Enhancing Process Models to Improve Business Performance: A Methodology and Case Studies

Marcus Dees[1,2], Massimiliano de Leoni[2] and Felix Mannhardt[2]

[1] Uitvoeringsinstituut Werknemersverzekeringen (UWV), The Netherlands
[2] Eindhoven University of Technology, Eindhoven, The Netherlands
marcus.dees@uwv.nl,{m.d.leoni, f.mannhardt}@tue.nl

Abstract. Process mining is not only about discovery and conformance checking of business processes. It is also focused on enhancing processes to improve the business performance. While from a business perspective this third main stream is definitely as important as the others if not even more, little research work has been conducted. The existing body of work on process enhancement mainly focuses on ensuring that the process model is adapted to incorporate behavior that is observed in reality. It is less focused on improving the performance of the process. This paper reports on a methodology that creates an enhanced model that limits the incorporated behavior to only those parts that violate no business rules and, in light of keeping the model as simple as possible, that have a significant improvement on the performance level. The practical relevance and feasibility of the methodology is assessed through two case studies. The result shows that the process model improved through our methodology still adheres to the desired prescriptive model and guarantees better KPI levels, in comparison with what observed using state-of-the-art techniques.

1 Introduction

One of the main targets of every company is to continuously improve its business processes to lower the costs, increase the revenue, guarantee higher customer satisfaction, etc. Nowadays, the amount of available data (e.g., event logs) has grown to gigantic proportions, thus enlarging the range of possibilities of analysing how processes are executed and of finding bottlenecks, pitfalls, etc.

Process enhancement/improvement belongs to the realm of process mining [1], which aims to extract business knowledge from logging data that record the events linked to executions of business processes. The lion's share of attention of Process Mining has typically been about discovering models representing the actual executions of business processes as well as about checking the compliance/conformance of process executions against predefined normative models. However, less attention has been paid to process enhancement.

In this paper, we start from the belief that process improvement can happen through improving process models. Different process models can be used to discuss alternative ways to execute business processes. An improved process model describes an improved way to execute business processes. Not only can process models have a descriptive nature, but also a prescriptive purpose: process models can be used to automate the process executions and enforce how processes are executed. An improved process model will enforce a better way to execute business processes.

In this paper, improving a process model corresponds to repairing it to reflect reality. As discussed in Section 5, the existing body of work focuses on ensuring that the repaired model allows for all behavior observed in a reference event log. However, this is often too extreme. First, process models are initially designed to comply with laws and regulations; therefore, new behavior can be incorporated into the model only if these parts comply with laws and regulations. Second, one should include extra behavior that is linked to significant improvement in the performance: Adding the behavior that is not linked to significant better perform would just make the model unnecessarily complex.

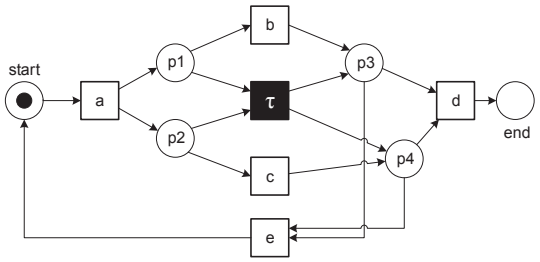
This paper provides a methodology to enhance a process model so as to incorporate observed behavior that is not allowed in the original model only if it is not in violation with laws and regulations and provides significant improvement of performances. Our methodology prevents behavior not related to good performance level from being incorporated into the model, so as to obtain models that are simple but, yet, guaranteeing good performance levels. The starting point is an existing process model and an event log. The event log consists of multiple traces, each of which records the events referring to one execution of the process. A Key Performance Indicator (KPI) is attached to each log trace and is typically associated to characteristics of the traces, such as its duration (e.g. the difference between the timestamp of last and first event), the value of certain attributes or the number of events (in total or related to a certain activity). The methodology builds on top of the model-repair techniques proposed by Fahland et al. [2]. In order to determine which deviations to incorporate as part of the model, the model-repair technique is combined with classification-tree learning techniques to determine if and what kind of correlation exists between KPI levels and the observed behavior. The model is enhanced by incorporating not-allowed behavior that is correlated with better KPI levels and that is not violating laws and company regulations.

The benefits and the practical feasibility of the methodology are assessed through two case studies. The first case study is performed with UWV, a company which provides unemployment benefits for Dutch residents. In particular, the case study illustrates how certain deviations are worth including in the enhanced process model whereas others must not be incorporated as they are either leading to poor KPIs or are not compatible with the Dutch laws. To illustrate a more generally applicability beyond the single case of interest, we also report on improvement for a SAP procurement process.

Section 2 introduces the existing body of work on which our methodology builds. Section 3 illustrates the different steps of the methodology. Section 4 shows the application of the methodology. Section 5 analyses the state of the art; finally, Section 6 concludes the paper.

2 Preliminaries

The research reported in this paper builds on a body of existing research in process mining. We assume that models are represented as Petri nets (see Section 2.1). Our methodology uses alignments of Petri nets and Event Logs to pinpoint deviations (see Section 2.2).



$$\gamma_1 = \begin{array}{|c|c|c|c|c|c|} a & d & b & c & \gg \\ \hline a & \gg & b & c & d \end{array}$$

$$\gamma_2 = \begin{array}{|c|c|c|c|c|} a & \gg & d & b & c \\ \hline a & \tau & d & \gg & \gg \end{array}$$

$$\gamma_3 = \begin{array}{|c|c|c|c|c|c|c|} a & \gg & d & \gg & b & c & \gg \\ \hline a & \tau & \gg & e & b & c & d \end{array}$$

(a) (b)

Fig. 1. A Petri net (a) and three examples of alignments of trace $\langle a, d, b, c \rangle$ with it (b)

2.1 Petri nets and Event Logs

Process models describe how activities in the process must be performed. Our approach is applicable to any modelling language. Here, we opt for the Petri net modelling language because it has simple and clear semantics.

Definition 1 (Petri net). *Let P be a set of places, T a set of transitions and $F \subseteq (P \times T) \cup (T \times P)$ a flow relation between places and transitions (and between transitions and places). A **Petri net** N is a tuple $N = (P, T, F)$.*

Figure 1(a) shows an example of a Petri net. In a Petri net, transitions represent process activities. The only exceptions are the *invisible* transitions, which do not represent pieces of the process' work but are necessary to properly model the process. These transitions are not recorded by the IT system. For further information, readers are referred to [1, 3]. For the Petri net in Fig. 1(a), transition names are depicted inside the transitions.

Places contain tokens; while the structure of the Petri net never changes, tokens are created and consumed. A transition is enabled if at least one token exists in each input place of the transition. By executing (i.e., firing) a transition, a token is consumed from each input places and a token is produced for each of its output places. The Petri net structure is static but the number of tokens in each place can change over time. The state of a Petri net is determined by the distribution of tokens over places, i.e. *the marking*. Processes have a precise initial and final state. Hence, when Petri nets represent business processes, they have an initial and a final marking. For the Petri net in Fig. 1(a), the markings with respectively one token in place *start* or in place *end* are the initial and final marking (and no tokens in any other place). A complete firing sequence is a sequence of transitions leading from the initial marking to the final marking. It indicates a complete execution of a process instance. The set of all complete firing sequences of a Petri net M is denoted by Ψ_M. Real executions of processes are recorded by information systems in event logs:

Definition 2 (Event, Trace, Log). *Let $N = (P, T, F)$ be a Petri net. Let $\mathcal{E}$ be a set of events. Each event $e \in \mathcal{E}$ corresponds to the firing of a transition $trans(e) \in T$. A*

trace $\sigma \in \mathcal{E}^*$ *is a sequence of events. An* **event log** $\mathcal{L}$ *consists of a set of traces, i.e.* $\mathcal{L} \in \mathbb{B}(\mathcal{E}^*)$.

2.2 Aligning Petri Nets and Event Logs and Repairing Event Logs

Not all event traces in an event log may be reproduced by a Petri net, i.e. not all event traces may correspond to a process trace, i.e. a complete firing sequence. Conformance checking aims to verify whether the observed behavior recorded in an event log matches the intended behavior represented as a process model. The notion of **alignments** [4] provides a robust approach to conformance checking, which makes it possible to pinpoint the deviations causing nonconformity. For this aim, the events in the event log need to be related to transitions in the model, and vice versa. Building this alignment is far from trivial, since the log may deviate from the model at an arbitrary number of places. We need to relate "moves" in the log to "moves" in the model in order to establish an alignment between a process model and an event log. However, it may be that some of the moves in the log cannot be mimicked by the model and vice versa. We explicitly denote such "no moves" by $\gg$.

Definition 3 (Alignment Moves). *Let* $N = (P, T, F)$ *be a Petri net and* L *be an event log. A legal* alignment move *for* N *and* L *is represented by a pair* $(s_L, s_M) \in (T \cup \{\gg\} \times T \cup \{\gg\}) \setminus \{(\gg, \gg)\}$ *such that:*

- (s_L, s_M) *is a* move in log *if* $s_L \neq \gg$ *and* $s_M = \gg$,
- (s_L, s_M) *is a* move in model *if* $s_L = \gg$ *and* $s_M \in T$,
- (s_L, s_M) *is a* synchronous move *if* $s_L = s_M$.

An alignment is a sequence of alignment moves:

Definition 4 (Alignment). *Let* $N = (P, T, F)$ *be a Petri net with an initial marking and final marking denoted with* m_i *and* m_f *Let also* L *be an event log. Let* LA_N *be the universe of all alignment moves for* N *and* L. *Let* $\sigma_L \in L$ *be a log trace. Sequence* $\gamma \in LA_N^*$ *is an* alignment *of* N *and* σ_L *if, ignoring all occurrences of* $\gg$, *the projection on the first element yields* σ_L *and the projection on the second yields a sequence* $\sigma'' \in T^*$ *such that* $m_i \xrightarrow{\sigma''} m_f$.

A move in log for a transition t indicates that t occurred when not allowed; a move in model for a visible transition t indicates that t did not occur, when, conversely, expected. Many alignments are possible for the same trace. For example, Fig. 1(b) shows three possible alignments for a trace $\sigma_1 = \langle a, d, b, c \rangle$. Note how moves are represented vertically. For example, as shown in Fig. 1(b), the first move of γ_1 is (a, a), i.e., a synchronous move of a, while the the second and fifth move of γ_1 are a move in log and model, respectively. We aim at finding a complete alignment of σ_L and N with minimal number of deviations for visible transitions, also known in literature as **optimal alignment**. Aligning an event log L means to compute an optimal alignment for each trace $\sigma_L \in L$. Clearly, different traces in L generally have different alignments, because they contain different events and deviations. With reference to the alignments in Fig. 1(b), γ_1 and γ_2 have two moves in model and/or log for visible transitions (γ_1 has one move in model for an invisible transition but it does not count for computing optimal alignments). Conversely, γ_3 has three moves for visible transitions, specifically one log move

Algorithm 1: Repair Event Log

Input: Event Log $\mathcal{L} \in \mathbb{B}(\mathcal{T})$, a Petri Net Model N, a set of legal moves that should not be repaired $D \subseteq A_{LM}$.
Result: Repaired Event Log

$\mathcal{L}' = \{\}$
foreach $\sigma_o \in \mathcal{L}$ **do**
  $\gamma = computeAlignment(N, \sigma_0)$
  $\sigma \leftarrow \langle\rangle$ // the repaired trace
  for $i \leftarrow 1$ **to** *length* (γ) **do**
    $(l, m) \leftarrow \gamma(i)$
    if $(l, m) \in D \wedge m = \gg \wedge\, l \neq \gg$ **then** // if log move not to be repaired
      $\sigma \leftarrow \sigma \oplus \langle l \rangle$ // keep the event l
    else if $(l, m) \notin D \wedge m \neq \gg \wedge\, l = \gg$ **then** // if model move not to be repaired
      $\sigma \leftarrow \sigma \oplus \langle m \rangle$ // add the missing event m
    else if $m \neq \gg \wedge\, l \neq \gg$ **then** // if synchronous move
      $\sigma \leftarrow \sigma \oplus \langle l \rangle$ // keep the event l
    end
  end
  $\mathcal{L}' \leftarrow \mathcal{L}' \cup \{\sigma\}$ // add the repaired trace to the repaired log
end
return $(\mathcal{L}')$

for d (3rd move) and two model moves for e (4th move) and d (last move). Since no alignment exists with only one move in model and/or log for visible transitions, both γ_1 and γ_2 are optimal alignments and any can equally be returned. For the sake of space, we assume here that all deviations (i.e., moves in model for visible transitions and moves in log) have the same severity. In [4], we show how this assumption can be removed. In the remainder, given a trace σ and a Petri net N, function $computeAlignment(N, \sigma)$ non-deterministically returns one of the optimal alignments for σ and N.

Alignments can be used to **repair a process model**. In Fahland et al. [2] techniques are presented to repair a model so the it can replay all behavior of the event log. Alignments can also be used to **repair an event log**. Repairing an event log consists of repairing every trace of the log. A trace can be repaired by taking the process projection of the computed optimal alignment after removing all moves on model for invisible transitions. For example, trace σ can be repaired as $\langle a, b, c, d \rangle$ if we consider alignment γ_1 in Fig. 1(b) as optimal alignment. Please note that several optimal alignments are possible and, hence, the trace can be repaired in multiple ways. However, if many optimal alignments are possible, each optimal alignment has the same probability to be chosen. Because of this, given the large amount of traces, the choice would not influence the applicability of the methodology discussed in Sect. 3. Repairing a model move in a trace means adding the missing event to the original trace. A log move is repaired by removing the event from the original trace. This means that, after repairing the event log, the number of traces does not vary. However, traces can become shorter or longer, depending on whether the alignment respectively contained moves on log or moves on model. When repairing an event log wrt. an alignment, it is also possible to not repair a selection of the deviations. For log or model moves, this respectively means that the corresponding event is kept or not added. For trace σ, not repairing deviations

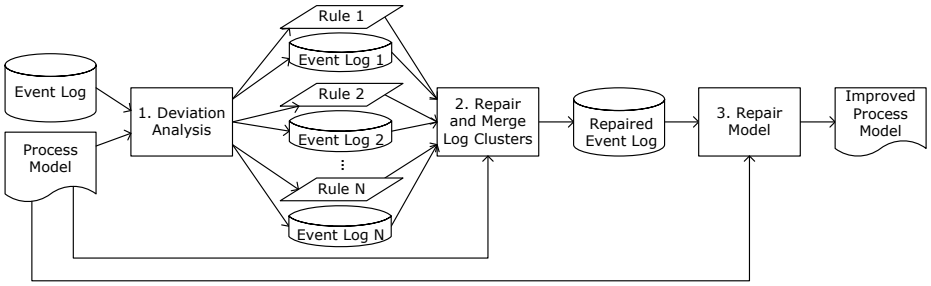


Fig. 2. The general approach

for d, would lead to the following repaired trace: $\langle a, d, b, c\rangle$, i.e. event d is not added for the corresponding move on model at the end of σ, nor is event d removed in the second position of the trace. Algorithm 1 describes how repairing an event log is done.[3] The input consists of an event log to be repaired, a Petri net model and a set of legal moves, i.e. deviations, that should not be repaired as inputs. The result is a repaired event log where all deviations, except the ones used as input, are fixed.

3 The Methodology

Figure 2 shows the steps of our methodology to enhance a process model. The basic inputs are an event log and a Petri net. The Petri net can be resulting from applying process discovery techniques or may have been designed by process owners according to how the process is expected to be executed.

Step 1. Deviation Analysis Deviations are detected and a set of rules is discovered that correlate deviations to a selected KPI, which is an event log attribute. The event log is clustered according to the rules: for each rule, the corresponding cluster contains all traces that comply with the rule.

Step 2. Repair and Merge Log Clusters Traces in the different clusters are repaired to only retain those deviations that have a positive impact on the value of the KPI. All log clusters are then merged to obtain a single repaired event log.

Step 3. Repair Model Finally the repaired log is used as input to repair the model: the process model is modified in such a way that it can replay all the behavior of the repaired event log. In the repaired event log we have repaired all deviations corresponding to behavior that should not be incorporated in the model. In this way the repair-model technique will only modify the model to make the desired deviating behavior possible.

Sections 3.1, 3.2 and 3.3 provide further details of each of the three steps of the methodology.

[3] In the algorithm, symbol $\oplus$ identifies the concatenation of two sequences. Given an alignment γ, $\gamma(i)$ denotes the i-th element of the alignment

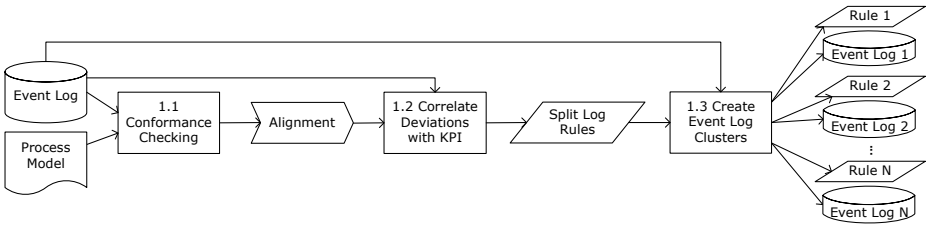


Fig. 3. Details of the Deviation Analysis.

3.1 Deviation Analysis

The aim of the deviation analysis is to determine which deviations have a positive effect on the process performance. To measure process performance, we introduce the concept of Key Performance Indicators:

Definition 5 (Key Performance Indicator). *Let $\mathcal{L}$ be an event log. Let $\mathcal{U}$ be the universe of possible values for a key performance indicator. A* key performance indicator *is a pair (κ, K) consisting of a function $\kappa : \mathcal{L} \to \mathcal{U}$ that assigns a KPI value $\kappa(\sigma)$ to each trace σ and of a set $K \subset \mathcal{U}$ that contains the KPI values that are satisfactory from a business viewpoint.*

Typically, the KPI value of a trace corresponds to or is a function of the attributes present in the event log. However, in this paper we remain general on how the KPI values of process executions (i.e., traces) are computed.

Step 1.1 The first step is checking conformance of the event log and the process model. This is done to determine all deviations that are observed between the log and the model. The result of conformance checking is an alignment as discussed in Section 2.2.

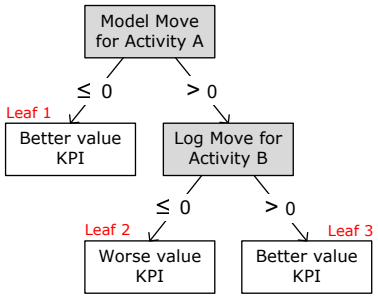


Fig. 4. Example of a decision tree built only upon model moves and log moves. A name is assigned to leaves for easy reference.

Step 1.2 The number of model moves and log moves for the recorded activities is correlated with the chosen KPI. As indicated below, we want to ensure that, when the model is improved, it remains compliant with rules and regulations. This step require analysts to provide us with a set of *disallowed activities* (i.e., transitions) $G_D \subseteq T$ that should never become part of the process model as well as with a set of *mandatory activities* be $G_M \subseteq T$, which should never become optional or be removed from the model.

During this step, we build a set of so-called *observation instances*, which are used to train a classification tree. We build one observation instance for each trace $\sigma \in \mathcal{L}$.

Given a trace $\sigma \in \mathcal{L}$ and a key performance indicator (κ, K), an observation instance is built with the following features:

- The number of model moves in the optimal alignment of T for each allowed activity $a \in T \setminus G_D$.
- The number of log moves in the optimal alignment of T for each non-mandatory activity $a \in T \setminus G_M$.
- The KPI value for σ, namely $\kappa(\sigma)$.

From the set of observation instance, we learn a classification tree, using the KPI component as dependent variable, namely to be predicted, and the number of log and model moves as independent variables.[4]

Consider, for example, the decision tree in Figure 4, which is possibly constructed after computing the optimal alignments and, subsequently, the classification tree. Each leaf is associated with a better or worse value of the KPI. In this example, to keep the explanation simple, we assume to only have two values for the KPI (i.e., $\mathcal{U}$ contains two values): a better value or a worse value. The leaves that are related to a better value are selected and can be interpreted as follows: *Activity A* should never be skipped (leaf 1) or when skipped, *Activity B* should be executed (leaf 3).

In the remainder, we represent the decision trees that are computed in this set as classification rules:

Definition 6 (Classification Rule). *Let A_L and A_M be the set of all log and model moves, respectively. Let (κ, K) be a KPI defined over a universe $\mathcal{U}$ of possible values. A* classification rule *is a tuple $(f, v) \in ((A_L \cup A_M) \not\rightarrow 2^{\mathbb{N}_0}) \times \mathcal{U}$.*

For the example of Fig. 4 the leaves of the decision tree can be described as the following classification rules:[5]

- $Rule_1 = (\{(\gg, A) \rightarrowtail \{0\}\}, BetterKPI)$
- $Rule_2 = (\{(\gg, A) \rightarrowtail \mathbb{N}, (B, \gg) \rightarrowtail \{0\}\}, WorseKPI)$
- $Rule_3 = (\{(\gg, A) \rightarrowtail \mathbb{N}, (B, \gg) \rightarrowtail \mathbb{N}\}, BetterKPI)$

The determination whether a given resulting decision tree is satisfactory, e.g. in terms of f-score support, is here left to the analyst, who can use domain knowledge to assess the quality of the tree.

Step 1.3. The classification tree can be seen as a clustering of the traces of an event log. Each leaf is a different cluster and the path from the root to the leaf provides a clustering rule. For reliability, for each cluster, we discard the event log traces that are wrongly classified, namely which are classified to have KPI values that are actually different from the actual observed values. The wrongly-classified traces might potentially affect the repair-model phase by allowing behavior in the model that would not be actually linked to good KPI values. The next definition defines how these log-trace clusters are created:

[4] Here we talk in term of classification tree, which can be a decision or regression tree. Decision trees are used when the KPI values are discrete; otherwise, we use regression tree.

[5] Notation $\{d_1 \rightarrowtail c_1, \ldots, d_n \rightarrowtail c_n\}$ indicates a function f with domain $\{d_1, \ldots, d_n\}$ in which $f(d_1) = c_1, \ldots, f(d_n) = c_n$.

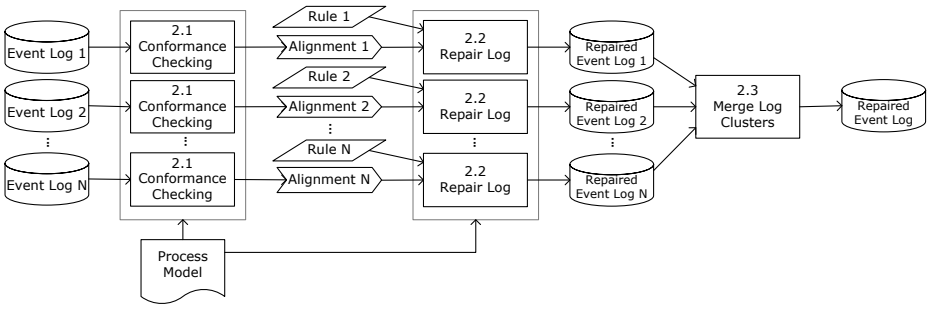


Fig. 5. Details of Repair and Merge Log Clusters.

Definition 7 (Log Trace Clustering). *Let (κ, K) be a KPI. Let $\{(f_1, v_1), ..., (f_n, v_n)\}$ be the set of decision tree rules for a tree with n leaves. For each (f_i, v_i), a log trace cluster $\mathcal{L}_i$ is created and contains all traces $\sigma \in \mathcal{L}$ such that $\kappa(\sigma) = v_n$ and, for each move type $a \in A_L \cup A_M$, the number of moves of type a in the optimal alignment $\gamma = computeAlignment(N, \sigma)$ is $\#_a \in f_i(a)$.*

3.2 Repair and Merge Log Clusters

The log clusters that have been created in the Step 1.3 now need to be repaired to reflect the repair rules. After that, the log clusters need to be merged to present a single repaired log for usage in Step 3. Figure 5 shows the details of this phase of the approach.

Step 2.1 Conformance Checking is done with the original process model and each log cluster. The alignment reproduces the deviations that were part of the Conformance Checking result of Step 1.1.[6]

Step 2.2 This step is repeated for each cluster L_i, associated with a classification rule (f_i, v_i). If L_i is linked with a satisfactory KPI value (i.e., $v_i \in K$), we repair all deviations in each trace of L_i, except for those being part of the repair rule associated with the log cluster. Otherwise, in case of unsatisfactory values, we repeat all deviations. For each cluster, we repair all deviations linked to behavior that are not correlated to improved KPI values and/or that would be violating laws and re. This guarantee that each repaired log cluster only retains those deviations that, on the one hand, are worth incorporating in the enhanced model and, on the other hand, would create a model that still adheres to the applicable laws and regulations. The other deviations are repaired and, hence, not eligible for repair in the next methodology steps. For instance, consider Fig. 4. For the event log cluster associated with Leaf 3 we repair all deviations except model moves for *Activity A* and log moves for *Activity B*, as the combination of model moves for *Activity A* and log moves for *Activity B* leads to a better KPI value. For the event log cluster associated with Leaf 2, we repair all deviations because the KPI value is poor. For the event log cluster associated with Leaf 1, we also repair all deviations

[6] Step 2.1 is a conceptual step. In practice, one does not need to recompute the alignments for the cluster logs as one can simply reuse the alignments obtained as result of Step 1.1

except for $ModelMove for Activity A$. However, the clustering rule says that all cluster traces have no model moves for *Activity A*; hence, in fact, all deviations are also repaired.

Definition 8 (Repaired Log Cluster). *Let L_i be an log trace cluster created on the basis of a classification rule (f_i, v_i). Let N be a Petri net model and (κ, K) a KPI. The corresponding* Repaired-Log *cluster is* $L'_i = \begin{cases} repairLog(L_i, N, dom(f_i)) & \text{if } v_i \in K \\ repairLog(L_i, N, \emptyset) & \text{otherwise} \end{cases}$

Step 2.3 We merge all repaired log clusters into a single event log. This is a requirement to apply the next step, namely repairing the process model.

3.3 Repair Model

This phase repairs the model. The input is the original model and the repaired log obtained as result of Step 2.3. The repaired log is equal to $\bigcup L'_i$. The process model is modified in such a way that it can play out all the behavior of the repaired event log. Consider again the example in Fig. 4: the model is modified so that *Activity A* may be skipped and *Activity B* is now allowed at certain points of the process (Leaf 3 in the figure). These modifications can be incorporated into the model because they have shown to achieve better KPI values.

Unfortunately, we cannot always manually modify the model to allow for extra behavior because it is not always clear which part of the model should be modified. For instance, for the example referring to Fig. 4, *Activity B* should become allowed at certain points of the process. But, at which points exactly? As another example, the same activity - say X - may be prescribed to happen at different points of the process. From the classification tree, we can hypothetically observe that activity X can be skipped while the KPI becomes better. However, at which point can activity X be skipped? In the light of above, we leverage on the repairing technique by Fahland et al. [2] to determine how to modify the model. This technique will modify the model so as to obtain a new model that can replay all the behavior observed in the repaired event log. The repairing technique allows one to apply a filtering such that the repaired model only incorporates the behavior that is observed with a frequency higher than a given threshold. For our methodology, this option is very interesting: if a certain deviating behavior has shown to guarantee good KPI levels, that is allowed by the repaired model only if it is sufficiently often recorded in the event log. Only behavior with enough evidence, namely occurring in a sufficient enough number of traces, should be kept.

4 Implementation and Evaluation

The entire methodology can be carried out through ProM 6.6[7], which is an extensible tool that supports a wide variety of process mining techniques in form of plug-ins. In particular, Steps 1.1 and 2.1 can be carried on through the plug-in *Replay a Log on Petri Net for Conformance Analysis*, which operationalizes the technique discussed in [4]. Steps 1.2 and 1.3 are implemented as plug-in *Perform Predictions of Business Process Features* (see also [5]). Step 2.2 is implemented as plug-in *Repair Log With Respect*

[7] `http://www.promtools.org/doku.php?id=prom66`

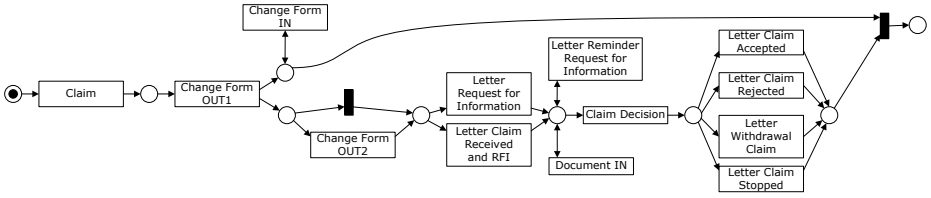


Fig. 6. The original process model of the unemployment benefits claim handling process at UWV.

to Alignment; step 2.3 is implemented through a number of plug-ins for log manipulation. Finally, Step 3 is implemented as plug-in *Repair Model*, which operationalizes the technique by Fahland et al. [2].

We assessed our methodology through two case studies. Section 4.1 reports on a first case study in collaboration with UWV (Uitvoeringsinstituut Werknemersverzekeringen) and refers to the provisioning of unemployment benefits for the Netherlands' residents. A second case study is presented in Section 4.2 and refers to a procurement process as implemented in a SAP system.

4.1 UWV case study

UWV is the social security institute of the Netherlands and responsible for the execution of a number of employee related insurances. The case study focuses on the unemployment benefits process of UWV. When employees become unemployed, they may be entitled to the benefits. Employees have to file a claim at UWV. UWV then decides whether they are entitled to benefits. When claims are accepted employees receive benefits with a regular frequency until they find a new job or the maximum period for their entitlements is reached. UWV refers to employees who are making use of their services as customers, therefore we use the term customer in the remainder of the paper.

UWV translates the legal text of the law into a process design that is executable. Within the boundaries of the law there is some flexibility in the way the process can be designed and executed. UWV can for example choose how to communicate with its customers. Communication can happen through several channels like internet, a letter or a telephone call. On the other hand, some parts of the process need to adhere to the Dutch laws.

Fig. 6 shows a prescriptive process model that encodes the relevant Dutch laws and the UWV's protocols. The process model was designed in collaboration with a UWV's process specialist.

In particular, UWV aims to improve the process by reducing the claim's throughput time, as this would likely reduce the costs and improve the satisfaction of customers, who receive a faster answer to claims. Therefore, **the throughput time is the KPI that we aim to minimize** by employing our methodology. Regarding the definition of KPI (κ, K), $\kappa(\sigma)$ is defined as the timestamp of the last event of σ minus the timestamp of the first event in σ, and K contains any time interval smaller than 8 days.

Together with a process model, our methodology requires an event log. The event log consists of 25476 traces and has 161365 events in total, which refer to 21 different activities. The event log recorded executions of several activities that are not part of

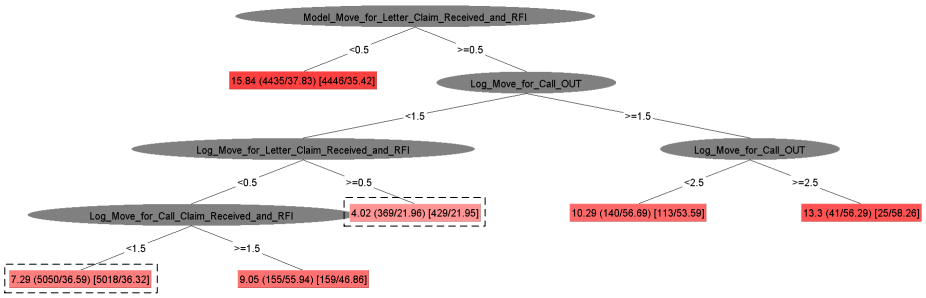


Fig. 7. Regression Tree for the case study that correlation the deviations with the throughput time. Two leaves with the lowest throughput time are selected as repair rules. They are highlighted with a dashed box.

the original model, for example: *Call Claim Received and RFI, Letter Information to Customer, Call Information to Customer* and *Call OUT*. These events were not foreseen in the reference model, but, in fact, they were executed by UWV employees. During the application of our methodology, we will hence investigate whether some of these activities ought to be added to the process model. In particular, we aim to incorporate any of those activities in the right positions if their execution has shown to lead to good KPI levels. From a business perspective, only the following model activities can be made optional or made possible at a different point of the process execution: *Letter Claim Received and RFI, Letter Request for Information, Letter Reminder Request for Information*. All other activities in the model are mandatory and may not be moved to other positions or removed. Similarly, only the following log activities may be added to model: *Call Claim Received and RFI, Call Information to Customer, Call OUT, Letter Information to Customer*.

For the validation, the event log was randomly split in two groups: 80% of the traces are used to improve the model using our methodology and 20% of the traces are used later to test the quality of the improved model. This is, in fact, performed in line with the validation techniques employed, e.g., in data mining.

The first step of the approach is the deviation analysis. It consists of building alignments between the model in Fig. 6 and the extracted event log (Step 1.1.). The alignments are used along with the event log to correlate the deviations with the KPI values (Step 1.2 in Fig. 3). The resulting tree is shown in Figure 7. We used a regression tree because the dependent variable, throughput time of the claim process, is an attribute that has a numeric data type. The tree has six leaves. There are two leaves with an average throughput time below the desired value of 8 days, i.e., the attribute used as KPI. For those two leaves, we generated the corresponding log clusters and we repaired the deviations that are not present in the regression tree, in accordance with the methodology step defined in Definition 8. Also, for the other leaves, we repair every deviation. Only correctly classified traces for each leaf are selected to be exported into a log cluster. The error threshold is set to 15%. In this way, we retain the traces referring to process executions with a KPI value relatively close to the expected value of the log cluster. The other traces are considered to be noise, which can negatively affect the final result of Step 3 (the application of the model-repair technique), obtaining a model that incor-

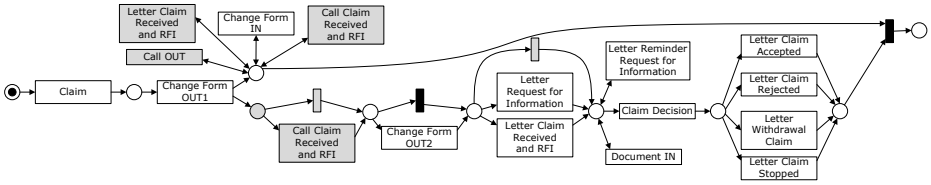


Fig. 8. Improved model using our methodology. The gray transitions are those which are added to the model.

porates behavior that has not been associated with better KPI values. The log clusters are merged into a new log (Step 2.3) that is compliant with the model, except for the deviations that are correlated with a better KPI value.

Finally the repaired event log is used together with the original model as input to improve the model. Figure 8 shows the improved model. Activities *Call Claim Received and RFI*, *Call OUT* and *Letter Claim Received and RFI* are allowed to be executed multiple times after the first change form has been sent. Before *Change Form OUT2* now *Call Claim Received and RFI* is optionally allowed to be executed. Finally, the activities *Letter Claim Received and RFI* and *Letter Request for Information* are now allowed to be skipped. In essence, the repaired model represents the fact that calling the customer for a confirmation of a claim reception along with, when necessary, asking for extra information, is faster than sending a letter. This is also understandable: waiting for a customer to respond to a letter would take time. When the customer ultimately does respond the employee doing the handling has to get back into the details of the claim. By calling the customers, the employee can, most of the time, handle the claim in one go without having to wait.

We assessed the usefulness of our methodology by comparing this model with the model obtained by only employing the model-repair technique by Fahland et al. [2]. For a fair comparison, the same parameters where used, namely only considering the traces that occur at least 50 times. The model obtained without our methodology is available in Figure 9. The model contains several activities that can be executed an arbitrary number of times in any order; e.g. see the part of the model that is within the dashed box. Clearly, this model is underfitting the event log because it allows for a lot of behavior that is not observed in reality. Also, the model also allows behavior that violates Dutch laws as well as UWV's internal regulations and protocols. This clearly manifests itself in the fact that sending one of the mandatory claim-outcome letters can be skipped. As final validation, we measured the KPI level of the traces that record executions compliant with the original model and we compared with the KPI levels of the executions compliant with the enhanced model, both with our methodology and with the technique by Fahland et al. [2]. As mentioned above, we employ the 20% of log traces that were left aside, hereafter named test log.

If we consider the traces of the test log that are compliant with the original model in Fig. 6, the average throughput time is 15.44 days. If we consider traces of the test log that are compliant with the model enhanced with our methodology in Fig. 8, the average throughput time is 10.88 days, leading to an improvement of 29.5%. With the model enhanced with only the technique by Fahland et al. [2] in Fig. 9, the average is

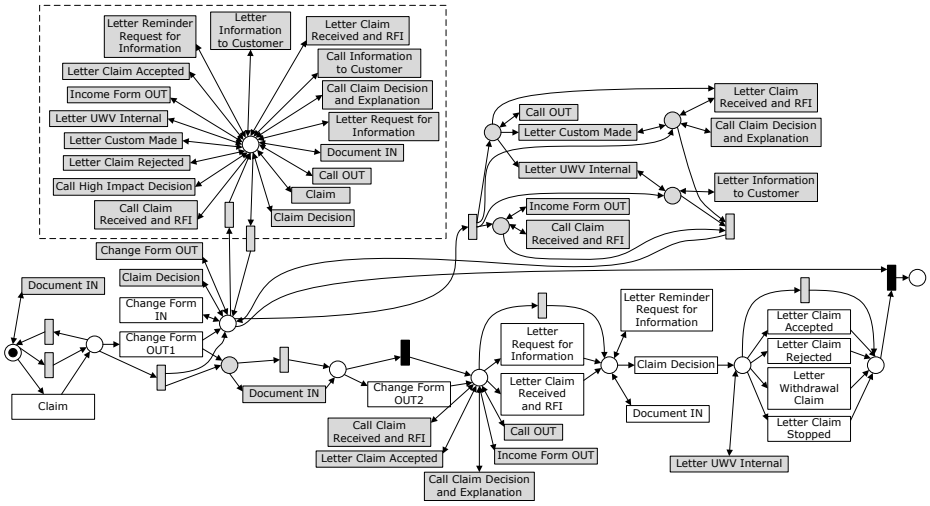


Fig. 9. Improved model based on the Fahland technique. The gray transitions are those which are added to the model.

11.18 days, leading to a decrease of throughput time of around 27.6% compared with the original model. It is clear that the overall improvement with our methodology is marginal compared with the technique by Fahland et al., if we look at the improvement of the KPIs, only. However, first, our model is compliant with regulations, whereas the other is not as discussed. Our model is clearly simpler and, very importantly, it is not underfitting, allowing for a lot of behavior that should be disallowed. If that behavior is incorporated, , the model becomes less insightful on which behavior is really linked to better KPI levels. In other words, the extra allowed behavior is not really all guaranteed to be connected to better KPI levels.

4.2 SAP Procurement case study

The procurement process is part of the SAP Material Management (SAP MM) module. This module supports the procurement and inventory functions occurring in day-to-day business operations. Figure 10 shows the process model we used in this case study. During this process changes may occur to the price of the goods, the quantity and the vendor where the goods are ordered. Changes incur higher costs. Therefore, the KPI that we want to **minimize** is **the number of purchase orders with a change**.

All activities that are shown in the model in Fig. 10 are present in the event log. Additionally there are some activities that are not part of the model, but exist in the event log. Examples of these are *Delete Purchase Order Item* and *Send Overdue Notice*. The event log is created consisting of 105708 traces. In the same way as for the case study with UWV discussed in Section 4.1, we split the event log in a training log with 80% of the remaining traces and in testing log that accounts for 20% of the traces. For the sake of space, we cannot discuss all the intermediate steps and we limit ourselves to discuss the final models. Fig. 11 shows the model obtained through the technique by Fahland et

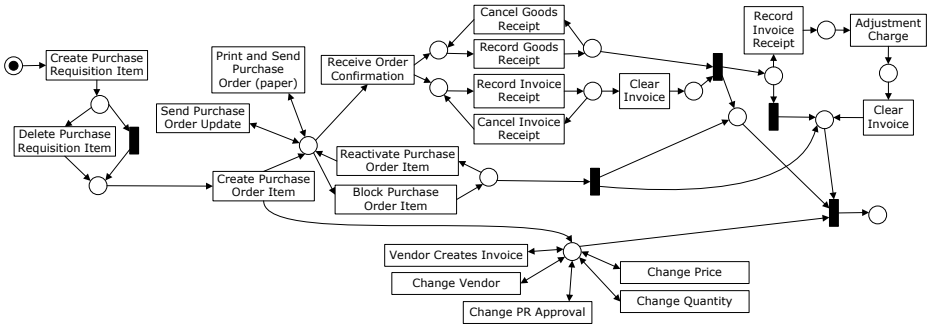


Fig. 10. The original process model of the SAP procurement process.

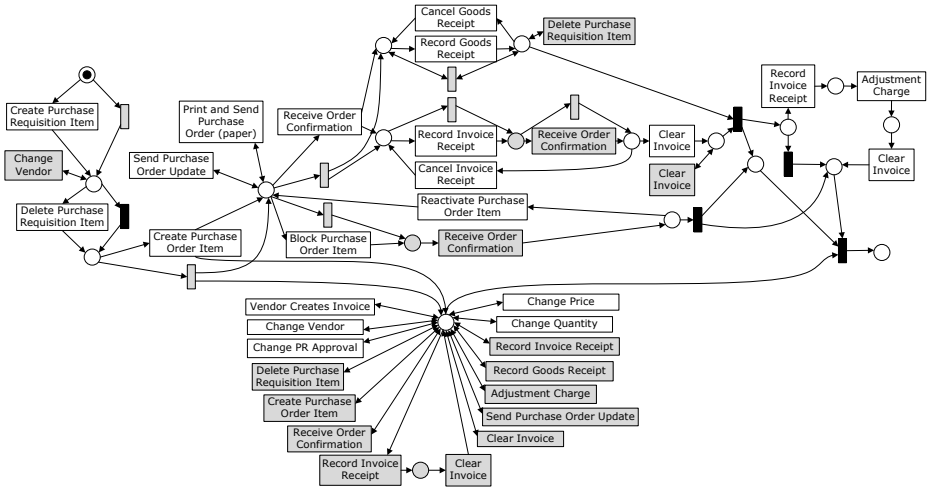


Fig. 11. The repaired process model of the SAP procurement process obtained by the Fahland technique. The gray transitions are those which are added to the model.

al. whereas Fig. 12 illustrates the model obtained through our methodology. The model obtained through our methodology is clearly simpler and easier to understand.

If we employ the testing log and apply it on the original model, then 52% of the fitting cases has a change. When the same testing log is applied to the model in Fig. 11 obtained by the technique by Fahland et al., 38% of the cases have a change. Finally, when it is applied to the model obtained through our approach in Fig. 12, 44% of the cases have a change. While our methodology guarantees an improvement of KPI between the original model and the one enhanced by by the technique by Fahland et al., it is again clear that our model is more valuable, in term of simplicity and of capacity of not being underfitting, with all the positive considerations drawn for the UWV case studies at the end of Section 4.1. It is also worth highlighting that we still have an improvement of 8% for what concerns the number of change/rework of the purchase orders.

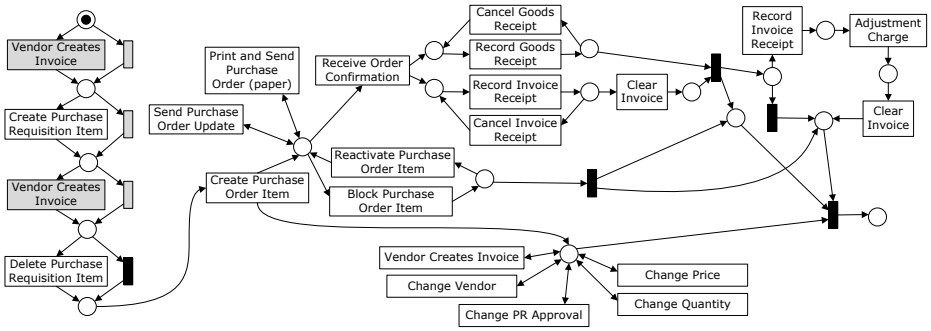


Fig. 12. The repaired model for the SAP procurement process using our approach. The gray transitions are those which are added to the model.

5 Related Work

Section 1 has already discussed one limitation of the model-repair techniques by Fahland et al. [2]: This technique considers all deviations to be eligible to be incorporated in the model, independently whether or not a certain deviation typically causes poor performance levels or violates laws and regulations. The evaluation showed that a lot of behavior is actually not necessary to obtain a model with good performance levels. The same limitation is also observed in works [6–8]; in particular, in [8] the decision whether or not to incorporate certain deviations into the model is driven by many quality dimensions that, however, do not include KPI fulfilment and adherence to legislations.

Schunselaar et al. [9] try to find a best configured process model given a set of configurable process models and a set of KPIs. Using a Pareto front, they select those process models that have the best performance on the set of KPIs. While this approach needs a set of configurable process models to capture variation in the KPIs values, our approach only needs one non configurable process model.

While our paper reports on the use of the techniques by Fahland et al. for repairing the process model, our methodology is not bounded to that repair technique only. We could alternatively use other techniques, such as that by Buijs et al. [8], which mediates several model quality criteria.

Similarly to [2], in [10] authors propose to repair the model while improving the correspondence, i.e. fitness, between model and log as much as possible. Costs are associated with repair actions and an optimal strategy is determined to minimize the costs of the repairs, but it is not possible to improve on a KPI. In [11], authors propose to repair a process model to make it similar to a collection of process models. Differently from our approach, they do not use an event log but a collection of models as input and try to minimise the edit distance to all models in the collection.

Repairing a process model can also be regarded as ensuring that the model is sound and, hence, does not contain deadlocks. Gambini et al. [12] and Lohmann [13] propose techniques along these lines. Repairing a process model can also be seen as simplifying the model while allowing for the same behavior, such as what is proposed by Fahland et al. [14]. Clearly, ensuring soundness and simplifying models pursue a different goal than what we aim at in our work.

The methodology proposed in this paper is also in line with the DMAIC methodology of Six Sigma [15], which comprises five steps: define the goals, measure key aspects, analyse data, improve and control the improvement. In fact, the first step of our methodology (Deviation Analysis) corresponds to the analyse data step of DMAIC and the second and third step corresponds to improving (the model of) the process. The DMAIC's step of measuring key aspects corresponds to having the KPI values as an event log attribute. The step of controlling the improvement is beyond the scope of this paper but corresponds to the natural follow-up once the model is improved. Namely one would like to verify whether the improved model actually leads to better KPI values.

6 Conclusion

Process mining is not just about discovering the control-flow or diagnosing non compliance. Enhancing business processes to improve the performance is perceived equally important. Enhancing a business process can be regarded from many perspectives. Here, we aim to repair the model thus incorporating some of the behavior that is observed in reality but disallowed by the model. However, the model should be extended so as to only allow additional behavior that *(i)* does not violate company rules and national regulations and that *(ii)* has shown to lead to better KPI levels. As discussed in Section 5, the existing body of work in the Process Mining context does not look at these aspects.

This paper has proposed a methodology that considers the two aspects above: the adherence to rules/laws and the potential KPI improvement. The methodology unifies an existing approach for model repair proposed by Fahland et al. [2] with other existing works in the field of conformance checking and deviation-to-KPI correlation.

A detailed evaluation has been carried out through two case studies. One case study based on the SAP procurement process and the other with UWV, a Dutch financial institute that provides social security benefits to the residents in the Netherlands. The results clearly show the practical usefulness of the methodology: we improve the SAP model so less changes will occur during the process, leading to lower costs. The UWV model is improved to allow all the behavior observed in the event log which is correlated with better KPI levels, i.e. a lower throughput time, while preventing violations of the Dutch unemployment-benefit laws.

As future work, we aim to extend our methodology so that the improved model only retains the behavior allowed by the original model that yields better KPI levels. Currently, our methodology only allows one to extend the model to incorporate behavior observed in reality that led to KPI improvements. Our methodology should be extended so that the new model forbids certain behavior allowed by the original model if it has shown to yield worse KPI levels. Also, it is worthy working further on our methodology to consider concept drift [16]. For instance, it would be relevant to investigate which drifts contribute to improve KPIs and to only incorporate those into the model. Since at the moment we can only improve one KPI at the time, we think that being able to improve multiple KPIs at the same time, is also a direction for future work.

Whereas the paper shows the relevancy of the methodology, one of the drawbacks is that all steps of the methodology need to be manually performed. This is clearly tedious and error-prone. This can be automated through the use of scientific workflows. Scientific Workflow Management systems help users to design, compose, execute, archive, and share workflows that represent some type of analysis or experiment. The advan-

tages of using scientific workflows for process mining are discussed in [17]. As future work, we will implement this methodology as an scientific workflow: This is far from being difficult as every step can be easily automated as an activity of a proper scientific workflow.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. 1st edn. Springer Publishing Company, Incorporated (2011)
2. Fahland, D., van der Aalst, W.M.P.: Model repair - aligning process models to reality. Information Systems (47) (2015) 220–243
3. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4) (Apr 1989) 541–580
4. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery **2**(2) (2012) 182–192
5. de Leoni, M., van der Aalst, W.M.P., Dees, M.: A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Information Systems **56** (2016) 235 – 257
6. Kalsing, A.C., do Nascimento, G.S., Iochpe, C., Thom, L.H.: An incremental process mining approach to extract knowledge from legacy systems. In: Proceedings of the 14th IEEE International Enterprise Distributed Object Computing Conference (EDOC), IEEE (2010) 79–88
7. Sun, W., Li, T., Peng, W., Sun, T.: Incremental workflow mining with optional patterns and its application to production printing process. Internation Journal Of Intelligent Control And Systems **12**(1) (2007) 44–55
8. Buijs, J.C.A.M., La Rosa, M., Reijers, H.A., van Dongen, B.F., van der Aalst, W.M.P.: Proceedings of the Second International Symposium Data-Driven Process Discovery and Analysis (SIMPDA 2012). Volume 162 of LNBIP., Springer (2013) 44–59
9. Schunselaar, D.M.: Configurable process trees : elicitation, analysis, and enactment. PhD thesis, Eindhoven University of Technology, Eindhoven (2016)
10. Artem Polyvyanyy, Wil M.P. van der Aalst, A.H.t.H., Wynn, M.T.: Impact-driven process model repair. ACM Transactions on Software Engineering and Methodology (TOSEM) (2016)
11. Li, C., Reichert, M., Wombacher, A.: Discovering reference models by mining process variants using a heuristic approach. In: Proceedings of the 7th International Conference on Business Process Management. BPM '09, Berlin, Heidelberg, Springer-Verlag (2009) 344–362
12. Gambini, M., La Rosa, M., Migliorini, S., Ter Hofstede, A.H.M.: Automated error correction of business process models. In: Proceedings of the 9th International Conference on Business Process Management. BPM'11, Berlin, Heidelberg, Springer-Verlag (2011) 148–165
13. Lohmann, N.: Correcting deadlocking service choreographies using a simulation-based graph edit distance. In: Proceedings of the 6th International Conference on Business Process Management (BPM 2008). Volume 5240 of LNCS., Springer (2008) 132–147
14. Fahland, D., van der Aalst, W.M.P.: Simplifying discovered process models in a controlled manner. Information Systems **38**(4) (2013) 585 – 605
15. International Organization for Standardization: ISO 13053:2011 quantitative methods in process improvement - Six Sigma - part 1: DMAIC methodology (September 2011)
16. Bose, R.P.J.C., van der Aalst, W.M.P., Žliobaitė, I.: Handling concept drift in process mining. Advanced Information Systems Engineering **6741** (2011) 391–405
17. Bolt, A., de Leoni, M., van der Aalst, W.M.P.: Scientific workflows for process mining: building blocks, scenarios, and implementation. STTT **18**(6) (2016) 607–628