ_2 is preferable because γ_3 contains more moves in only the log or model, and γ_3 has a "move in both" (x, y) with $x \neq y$ (second position).

To qualify the quality of an alignment we introduce a *distance function* on legal moves: $\delta \in A_{LM} \to \mathbb{N}$. The distance function associates costs to moves in an alignment:

- if $x \in A_L$ and $y = \perp$, then $\delta(x, y)$ is the cost of "move x in log",
- if $x = \perp$ and $y \in A_M$, then $\delta(x, y)$ is the cost of "move y in model", and
- if $x \in A_L$ and $y \in A_M$, then $\delta(x, y)$ is the cost of "move x in log and move y in model" (typically $\delta(x, y) = 0$ if $x = y$).

Distance function δ can be generalized to alignments by taking the sum of the costs of all individual moves: $\delta(\gamma) = \sum_{(x,y)\in\gamma} \delta(x, y)$.[1]

We define a *standard distance function* δ_S. For $x \in A_L$ and $y \in A_M$: $\delta_S(x, \perp) = 1$, $\delta_S(\perp, y) = 1$, $\delta_S(x, y) = 0$ if $x = y$, and $\delta_S(x, y) = \infty$ if $x \neq y$. Only moves where log and model agree on the activity have no associated costs. Moves in just the log or model have cost 1. δ_S associates high costs to moves where both log and model make a move but disagree on the activity. In "$\delta_S(x, y) = \infty$", ∞ should be read as a number large enough to discard the alignment (see below). Using the standard distance function δ_S: $\delta_S(\gamma_1) = \delta_S(a, a) + \delta_S(c, c) + \delta_S(d, d) + \delta_S(e, e) + \delta_S(h, h) = 0$, $\delta_S(\gamma_2) = 4$ (four moves in just the log or model), and $\delta_S(\gamma_3) = \infty$ (because of move (b, a)).

Various cost functions can be defined. For example, the weights of activities may vary. Skipping a payment may have a much higher cost than skipping some minor check. The cost function can also be used to compare activities which are not identical but similar, e.g., in the example we may define $\delta(b, c) = \delta(c, b) = 0.8$ because both b and c examine the request and are exchangeable to some degree. It is also possible that the model has multiple activities which have the same observable effect. For example, in a Petri net there may be multiple transitions (e.g., t_1 and t_2) corresponding to some activity x: $\delta(x, t_1) = \delta(x, t_2) = 0$. In fact, it is often desirable to give all transitions a unique name. This way any $\sigma_M \in \beta(M)$ corresponds to a unique path in the model. In the remainder, we assume the standard distance function δ_S unless mentioned otherwise.

Thus far we considered a *specific* full execution sequence in the model. However, our goal is to relate traces in the model to the *best matching* full execution sequence in the model. Therefore, we define the notion of an *optimal alignment*. Let $\sigma_L \in L$ be a trace in event log L and let M be a model. $\Gamma_{\sigma_L,M} = \{\gamma \in {A_{LM}}^* \mid \exists_{\sigma_M\in\beta(M)}\ \gamma\ \mathit{is\ an\ aligment\ of}\ \sigma_L\ \mathit{and}\ \sigma_M\}$. An alignment $\gamma \in \Gamma_{\sigma_L,M}$ is *optimal* for log trace $\sigma_L \in L$ and model M if for any $\gamma' \in \Gamma_{\sigma_L,M}$: $\delta(\gamma') \geq \delta(\gamma)$.

For an event log L and process model M, we define $\lambda_M \in {A_L}^* \to {A_{LM}}^*$: a mapping that assigns any $\sigma_L \in L$ to an optimal alignment, i.e., $\lambda_M(\sigma_L) \in \Gamma_{\sigma_L,M}$ and $\lambda_M(\sigma_L)$ is optimal. It is easy to see that such a mapping exists (but there may be multiple). Consider for example M_2 and $\sigma_L = acdefdbeh$, i.e., the trace that occurred 47 times in the log of Fig. 1. Mapping λ_{M_2} may produce the following optimal alignment:

$$\lambda_{M_2}(\sigma_L) = \begin{array}{|c|c|c|c|c|c|c|c|c|} a & c & d & e & f & d & b & e & h \\ \hline a & c & d & \perp & \perp & \perp & \perp & e & h \end{array}$$

[1]Note that summation is generalized to sequences and multisets, i.e., $\sum_{x\in aab} f(x) = \sum_{x\in[a,a,b]} f(x) = 2f(a) + f(b)$.

Note that the alignment defines a path in the model from an initial state to a final state (in this case $acdeh$). $\overline{\lambda}_M(\sigma_L) \in \beta(M)$ is the projection on the second element of $\lambda_M(\sigma_L)$ (ignoring $\perp$). Hence, $\overline{\lambda}_M \in {A_L}^* \rightarrow {A_M}^*$ such that $\overline{\lambda}_M(\sigma_L)$ gives the *best matching full execution sequence* for trace σ_L in model M. Consider for example M_1 and $\sigma_L = abefbh$, i.e. a trace that cannot be generated by the Petri net in Fig. 1. Given the standard distance function δ_S there is just one choice:

$$\lambda_{M_1}(\sigma_L) = \begin{array}{|c|c|c|c|c|c|c|c|c|} a & b & \perp & e & f & b & \perp & \perp & h \\ \hline a & b & d & e & f & b & d & e & h \end{array}$$

Hence, $\overline{\lambda}_{M_1}(\sigma_L) = abdefbdeh$ is the best matching sequence with cost $\delta_S(\lambda_{M_1}(\sigma_L)) = 3$.

In general, there may be many other alignments having the same cost. However, function λ_M provides an "oracle" that, given an log trace σ_L, produces *one* best matching full execution sequence $\overline{\lambda}_M(\sigma_L)$. It is always possible to create such a function based on some predefined distance function. In (6; 5) various approaches are given to create optimal alignment with respect to some predefined distance function. These approaches are based on the A^* algorithm, i.e., an algorithm originally invented to find the shortest path between two nodes in a directed graph. The A^* algorithm can be adapted to find an optimal alignment between model and log. In this article we do not elaborate on the A^* algorithm. In principle, any optimization approach can be used to compute $\lambda_M(\sigma_L)$. Here we focus on the result that after alignment *any trace in the event log can be replayed on model* (even if the trace does not fit perfectly).

Conformance Checking

For conformance checking we use the "oracle" λ_M that relates traces in the event log to paths in the model. In this section, we consider the standard four quality dimensions for comparing model and log: (a) *fitness*, (b) *simplicity*, (c) *precision*, and (d) *generalization* (1).

Fitness: Can the Model Generate The Observed Behavior?

A model with good *fitness* allows for most of the behavior seen in the event log. A model has a perfect fitness if all traces in the log can be replayed by the model from beginning to end. To quantify fitness, we can directly apply the distance function δ on the optimal alignments provided by the "oracle" λ_M. The total alignment cost for event log L and model M is

$$\mathit{fcost}(L, M) = \sum_{\sigma_L \in L} \delta(\lambda_M(\sigma_L))$$

Consider for example event log L and the four models shown in Fig. 1. $\mathit{fcost}(L, M_1) = 0$, $\mathit{fcost}(L, M_2) = 2884$, $\mathit{fcost}(L, M_3) = 0$, and $\mathit{fcost}(L, M_4) = 0$. This shows that

M_1, M_3, and M_4 are able to replay the log without any problems. The total alignment cost for model M_2 is 2884. This sum is caused by the moves in just the model or just the log, for all cases different from $acdeh$.

Often, we would like to express fitness as a number between 0 (very poor fitness) and 1 (perfect fitness). There are several ways to normalize the alignment costs. One approach is to divide $\mathit{fcost}(L, M)$ by the maximal possible cost. Let us assume that for $x \in A_L$ and $y \in A_M$: $\delta(x, y) = \infty$ if $x \neq y$, i.e., we only allow a move on both model and log (x, y) if $x = y$. In this case, the worst-case scenario is that there are just moves in the log and moves in the model (and never both) in the optimal alignment. $\mathit{move}_L(L) = \sum_{\sigma_L \in L} \sum_{x \in \sigma_L} \delta(x, \perp)$ is the total cost of moving through the whole log without ever moving together with the model. $\mathit{move}_M(M) = \min_{\sigma_M \in \beta(M)} \sum_{y \in \sigma_M} \delta(\perp, y)$ is the total cost of making moves on model only. Note that we consider the "least expensive path" because an optimal alignment will try to minimize costs. This results in the following definition of fitness:

$$\mathit{fitness}(L, M) = 1 - \frac{\mathit{fcost}(L, M)}{\mathit{move}_L(L) + |L| \times \mathit{move}_M(M)}$$

$\mathit{fitness}(L, M_1) = 1$, $\mathit{fitness}(L, M_2) = 0.8$, $\mathit{fitness}(L, M_3) = 1$, and $\mathit{fitness}(L, M_4) = 1$. The fitness value for model M_2 is 0.8. This may seem high as just 455 of 1391 cases fit completely (i.e., less than 33% of the process instances can be replayed from beginning to end). However, note that all cases start with a and execute at least one d and e. Moreover, the majority of cases ends with h. This explains the high fitness value.

Note that there are alternative ways to normalize the alignment cost to produce a fitness value, e.g., a fitness value only considering moves on log.

Precision: Avoid Underfitting

Model M_3 in Fig. 1 has a fitness of 1 although the model clearly allows for behavior very different from the behavior seen in the event log. A model is *precise* if it does not allow for "too much" behavior. A model that is not precise is "underfitting". Underfitting is the problem that the models such as M_3 over-generalize the example behavior seen in the log.

In (25) two so-called "appropriateness notions" are defined (behavioral appropriateness and structural appropriateness). The goal is to discover situations where "too" many transitions are enabled. For example, after the execution of a in M_3, seven activities (b, c, d, e, f, g and h) are continuously enabled until completion. However, based on the event log it appears that in most situations just one to three activities are enabled rather than seven.

In (23) a so-called prefix automaton is constructed to see where the process model allows for more behavior than actually observed in the event log. This approach is further improved in (22; 24).

The approach proposed in this paper is very different from (25). Instead, we propose an approach similar to (22), i.e., using a prefix automaton combined with an alignment between model and log. Before doing so, we revisit our notion of event logs.

Event log L does not need to fit model M. However, we can construct a modified event log $L' = [\overline{\lambda}_M(\sigma) \mid \sigma \in L]$. In the remainder, we only use such preprocessed logs, i.e., *from now on we only consider event logs after alignment*. Note that this implies that given a log L and model M: (a) $A_L = A_M$ and (b) $\sigma \in L$ implies $\sigma \in \beta(M)$.

Moreover, we assume any model M to be *deterministic*, i.e., given some execution sequence $\sigma \in \beta(M)$ there is only one corresponding path in the model. This also holds for prefixes of σ. Model M_4 in Fig. 1 does not meet this requirement unless we relabel the transitions. For any state in a deterministic model, there cannot be two enabled transitions with the same label. Note that any model can be made deterministic: simply rename competing transitions with the same label and adapt the distance function accordingly.

To simplify our definition of precision, we view an event log as a *collection of unique events* $\mathcal{E}$. Consider the first activity of trace $acdeh$ in the event log of Fig. 1. There are 455 distinct events corresponding to initial activity a in $acdeh$. For the example log, $\mathcal{E}$ contains 7539 events. The activity associated to $e \in \mathcal{E}$ is $act(e)$ and the timestamp of e is $time(e)$. In addition, we define the following functions given $e \in \mathcal{E}$ and some model $M = (S, S_I, S_F, A_M, T)$:

- $state_M(e) \in S$ is the *state* in M *just before* the occurrence of e. Note that it is possible to derive this state because we consider a preprocessed log with fitness 1 and a deterministic model.
- $context_L(e) \in {A_M}^*$ is the *activity prefix* of the process instance *just before* the occurrence of e, i.e., the sequence of all activities that happened before event e. We call this prefix the *context* of e.
- $en_M(e) \subseteq A_M$ is the set of activities enabled in $state_M(e)$, i.e., $en_M(e) = \{a \in A_M \mid \exists_{s \in S}\ (state_M(e), a, s) \in T\}$. Note that $act(e) \in en_M(e)$.
- $en_L(e) \subseteq A_M$ is the set of activities that were executed in the same context, i.e., $en_L(e) = \{act(e') \mid e' \in \mathcal{E} \wedge context_L(e') = context_L(e)\}$. $en_L(e) \subseteq en_M(e)$ because we consider a preprocessed log with fitness 1 and a deterministic model (the same prefix always leads to the same state).
- $sim(e) \subseteq \mathcal{E}$ is the set of all events that happened in the same state as the state just before executing e, i.e., $sim(e) = \{e' \in \mathcal{E} \mid state_M(e') = state_M(e)\}$. Note that $e \in sim(e)$.
- $diff(e) = \{act(e') \mid e' \in sim(e)\} \subseteq A_M$ are the different activities that were executed in the same state as the state just before executing e.

Each event $e \in \mathcal{E}$ implicitly refers to a point in the event log just before executing e. For such a point we can count the number of enabled activities in the model $en_M(e)$

and the number of observed activities actually executed in a similar context $en_L(e)$. This can be used to define the following precision notion:

$$precision(L, M) = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \frac{|en_L(e)|}{|en_M(e)|}$$

If all behavior allowed by the model is actually observed, then $precision(L, M) = 1$. By taking the average over all events, we automatically take frequencies into account. If the model has an activity that is enabled on a frequent path but the activity is never executed, then this is more severe than an unused activity enabled along an infrequent path.

For our running example we compute: $precision(L, M_1) = 0.97$, $precision(L, M_2) = 1$, $precision(L, M_3) = 0.41$, and $precision(L, M_4) = 1$. This clearly shows that model M_3 is underfitting.

Generalization: Avoid Overfitting

Model M_4 simply encodes the event log and does not generalize. In general, a process model should not restrict behavior to just the examples seen in the log. A model that does not *generalize* is "overfitting". Overfitting is the problem that a very specific model is generated whereas it is obvious that the log only holds example behavior (i.e., the model explains the particular sample log, but it is unlikely that another sample log of the same process can be explained well by the current model).

It is difficult to reason about generalization because this refers to unseen examples. However, we can use an approach similar to the one used for the quantification of precision. Again we consider all points in the event log where event $e \in \mathcal{E}$ is about to happen. Given an event e we can find all events e' that occurred in the same state in model M. Every event can be seen as an observation of an activity in some state s. Suppose that state s is visited n times and that w is the number of *different* activities observed in this state. Suppose that n is very large and w is very small, then it is unlikely that a new event visiting this state will correspond to an activity not seen before in this state. However, if n and w are of the same order of magnitude, then it is more likely that a new event visiting state s will correspond to an activity not seen before in this state. Using this idea we define generalization as follows:

$$generalization(L, M) = 1 - \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} pnew(|diff(e)|, |sim(e)|)$$

$pnew(w, n)$ is the estimated probability that a next visit to state $s = state_M(e)$ will reveal a new path not seen before: $w = |diff(e)|$ is the number of unique activities observed leaving state s and $n = |sim(e)|$ is the number of times s was visited by the event log. We use an estimator inspired by (7): $pnew(w, n) = \frac{w(w+1)}{n(n-1)}$ if $n \geq w+2$ and $pnew(w, n) = 1$ otherwise. This estimate can be derived under the Bayesian assumption that there is an unknown number of possible activities in state s and that probability distribution over these activities follows a multinomial distribution. This results in the posterior transition probability $pnew(w, n)$ if $n \geq w + 2$ (7). For $n \leq w + 1$, the Bayesian analysis proposed in (7) does not provide a transition probability. In this case there are too few repetitions to properly estimate the posterior transition probability. For $n = w + 1$, $\frac{w(w+1)}{n(n-1)} = 1$. For $n = w$, $\frac{w(w+1)}{n(n-1)} > 1$ (or undefined). Therefore, we assume the transition probability to be 1 if $n \leq w + 1$. This is a reasonable assumption (especially for larger n): if most observations are unique, it is likely that the next observation will also be unique.

$generalization(L, M)$ is close to 0 if it is very likely that new events will exhibit behavior not seen before. $generalization(L, M)$ is close to 1 if it is very unlikely that the next event will reveal new behavior.

For our running example, $generalization(L, M_1) = 1.0$, $generalization(L, M_2) = 1.0$, $generalization(L, M_3) = 1.0$, and $generalization(L, M_4) = 0.99$. The high values for M_1, M_2, and M_3 correspond to the intuitive notion of generalization. The next trace to be observed is likely to fit into these models. However, the value for M_4 may be higher than expected. The reason is that *generalization* takes the average over all events, and most events occur (by definition) in the highly frequent traces. For example, there are 455 process instances having a trace $acdeh$. For all intermediate states there is only one possible next activity ($w = 1$) whereas these states are visited 455 times ($n = 455$). Hence $pnew(w, n) = \frac{1(1+1)}{455(455-1)} \approx 0$ for all events having such a trace resulting in a very high precision value. The effect becomes clear if we assume that each of the 21 possible traces occurs only once. Let us call this log L_s. Then $generalization(L_s, M_1) = 0.99349$, $generalization(L_s, M_2) = 0.99524$, $generalization(L_s, M_3) = 0.99750$, and $generalization(L_s, M_4) = 0.11547$. For this smaller log, it becomes clear that M_4 is indeed less general (i.e., more overfitting).

Alternatively, we can also define the notion of generalization at the *state level* rather than the event level:

$$generalization_S(L, M) = 1 - \frac{1}{|S|} \sum_{s \in S} pnew(|diff(s)|, |sim(s)|)$$

Now functions $diff$ and sim are defined for states rather than events. For $s \in S$: $sim(s) = \{e \in \mathcal{E} \mid state_M(e) = s\} \subseteq \mathcal{E}$ is the set of all events that happened in state s and $diff(s) = \{act(e) \mid e \in sim(s)\} \subseteq A_M$ are the different activities that were executed in state s. This metric will punish the parts of the model that are low-frequent but overfitting much harder. Hence, this metric results in an even lower generalization value for M_4.

When evaluating a process discovery *algorithm*, one can also use an alternative approach. First, construct the model M for the whole event log L. Suppose that $|L| = k$ and that the cases are numbered: $1, 2, \ldots, k$. Create k events logs by removing precisely one process instance: $L_1, L_2, \ldots, L_k$. Event log L_i is L without the events corresponding to case i. Next, construct a model M_i for each of the event logs L_i. Let l be the number of models M_i that can replay case i successfully. The fraction l/k is an alternative metric for generalization. If l/k is close to 0, then the model is overfitting. If l/k is close to 1, the model is able to generalize well. Also other forms of cross validation (e.g., k-fold cross-validation) can be used to quantify the degree of generalization.

Simplicity: Occam's Razor

The fourth quality dimension is related to Occam's Razor which states that "one should not increase, beyond what is necessary, the number of entities required to explain anything". Following this principle, we look for the "simplest process model" that can explain what is observed in the event log. Model M_1 in Fig. 1 is simple compared to M_4. This makes M_4 a less suitable model.

There are various techniques to quantify model complexity. The complexity of the model could be defined by the number of nodes and arcs in the underlying graph. Also more sophisticated metrics can be used, e.g., metrics that take the "structuredness" or "entropy" of the model into account. A detailed discussion of these complexity metrics is outside the scope of this article. We refer to (21) for pointers to over twenty metrics described in literature.

Performance Analysis

After aligning event log and model, all kinds of analysis techniques based on "replay" come into reach. These replay techniques may use additional attributes and are not restricted to activity names. As mentioned earlier, events may have timestamps and refer to resources, customers, organizational units, cost rates, etc. Timestamps can be used to compute flow times, waiting times, service times, synchronization times, etc. Consider two subsequent events e_1 and e_2 with $act(e_1) = a$, $act(e_2) = b$, $time(e_1) =$ `23-11-2011:15.56`, and $time(e_2) =$ `23-11-2011:16.20`. If b is causally dependent on a, we record a time of 24 minutes in-between a and b. By repeatedly measuring such time differences during replay, we can compute the average time that elapses in-between a and b.

Such detailed analysis is only possible after successfully aligning model and log. A process model annotated with time information can be used to diagnose performance problems. Moreover, the learned temporal behavior can be used for predictions ("What is the remaining flow time for this running case?") and recommendations ("Which action minimizes the expected overall costs?"). See (1) for more details and concrete examples.

Related Work

This article describes a collection of process mining algorithms aiming at conformance checking and performance analysis. See (1) for an introduction to process mining. We refer to the recently released Process Mining Manifesto (18) for the main challenges in process mining.

Cook et al. (9; 8) were among the first to quantify the relationship between event logs and process model. They compare event streams of the model with event steams generated from the event log.

Several authors proposing process discovery algorithms also provide a quality metric (often related to fitness). For example, in (20) the authors define a fitness function for searching for the optimal model using a genetic approach. In (26) a "process mining evaluation framework" for benchmarking process discovery algorithms is proposed.

The first comprehensive approach to conformance analysis was proposed in (25) by Rozinat and Van der Aalst. Two different types of metrics are proposed: (a) *fitness metrics*, i.e., the extent to which the log traces can be associated with valid execution paths specified by the process model, and (b) *appropriateness metrics*, i.e., the degree of accuracy in which the process model describes the observed behavior, combined with the degree of clarity in which it is represented. Fitness in (25) is measured by "replaying the event log" and counting the number of missing and remaining tokens. This typically results in rather high fitness values as also pointed out in (6; 28). In (25) four appropriateness metrics are defined. *Simple behavioral appropriateness* looks at the average number of enabled transitions. If most transitions are continuously enabled, the model is likely to lack precision (i.e., underfitting). *Advanced behavioral appropriateness* compares the "footprint" of the log (follows and precedes relationships) to the "footprint" of the model. *Simple structural appropriateness* and *advanced structural appropriateness* quantify the complexity of the model. Note that the appropriateness metrics in (25) cover the simplicity, precision, and generalization dimensions discussed in this article.

One of the drawbacks of the approach in (25) and most other approaches that "play the token game", is that fitness is typically overestimated. When a model and log do not fit well together, replay will overload the process model with superfluous tokens. As a result, the model will allow for too much behavior. Approaches such as the one in (25) also have problems when the model has "invisible activities" (silent steps that are not recorded in the event log) or "duplicate activities" (multiple transitions bearing the same label). To deal with such phenomena state-space exploration and heuristics are needed to replay the event log. In fact, most conformance techniques give up after the first non-fitting event or simply "guess" the corresponding path in the model. Therefore, Adriansyah et al. formulated conformance checking problems as an optimization problem (6; 5). This is the approach we also use in this article.

The lion's share of attention in conformance checking has been devoted to checking fitness. Besides the four appropriateness notions in (25), Munoz-Gama et al. quantified additional precision notions (22; 23; 24). In this paper, we use an approach similar to the one reported in (22). Unlike (23; 24), this approach is robust in case of non-fitting traces.

It is difficult to use classical quality notions such as *precision* and *recall* for process mining. The main reason is that event logs only contain positive examples, i.e., one can see what "did happen" but not what "could not happen". Therefore, some authors suggest inserting artificially generated "negative events" (12; 29). Goedertier et al. proposed such events for both process discovery and conformance checking (12). De Weerdt et al. defined a so-called F-measure based on artificially generated negative events (29). The authors of the latter paper also conducted a comparative analysis of several conformance metrics (28). However, their study did not consider the more advanced alignment-based approaches discussed in this article.

In (15) a so-called completeness metric and soundness metric are defined. These metrics compare the traces of the model with the traces in the log. This approach suffers from several drawbacks. First of all, only complete traces are compared. Second, it is assumed that the model's behavior can be enumerated. Finally, it is assumed that the log contains all possible traces.

In (11), the techniques presented in (6; 5) are generalized to artifact-centric processes (the so-called Proclets). The conformance notions in (11) also take interactions between process instances into account.

Several approaches create so-called behavioral footprints to compare event log and model (1). The key idea is that a footprint can be based on observed behavior and modeled behavior as described in (1). Another example of such a footprint is the so-called "behavioral profile" (30). The problem of this approach is that it cannot handle loops properly (unlike (1; 25)).

All techniques discussed thus far, compare model and log. There are also many compliance approaches that compare a model and another model or a model and a set of rules (10; 13; 14; 19). These approaches are very different from the techniques discussed in this paper as they do not take the actual observed behavior into account.

As indicated in this article, alignments can also be used for performance analysis as most event logs contain timestamps (1). Replaying event logs with timestamps allows for bottleneck analysis and prediction as demonstrated in (2; 3).

Conclusion

Process mining can be seen as the "missing link" between data mining and traditional model-driven BPM. Although process discovery received the lion's share of attention, the alignment between event log and model is at least as important. An alignment makes it possible to replay the event log on the model. This can be used for performance analysis and conformance checking. As demonstrated, it is possible to decouple analysis and first establish an optimal alignment before measuring conformance or identifying bottlenecks.

Many of the ideas presented in this article have been implemented in *ProM*, an open-source process mining platform offering a wide variety of techniques for process discovery, conformance checking, and model enhancement (1; 27). From a computational point of view it is challenging to compute an optimal alignment (6; 5). Hence, future work should aim at more efficient algorithms. Moreover, it will be fruitful to explore the full application potential of replay (beyond conformance checking and performance analysis).

References

[1] W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer-Verlag, Berlin, 2011.

[2] W.M.P. van der Aalst, M. Pesic, and M. Song. Beyond Process Mining: From the Past to Present and Future. In B. Pernici, editor, *Advanced Information Systems Engineering, Proceedings of the 22nd International Conference on Advanced Information Systems Engineering (CAiSE'10)*, volume 6051 of *Lecture Notes in Computer Science*, pages 38–52. Springer-Verlag, Berlin, 2010.

[3] W.M.P. van der Aalst, M.H. Schonenberg, and M. Song. Time Prediction Based on Process Mining. *Information Systems*, 36(2):450–475, 2011.

[4] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[5] A. Adriansyah, B. van Dongen, and W.M.P. van der Aalst. Conformance Checking using Cost-Based Fitness Analysis. In *IEEE International Enterprise Computing Conference (EDOC 2011)*. IEEE Computer Society, 2011.

[6] A. Adriansyah, B.F. van Dongen, and W.M.P. van der Aalst. Towards Robust Conformance Checking. In M. zur Muehlen and J. Su, editors, *BPM 2010 Workshops, Proceedings of the Sixth Workshop on Business Process Intelligence (BPI2010)*, volume 66 of *Lecture Notes in Business Information Processing*, pages 122–133. Springer-Verlag, Berlin, 2011.

[7] C. Boender and A. Rinnooy Kan. A Bayesian Analysis of the Number of Cells of a Multinomial Distribution. *The Statistician*, 32(1-2):240–248, 1983.

[8] J.E. Cook, C. He, and C. Ma. Measuring Behavioral Correspondence to a Timed Concurrent Model. In *Proceedings of the 2001 International Conference on Software Mainenance*, pages 332–341, 2001.

[9] J.E. Cook and A.L. Wolf. Software Process Validation: Quantitatively Measuring the Correspondence of a Process to a Model. *ACM Transactions on Software Engineering and Methodology*, 8(2):147–176, 1999.

[10] A. Elgammal, O. Turetken, W. van den Heuvel, and M. Papazoglou. Root-Cause Analysis of Design-time Compliance Violations on the basis of Property Patterns. In P. Maglio, M. Weske, J. Yang, and M. Fantinato, editors, *Proceedings of Service-Oriented Computing (ICSOC 2010)*, volume 6470 of *Lecture Notes in Computer Science*, pages 17–31. Springer-Verlag, Berlin, 2010.

[11] D. Fahland, M. de Leoni, B.F. van Dongen, and W.M.P. van der Aalst. Conformance Checking of Interacting Processes with Overlapping Instances. In S. Rinderle, F. Toumani, and K. Wolf, editors, *Business Process Management (BPM 2011)*, volume 6896 of *Lecture Notes in Computer Science*, pages 345–361. Springer-Verlag, Berlin, 2011.

[12] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens. Robust Process Discovery with Artificial Negative Events. *Journal of Machine Learning Research*, 10:1305–1340, 2009.

[13] G. Governatori, J. Hoffmann, S. Sadiq, and I. Weber. Detecting Regulatory Compliance for Business Process Models through Semantic Annotations. In *4th International Workshop on Business Process Design*, 2008.

[14] G. Governatori, Z. Milosevic, and S. Sadiq. Compliance Checking Between Business Processes and Business Contracts. In *10th International Enterprise Distributed Object Computing Conference (EDOC 2006)*, pages 221–232. IEEE Computing Society, 2006.

[15] G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Discovering Expressive Process Models by Clustering Log Traces. *IEEE Transaction on Knowledge and Data Engineering*, 18(8):1010–1027, 2006.

[16] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT press, Cambridge, MA, 2001.

[17] A.H.M. ter Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation: YAWL and its Support Environment*. Springer-Verlag, Berlin, 2010.

[18] IEEE Task Force on Process Mining. Process Mining Manifesto. In *BPM Workshops*, volume 99 of *Lecture Notes in Business Information Processing*. Springer-Verlag, Berlin, 2011.

[19] N. Lohmann. Compliance by Design for Artifact-Centric Business Processes. In S. Rinderle, F. Toumani, and K. Wolf, editors, *Business Process Management (BPM 2011)*, volume 6896 of *Lecture Notes in Computer Science*, pages 99–115. Springer-Verlag, Berlin, 2011.

[20] A.K. Alves de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic Process Mining: An Experimental Evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.

[21] J. Mendling, G. Neumann, and W.M.P. van der Aalst. Understanding the Occurrence of Errors in Process Models Based on Metrics. In F. Curbera, F. Leymann, and M. Weske, editors, *Proceedings of the OTM Conference on Cooperative information Systems (CoopIS 2007)*, volume 4803 of *Lecture Notes in Computer Science*, pages 113–130. Springer-Verlag, Berlin, 2007.

[22] J. Munoz-Gama, A. Adriansyah, J. Carmona, B.F. van Dongen, and W.M.P. van der Aalst. Alignment Based Precision Checking in Process Mining. Eindhoven University of Technology, Eindhoven, 2011.

[23] J. Munoz-Gama and J. Carmona. A Fresh Look at Precision in Process Conformance. In R. Hull, J. Mendling, and S. Tai, editors, *Business Process Management (BPM 2010)*, volume 6336 of *Lecture Notes in Computer Science*, pages 211–226. Springer-Verlag, Berlin, 2010.

[24] J. Munoz-Gama and J. Carmona. Enhancing Precision in Process Conformance: Stability, Confidence and Severity. In N. Chawla, I. King, and A. Sperduti, editors, *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, Paris, France, April 2011. IEEE.

[25] A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008.

[26] A. Rozinat, A.K. Alves de Medeiros, C.W. Günther, A.J.M.M. Weijters, and W.M.P. van der Aalst. The Need for a Process Mining Evaluation Framework in Research and Practice. In A. ter Hofstede, B. Benatallah, and H.Y. Paik, editors, *BPM 2007 International Workshops (BPI, BPD, CBP, ProHealth, RefMod, Semantics4ws)*, volume 4928 of *Lecture Notes in Computer Science*, pages 84–89. Springer-Verlag, Berlin, 2008.

[27] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. ProM 6: The Process Mining Toolkit. In M. La Rosa, editor, *Proc. of BPM Demonstration Track 2010*, volume 615 of *CEUR Workshop Proceedings*, pages 34–39, 2010.

[28] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens. A Critical Evaluation of Model-Log Metrics in Process Discovery. In M. zur Muehlen and J. Su, editors, *BPM 2010 Workshops, Proceedings of the Sixth Workshop on Business Process Intelligence (BPI2010)*, volume 66 of *Lecture Notes in Business Information Processing*, pages 158–169. Springer-Verlag, Berlin, 2011.

[29] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens. A Robust F-measure for Evaluating Discovered Process Models. In N. Chawla, I. King, and A. Sperduti, editors, *IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2011)*, pages 148–155, Paris, France, April 2011. IEEE.

[30] M. Weidlich, A. Polyvyanyy, N. Desai, J. Mendling, and M. Weske. Process Compliance measurement Based on Behavioral Profiles. *Information Systems*, 36(7):1009–1025, 2011.

[31] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag, Berlin, 2007.