Software and Systems Modeling (2023) 22:1099–1111
https://doi.org/10.1007/s10270-022-01067-0

THEME SECTION PAPER

On the use of domain knowledge for process model repair

Kate Revoredo[1,2]

Received: 30 December 2021 / Revised: 5 October 2022 / Accepted: 3 November 2022 / Published online: 14 December 2022

Abstract
Process models are important for supporting organizations in documenting, understanding and monitoring their business. When these process models become outdated, they need to be revised to accurately describe the new status quo of the processes in the organization. *Process model repair* techniques help at automatically revising the existing model from behavior traced in event logs. So far, such techniques have focused on identifying which parts of the model to change and how to change them, but they do not use knowledge from practitioners to inform the revision. As a consequence, fragments of the model may change in a way that defies existing regulations or represents outdated information that was wrongly considered from the event log. This paper uses concepts from *theory revision* to provide formal foundations for process model repair that exploits domain knowledge. Specifically, it conceptualizes (1) what are unchangeable fragments in the model and (2) the role that various traces in the event log should play when it comes to model repair. A scenario of use is presented that demonstrates the benefits of this conceptualization. The current state of existing process model repair techniques is compared against the proposed concepts. The results show that only two existing techniques partially consider the concepts presented in this paper for model repair.

Keywords Process model repair · Process mining · Concept drift · Theory revision

1 Introduction

Business process management (BPM) [1] relies on process models to support organizations in documenting, understanding and monitoring their processes. These models are also important for model-driven software development [2], when it comes to managing various workflows around software development such as issue-to-resolution process, bug-fixing, version release-planning etc. Such models can be manually specified by stakeholders of the process or automatically discovered from process data (i.e., observed behavior: information of past process instances) traced in event logs, using process discovery techniques [3]. Over time, changes in the regulations, business or organizations culture, may make these models obsolete and less useful for monitoring. Thus, there is need to revise these process models to meet the new understanding of the business.

In the BPM area, there are some initiatives on process model repair [4] to automatically revise the current model from process data. They use event data collected from information systems to guide the necessary changes in the model. This is typically done by applying conformance checking [5] techniques for identifying non-conforming traces and applying local changes to the model to make it compliant to the analyzed traces. These techniques mainly focus on identifying where to apply the change and how to change the model. However, they overlook the importance of domain knowledge in guiding the revisions, for instance to avoid changing fragments of the model that represent a regulation and therefore should be modeled exactly as it was previously or by indicating cultural behavior that should be avoided. Although current techniques can learn a repaired model with high quantitative quality (e.g., maximal fitness), these models may not fully align with practitioners needs. When practitioner knowledge is used, the repair techniques lead to a repair model with better precision and alignment with the practitioners needs [6].

Communicated by Dalila Tamzalit and Ludovico Iovino.

✉ Kate Revoredo
kate.revoredo@hu-berlin.de

[1] Humboldt-Universität zu Berlin, Berlin, Germany

[2] Vienna University of Economics and Business (WU), Vienna, Austria

The necessary changes to the process model can have the goal to represent new behavior or to prevent undesired behavior. For instance, the organization may decide to switch to a more sustainable business and manage their internal projects digitally. In this scenario, it is important to distinguish in the historical data between desired and undesired behavior (i.e., behavior that the practitioners accept to happen and behavior that must not happen, respectively). That is, events from the previous physically managed cases should be marked as undesired, while events from the digitally handled process should be marked as desired behavior. The process model repair technique should take these markings into account to guarantee that the final model accurately represents the current business process. Another benefit of considering domain knowledge concerns specific fragments of the process which represent normative work (e.g., safety fallback procedures in case of emergency, organization code of conduct). This kind of work is described by activities which must be done exactly as described in the model. Also, some parts of the model were derived from extensive discussions among the different stakeholders and they represent a common understanding among the process participants. It is important that the model repair technique does not change these parts, so the final model is more aligned with practitioners needs. In this paper, I address the problem of how can domain knowledge from practitioners support process model repair. Based on concepts from Theory revision [7,8], I formulate how practitioner knowledge can be used as input for process model repair techniques.

Theory revision [7,8] is part of the *Inductive Logic Programming* [9] (ILP) area and it is motivated by *concept drifts* (i.e., the situation in which properties and relations of the studied data can change over time) [10]. It focuses on minimally changing a logical theory in the presence of positive and negative observations with the aim of finding a more accurate theory. Theory revision brings two advantages to the practitioner. First, it allows to distinguish between positive and negative observations (i.e., facts that must or must not be explained by the revised theory). Second, it allows to precisely specify parts of the model to be kept unchanged during the revision.

This paper aims at bridging *process model repair* and *theory revision*. More specifically, it provides the fundamental concepts needed to formulate a process model repair problem as a theory revision problem. To this end, it provides a schema which can be used to guide (i) the distinction between desired (positive) and undesired (negative) behavior, and (ii) changeable and unchangeable model fragments to be considered in the model repair. With that, it is expected that the final model is closer to the practitioners needs and with better quality characteristics. The idea of framing an existing task as a theory revision task has been used in other areas such as learning game rules [11], updating social network in the presence of stream data [12] and discovering links in real biological networks [13]. This paper is a first attempt to provide formal foundations for process model repair. Especially, it sheds light on the necessity of the fragmentation of the initial process model and the partition of the event log involving practitioners' knowledge. This paper extends initial ideas presented in [14], providing a more detailed formalization, a scenario of use and a systematic analysis of existing process model repair techniques exploring whether concepts of theory revision are already in use and which are further concepts that are useful for model repair.

The rest of the paper is structured as follows. Section 2 reviews preliminary concepts. Section 3 presents the formal conceptualization for process model repair based on theory revision. Section 4 illustrates the conceptualization with a scenario of use considering the *Process Discovery Contest 2021* dataset. Section 5 presents the empirical evaluation of the existing process model repair approaches, positioning them against the derived concepts. Section 6 briefly describes existing literature related to this work. Section 7 concludes the paper and outlines future work.

2 Preliminaries

The scope of this article is to indicate how practitioner knowledge can be input into process model repair. The motivation derives from the area of Theory Revision, where it is known that background knowledge improves the quality of the revised theory. In a business process context, background knowledge and revised theory are mapped into, respectively, practitioners knowledge and repaired process model. Practitioners in BPM [1] are also known as stakeholders who have an understanding of the process and are able to express various process-related concepts such as the goals, the activities, the resources, the time, the logical control flow, etc. Typically, these stakeholders cover jobs such as Process Owners, Process Managers, Business Analysts. A revised theory in BPM is translated as a repaired process model (i.e., a process model that was changed in its structure in order to conform/comply to new needs or requirement, or to best represent reality).

This section gives the fundamental background to understand how theory revision can be used for process model repair. More specifically, Sect. 2.1 briefly reviews the concept of process model repair, while Sect. 2.2 describes the concept of theory revision.

2.1 Process model repair

A *process model* (M) is a description of the business process. It represents the sequence of process events (i.e., the activities and control-flow of the process). Usually, this is represented as a directed graph where the nodes represent the events and

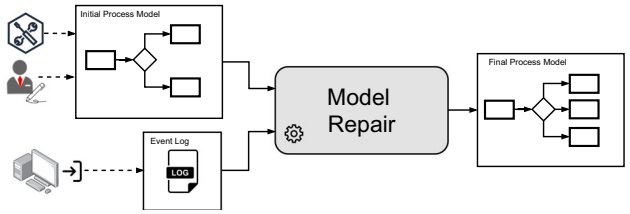

Fig. 1 Process model repair schema

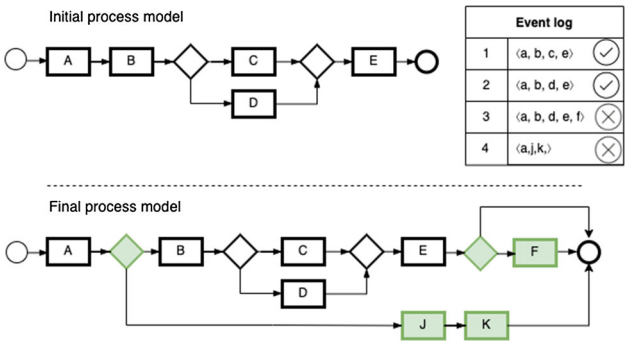

Fig. 2 Process model repair example

the edges represent the potential flow of control among the events. An *event log* (L) is a multi-set of traces and each trace (T) represents the execution of a process instance, (i.e., the sequence of process activities). Techniques for automatically discovering a process model from an event log were proposed in the literature [3]. The final process model is expected to conform to the event log (i.e., the model M is expected to replay all the traces T in L).

The event log represents the observed behavior while the model represents the expected behavior. Over time, the expected and the observed behavior may not align anymore, for instance because of *concept drift* [10]. In this case, *process model repair* [15] may be applied. Process model repair aims to improve the quality of a model through process data by applying minimal changes to the initial model. Existing methods take as input a model and an event log, and produce a new model that resembles the original one as much as possible while still guaranteeing that the new model is able to replay the traces in the log.

Figure 1 illustrates the process model repair schema.

In more detail, process model repair techniques need two inputs: a process model and an event log. A process model typically uses a graph-based notation to express the partial order relation of the activities within the different traces constituting the event log. Such model may have been designed manually by a person using a modeling tool or may have been generated by a process discovery tool. An event log typically comes from an information system (e.g., a BPMS, a database, etc). The data present in the event log may record events about a large amount of time, including time periods in which the process is enacted differently (i.e., the actual process changed with respect to its model). This means that the initial model is no longer able to accurately describe the behavior recorded in the event log, especially when it comes to newer traces. Thus, the task of model repair is that to produce a new process model that is able to best describe all the facts and relations observed in the event log.

Let us consider the example illustrated in Fig. 2. In the upper part, the input of the model repair technique is depicted, that is, a process model (left) and an event log (right). The process model can be expressed in any modeling language. In this example, circles represent events, labeled rectangles represent activities, diamonds represent choices and arrows represent ordering relations. The activities in the process model are labeled with capital letters denoting the type of activity. In the event log, traces contain sequences of instances of the activities denoted with small letters. The lower part depicts the result of the model repair technique. The green components represent newly introduced parts. The repair technique starts by replaying the event log in the initial model and checking conformance of the traces. The input event log has four traces. Traces 1 and 2 are *conformant* (i.e., they can be replayed by the initial process model) while traces 3 and 4 are *non conformant* (i.e., they cannot be replayed by the initial process model). Trace 3 has a final event f that is not expected according to the model. As a possible repair, activity F is included in the model as an alternative flow after activity E. For trace 4, events j and k are not instances of any activities of the model. As a repair, the sequence J followed by K is included in the model as an alternative flow after activity A. With these repairs both not previously replayed traces are then replayed by the final process model, improving the fitness [16] of the model.

Process model repair can be positioned in between process discovery [3] and conformance checking [5] (i.e., taking a predefined model as the norm and checking whether the event log complies with it). The final model may reflect reality (i.e., observed behavior recorded in the event log) better than the initial model, but may also be very different from it, which can make the final model useless in practice. For instance, practitioners may heavily rely on the initial model to understand how a particular process functions. Presenting to them a model very different from the one they are accustomed to may result in the final model being ignored by them. To address this issue, a minimality criterion is considered when repairing the initial model guaranteeing that the final model is as similar as possible to the initial one. However, the minimality criterion does not guarantee that still important parts (e.g., commonly agreed pieces of the process that represent a shared understanding of the work) are not changed by the technique. In this paper, I argue that the repair would be more

useful to the practitioners if it takes into account predefined fragments of the model that they do not want to modify.

Furthermore, as stated in [17,18], existing approaches for process model repair are applied to the whole event log. They apply the changes based on all the traces that did not comply with the model, thus including traces that the practitioners do not want to take into account for the repair. As a consequence, the final models is unnecessarily complex and harder to understand by the practitioners. In this paper, I also argue that the repair technique can benefit from a pre-processing step, in which the relevant traces for the repair are identified.

2.2 Theory revision

A theory revision technique receives as input an initial logical theory (T_i) and a set of factual data (C). The theory is composed by a set of logic rules and can be either specified manually by domain analysts or automatically learned using an Inductive Logic Programming (ILP) system [9]. Furthermore, the initial theory is divided into two parts: an unchangeable part, which is assumed to be correct, and a changeable part that can be modified by the revision. The data are split into positive (C_p) and negative (C_n) observations. The final theory (T_f) should logically imply all the positive observations (completeness) ($\forall c_p \in C_p, T_f \vDash c_p$), none of the negative observations (consistency)($\forall c_n \in C_n, T_f \nvDash c_n$) and satisfy a criterion of minimality [7]. Figure 3 illustrates the theory revision schema.

In more detail, theory revision needs two inputs: an initial theory and a dataset. The initial theory is an approximately correct (i.e., only a few parts of the theory is preventing it from correctly explaining the dataset) set of logical rules, and it is divided into two parts: a changeable set of rules and an unchangeable one. The approximately correct requirement guarantees that revising the theory is more beneficial than relearning the theory from scratch, given that only a few parts of the theory are preventing it from correctly reflecting the dataset. Rules may be expressed in first-order logic notation (e.g., Horn clauses). These rule sets may have been specified manually by a practitioner or may have been learned via a machine learning technique. A dataset is a collection of facts

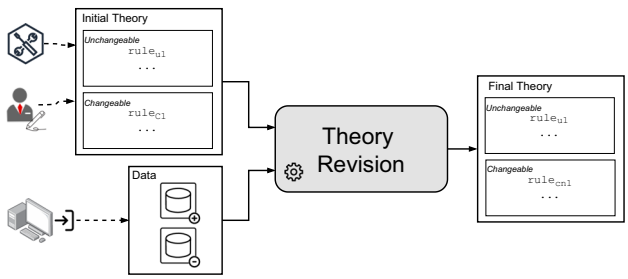

Fig. 3 Theory revision schema

that may come from an information system (e.g., a database). The data present in the dataset is divided into two sets: the positive and the negative sets. The positive set represents facts that must be explained by the theory whereas the negative set represent facts that must not be explained. The task of theory revision is to generate a final theory in which all the unchangeable rules of the initial theory are still present and some changeable rules have been replaced by new ones.

When applying theory revision, three considerations must be made. First, it must be clear where the theory should be modified (*revision points*). Second, it must be clear how the theory should be revised (*revision operators*). Third, it must be clear what *evaluation function* is going to be considered in order to choose the best revision.

Revision points are defined through the data. Positive observations define *generalization revision points* while negative observations define *specialization revision points*. The first are the literals in a rule responsible for the failure of proving a positive example (failure point) and other antecedents (contributing points) that may have contributed to this failure. The second are defined by clauses used in successful proofs of negative examples. The specification of a revision point determines the type of revision operator that will be applied to make the theory consistent with the data. Generalization operators are used when a positive observation is not proved by the theory, i.e., the theory must be more generic in order to explain a positive observation. The second group is applied when a negative observation is proved by the theory, i.e., the theory should be more specific in order to not explain a negative observation.

Theory revision relies on operators that propose modifications at each revision point. Any operator used in machine learning of first-order logic can be used in a theory revision system. For instance, the specialization operator *delete-rule*, that deletes the rule that is causing the proof of a negative observation and the operator *add-antecedent*, that adds antecedents to a rule in an attempt to make negative observations unprovable. As examples of generalization operators, we can consider the *delete-antecedent* operator that deletes antecedents from a rule making this rule more generic and therefore allowing the proof of positive observations previously not proved by the theory. Another operator is the *add-rule* operator. This operator leaves the original rule in the theory and generates new ones based on the original rule in two steps. First it copies the original rule and, using hill-climbing antecedent deletion, deletes antecedents without allowing any negative observation to be proven, and also those that allow one or more previously unprovable positive observations to be proven (even if doing so allows proofs of negatives). Then it creates one or more specializations of this core rule using the add-antecedents operator, to allow proofs of the desired positives while eliminating the negatives. An

evaluation function such as accuracy is used to select the best proposed revision to be implemented.

3 Process model repair as a theory revision task

This section discusses how to frame the task of process model repair as a theory revision task. After introducing the overarching method shown in Fig. 4, it delves into the details of its constituting components.

As input, a process model repair technique receives an initial model (M_i) and an event log (L) and outputs a revised (M_f). The initial model approximately fits the event log, i.e., its evaluation is sufficiently high (e.g., its fitness is above a threshold) which justifies the repair instead of discovering a new model from scratch. The initial model is divided into two parts. The first one (M_u) represents the part of the process model that should be *unchangeable* during the repair procedure. The specification of this part is done by the practitioners, based on their knowledge about the domain. For instance, it can represent some external or internal regulations that should be kept. Once the protected part of the process model is defined, all the rest is associated to the *changeable* part (M_{c_i}). For the repair to happen, it is necessary that M_{c_i} is defined. The M_u can be empty meaning that the practitioners chose to allow repair to be considered in the whole structure of the process model. The event log is also divided in two parts: positive (L_p) and negative (L_n) event logs. L_p corresponds to the traces representing acceptable behavior while L_n corresponds to behavior that should be avoided. The model repair technique implements changes in the model (M_{c_i}) guided by the event logs generating a final model (M_f) that includes the unchangeable part (M_u) plus the repaired model (M_{c_f}).

The changes are made in a batch mode, i.e., all the traces are received at once and the changes to the model are made considering all of them. There are some approaches for model repair that work in an incremental manner. In [19], an approach for incrementally learning declarative process models was proposed. The constrains are represented in a fragment of first-order logic consisting of Datalog Horn clauses. The approach implements changes in the model based on one trace. It can learn from scratch as well as implementing modifications to an existing model. It does not require the definition of positive and negative observations. It implements the modifications only based on positive observations. Incremental approaches are not in the scope of the present work.

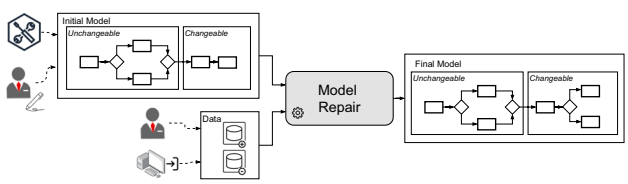

Fig. 4 Process model repair framed as a theory revision problem schema

3.1 Fragmenting the process model

The main motivation for repairing an existing model instead of re-learning it from scratch is to try to keep the repaired model as similar as possible to the original one. The reason for that is that learning a model is not a trivial task. Very often automatic discovery techniques learn complex models and do post-processing with the goal of simplifying them [20]. The final model must be useful to the practitioners. The manual specification of the model is also not an easy task, being very time demanding and complex, sometimes requiring many interactions between the process stakeholders until convergence to a final model. The point is that the current model, although not up-to-date can have many fragments that still cope with the business and/or reflect an understanding that the process stakeholders want to keep.

This paper argues that a model repair procedure should be flexible to allow the specification of fragments of the model as unchangeable, while the rest are tagged as changeable. Repair proposals will only be applied to the changeable fragments of the model. The idea of fragmenting a process model has been investigated when learning a process model from scratch [21]. Techniques for decomposing a process model (e.g., [22]) can be used for defining fragments of the model that can be changed and the fragments that should remain unchanged. The distinction between the fragments that can be changed from the ones that cannot is a manual task performed by the practitioners.

3.2 Partitioning the event log

Information Systems such as enterprise resource planning (ERP) store their execution information into event logs, which are a powerful source of observed behavior. Event logs are stored using the IEEE XES (Extensible Event Stream)[1] format. Its quality should be checked before running any process mining technique (or process model repair technique), given that the quality of the final model is closely associated to the quality of the event log. Techniques such as the one presented in [23] may be used to assess the quality of the event log. Usually, process model repair approaches consider these observed traces as acceptable behavior of the process and guide the modifications in the model with the aim of complying with these traces. However, some of these trace

[1] https://xes-standard.org/

may represent undesirable behavior, e.g., unsuccessful process execution or violation to an external regulation, and they should be separated from the desirable ones. Furthermore, some behavior that was prevented from happening, e.g., with constraints implemented in the information system, is not stored in these logs, given that they were never observed. Such behavior is still relevant to be known during the repair procedure to guarantee that a repair proposal will avoid it and therefore support practitioners on preventing them from happening in the future. I argue that the model repair task should consider positive and negative observations of the process, i.e., it should rely on a set of desirable and undesirable process instances.

The definition of the set of positive and negative traces can be made manually, semi-automatically or automatically. The starting point in all the cases is an event log. In a manual generation the practitioner goes over the event log and distinguishes desirable and undesirable behavior defining positive and negative event logs, respectively. The inclusion of additional behavior is also possible, specially in the negative event log to represent behaviors that were never observed and one wants to guarantee that the final model will not replay them in the future.

For defining the event log automatically, some strategies can be considered. For instance, one can assume that the whole event log represents desirable behaviors and has high quality, defining the positive event log. For the generation of the negative event log some existing techniques may be applied. In [24,25], the authors propose a tool to simulate declare rules and generate an event log that satisfies those rules. This technique can be used for generating the negative traces, i.e., defining unacceptable behavior as declare rules and simulating these rules. In [26], the event log was enriched with negative events. A negative event was considered by the authors as an event never observed in the event log, meaning an event prevented from happening. The traces generated with negative events may compose the negative event log.

In most scenarios, a manual definition of the positive and negative event log may be impractical. Conversely, automatic approaches may be able to find positive and negative event log, but they may lack quality which can compromise the quality of the final model. In this direction, semi-automatic approaches are suitable for the task. For instance, in [27], the authors propose a combination of different filters that partition the event log in two sets, the first one is composed by the traces that satisfy the filters and the second is composed by the remaining traces. A process model is learned from each of the sets. This approach could be used for generating the positive and the negative set of traces, i.e., the filtered traces would represent the positive observations while the unfiltered traces would represent the negative observations. The practitioner defines the filters accordingly to her understanding of the domain guiding the split into positive and negative event logs. In [28], Key Performance Indicators (KPIs) are used to filter out traces that follow some KPIs and these traces are used for the repair. This approach can be used to generate the positive and negative event logs, e.g., traces that align with the KPI are included in the positive event log and traces that do not are included in the negative event log.

As in any model learning approach, the quality of the repaired model is directly related to the quality of the input data, i.e., the positive and the negative event logs. Therefore, existing approaches for evaluating the quality of the event log, e.g., checking for missing events (i.e., events that occurred in the IS but were not recorded) or log errors (i.e., events that did not occur, but were stored in the event log) and improving it when necessary can be used as a pre-processing step to guarantee that the provided positive and negative event log have the necessary quality for the repair task.

3.3 Model repair

In this section, I sketch the main steps one should consider when developing a process model repair technique that incorporates practitioner knowledge or when extending an existing technique. Algorithm 1 depicts the top level algorithm for a process model repair framed as theory revision problem. The algorithm receives as input an initial process model (M_i) with the specification of the fragments that should remain unchanged (M_u) and the fragments that can be changed (M_{c_i}) during the repair, an event log (L) partitioned in positive event log (L_p) and negative event log (L_n) and an evaluation function (e.g., fitness). It returns as output a final model (M_f) composed by the unchanged fragments (M_u) and revised fragments (M_{c_f}).

The algorithm starts by generating the revision points which are parts in M_{c_i} where deviations were found. Conformance checking [5] techniques can be used for this task. When incorporating practitioner knowledge, the process model repair technique designers must keep track of the two possible types of revision points, generalization or specialization revision points. The former are sequences of activities that did not replay part of a positive trace and the latter are the sequences of activities that allowed a negative trace to be replayed.

Then, for each revision point, proposals of repair are made. At this point, the designers also must consider two types of revision operators, generalization or specialization revision operators, that would be applied correspondingly to the types of revision points. Different revision operators may be applied, e.g., for including an activity or a sub-process or removing infrequent activities. Each of these proposal repairs are then evaluated considering an evaluation criterion, e.g., fitness [16]. Also, a minimality criterion should be considered and proposed changes that affect the model minimally should have preference over changes that entail a bigger mod-

Algorithm 1 Process Model Repair

Require: Initial Model $M_i = M_u \cup M_{c_i}$, Event Log $L = L_p \cup L_n$, Evaluation function ef
Ensure: Revised Model $M_f = M_u \cup M_{c_f}$
1: **repeat**
2: generate revision points in M_{c_i} based on L
3: **for all** revision points **do**
4: generate revisions
5: evaluate revisions based on ef
6: implement the best revision found
7: **end for**
8: **until** the model cannot be improved

ification, e.g., given that the evaluation function returns the same value, the repair that includes less activities should be preferable. The repair that improved the most the model is the one implemented. The repair finishes when the model cannot be improved anymore, i.e., the value of the evaluation measure can not be improved anymore.

The core of the algorithm is the input in which the practitioners are able to contribute by defining unchangeable fragments (M_u) and providing acceptable (L_p) and unacceptable (L_n) behaviors. The identification of the *revision points* and the definition of the *revision operators* are a decision for the process model repair approaches designers. And existing process model repair approaches can potentially be extended by receiving the new inputs and by proposing changes on their repair technique to consider them.

4 Scenario of use

In this section, I illustrate how a process model repair technique can be extended to consider domain knowledge, i.e., how a process model repair technique can be framed as a theory revision problem in what concerns the fragmentation of the process model and the partitioning of the event log. Section 4.1 describes the setting of the scenario of use, and Sect. 4.2 describes the results obtained with the repair.

4.1 Setting

To illustrate how to frame the process model repair problem as a theory revision problem, I chose the process model repair technique proposed in [29], given that it is the available technique more closely related to the concepts proposed in this paper. Section 5 describes in detail the analysis of the available techniques.

For the event log, I used the *Process Discovery Contest 2021* data set[2]. The dataset contains 480 training logs, 96 corresponding test logs, 96 corresponding ground truth logs, and 96 process models. The dataset was generated from a base model, which was inspired by a model discovered from a real-life event log. The logs are all stored using the IEEE XES file format, while the models are workflow nets (a subclass of Petri nets) stored in a PNML file. I randomly chose an event log from the set of test logs and its corresponding process model. The event log (L) contains 250 cases with 7097 events. The process model (M_i) has 50 transitions and 44 places and it is depicted in Fig. 5, where circles represent places, squares transitions and black squares silent transitions (transitions that are not stored in event logs). I use fitness as the evaluation function (ef). The fitness of the event log with respect to the initial process model is 0.76. Considering a threshold of 0.70 for the fitness, it justifies to repair the initial model instead of discovering a new model from scratch.

As regards the inputs of the model repair procedure, I manually chose the fragment of the model that should kept unchanged, and I used filtering based on an activity to automatically split the event log. For that, I used the *Filter log by attributes* plugin from the UMA package in the Prom toolkit[3]. For model repair, I used the approach proposed in [29] through its plugin "Repair Model" implemented in the Prom toolkit. The *revision points* are identified by using conformance checking and the *revision operators* focus on including sub-processes or loops to the current model. The approach was run with its default parameters, except for the parameter concerning the removal of infrequent nodes which was set to not remove infrequent nodes.

4.2 Results

As a benchmark, the repair approach was applied to the initial process model (M_i) without setting unchangeable fragments, and to the event log (L) without partitioning it. The fitness of the repaired model (M_f) was 1. M_f is shown in Fig. 6.

Analyzing the initial fragment of M_i and its repair M_f (see Fig. 9 (a) and (b) respectively), it is possible to observe that transition *t09* was not always executed. In a first scenario (Scenario 1), a practitioner considers that this optional behavior is an unacceptable behavior (e.g., it represents a regulation), and therefore she wants to indicate that the sequence flow transition *t09* followed by transition *t10* must always happen. Then, this fragment of M_i was manually set to unchangeable (M_u). Given that the approach presented in [29] does not consider the concept of changeable and unchangeable fragments, in order to simulate it, I renamed the transitions to unknown labels. With that, it is expected that the repair approach would not change the behavior described by this fragment, given that it was not observed in the event log, and it would not remove it, given that the input parameter was set to keep all nodes including infrequent

[2] https://icpmconference.org/2021/process-discovery-contest/

[3] https://www.promtools.org/doku.php

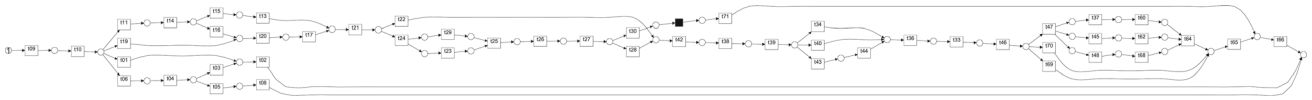

Fig. 5 Initial model (M_i)

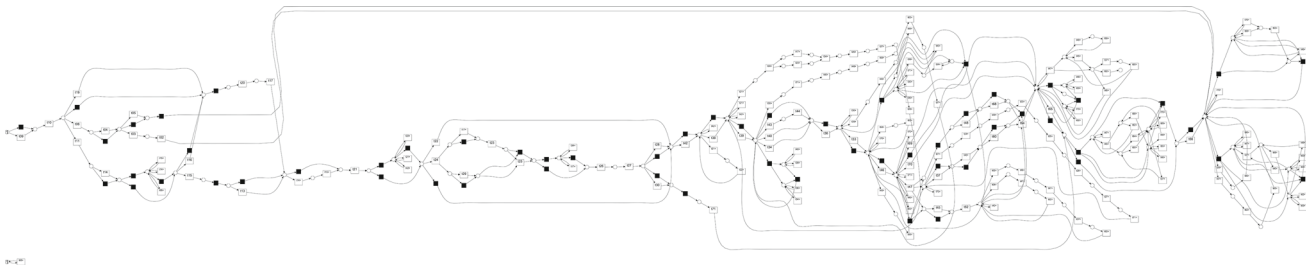

Fig. 6 Benchmark: repaired model (M_f) learned using the event log (L) without partitioning it, and the initial model (M_i) without setting unchangeable fragments

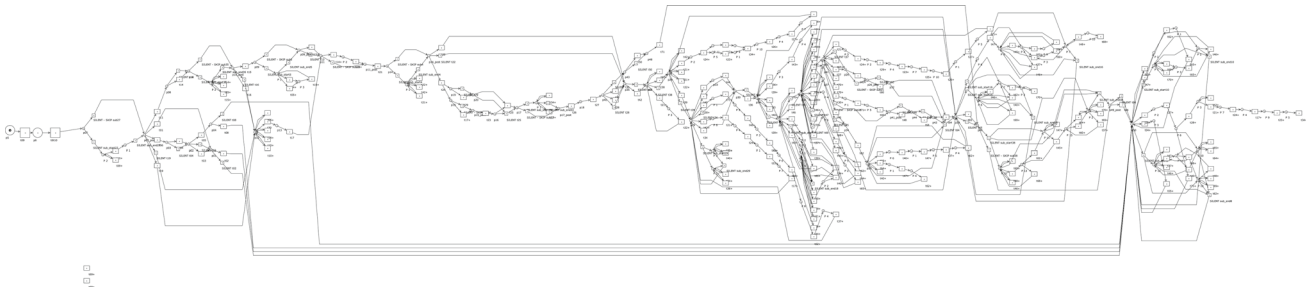

Fig. 7 Scenario 1: repaired model ($M_f = M_u \cup M_{c_f}$) with the initial fragment of M_i set as an unchangeable fragment

ones. All the rest of fragments of M_i are changeable (M_{c_i}) for the repair approach. The repair approach runs receiving as input $M_i = M_u \cup M_{c_i}$ and L. The repaired model ($M_f = M_u \cup M_{c_f}$) can be seen in Fig. 7 and its initial fragment in Fig. 9 (c). The fitness of the repaired model (M_f) was 1. It is possible to see that the initial fragment is kept, but the traces containing the undesired behavior, i.e., execution of transition *t10* without the execution of *t09* before, were considered by the repair approach and changes in the model were done in order to fit these traces. In Scenario 2, the practitioner wants to enforce that these are undesirable behavior that should not be replayed by the final process model. The event log is partitioned with traces that contain *t09* representing desirable behavior (L_p) and the traces without transition *t09* representing undesirable behavior (L_n). Event log L_p contains 178 traces while L_n contains 72. The repair approach runs receiving as input $M_i = M_u \cup M_{c_i}$ and $L = L_p \cup L_n$. The repaired model ($M_f = M_u \cup M_{c_f}$) is depicted in Fig. 8 and its initial fragment is shown in Fig. 9 (a) having the same initial fragment as the initial model. The fitness of the repaired model (M_f) was 1.

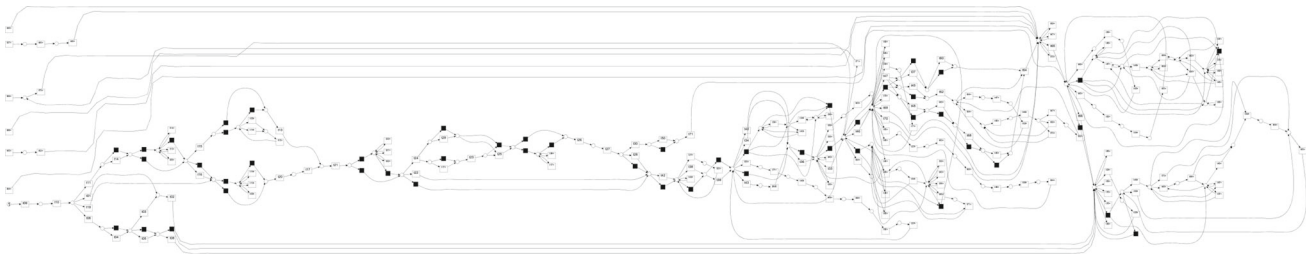

Fig. 8 Scenario 2: repaired model ($M_f = M_u \cup M_{c_f}$) with the initial fragment of M_i set as an unchangeable fragment and the event log partitioned based on having transition *t09* (L_p) and not having transition *t09* (L_n)

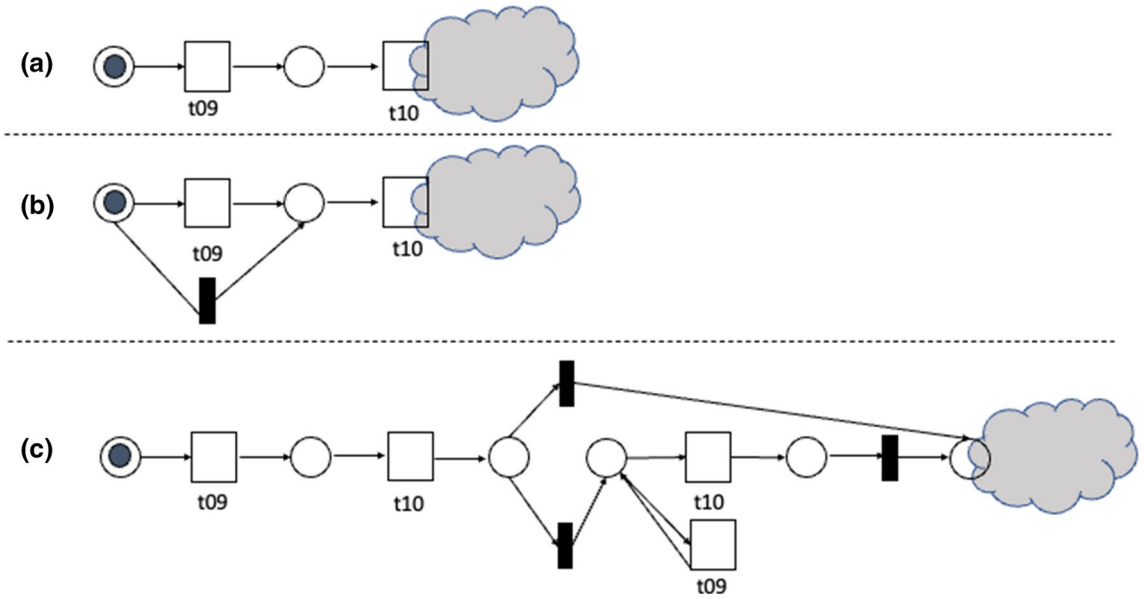

Fig. 9 (**a**) Initial fragment of the initial process model; (**b**) initial fragment of the repaired model learned using the event log without partitioning it, and the initial model without setting unchangeable fragments; (**c**) initial fragment of the repaired model learned using the sequence flow $t09 \rightarrow t10$ as unchangeable and the event log without partitioning it

With this illustrative scenario of use, it was possible to show that the final model depends on the choices made for the inputs and the practitioner plays an important role in setting these inputs appropriately. Being the domain expert, the practitioner is able to identify the parts of the model that should not be changed and also the relevant data that should be considered for the changes. Although the fitness of the repaired models found for the Benchmark, Scenario 1 and Scenario 2 are the same, their structure is different representing different behaviors. The ones in Scenario 1 and Scenario 2 reflect practitioners needs. It is worth mentioning that none of the existing approaches for process model repair provide a tool for directly implementing the ideas discussed in this paper. In the following section, I analyze the existing approaches in more detail.

5 Analysis of literature on existing process model repair techniques

This section reports the results of the analysis of the existing approaches for process model repair with respect to the proposal of framing the problem as a theory revision problem, i.e., how close to the framing proposal are the current process model repair approaches.

The selection and analysis of the existing process model repair techniques was done accordingly to systematic mapping studies [30]. To identify the existing approaches, I performed keyword search on scientific databases. I searched for scientific papers with the keywords *"process" AND "model repair"* in title, abstract or keywords using the Scopus[4] and DBLP[5] databases. A total of 151 papers were collected, 118 from Scopus and 48 from DBLP. The first excluding criteria were duplication and papers not related to the process area and in a language different than English. Then, the remaining 52 papers were read to exclude works that were not based on data or procedural process model repair. Also, only fully automatic process model repair approaches were considered. A total of 24 papers were then selected for analysis.

All the approaches rely on using conformance checking techniques [5] for identifying traces not conforming with the model and to guide the necessary repair in the model. Therefore, the techniques identify *revision points*. They vary mainly according to the conformance checking technique used (e.g., token-replay based, footprint-based or alignments based). For repairing the revision points the techniques vary according to the proposed repair method (e.g., adding a concurrent branch or a sub-process). Thus, the techniques use *revision operators* for proposing changes to the process model. This paper then focused the analysis of the existing approaches for process model repair on (i) specification of fragments of the initial model defining changeable and unchangeable fragments; (ii) partitioning of the event log into positive and negative traces; (iii) application of the min-

[4] http://scopus.com

[5] https://dblp.uni-trier.de/

Table 1 Summary of the model repair approaches

Approach	Model fragmentation	Event log partitioning	Minimality criterion
[31]			•
[32]			•
[33–45]			•
[46–48]	•		•
[49]			•
[50]			•
[15,29]		•	•
[51]			•
[52]			•

imality criterion, i.e., the guarantee of the repair procedure that the initial model was minimally changed.

This paper considered approaches where the revision procedure is fully automatized. Table 1 summarizes the findings.

In [31], the alignment of a trace and a process model is used to propose changes in the model. The approach uses a metric to calculate the alignment of the proposed repaired model with the initial one, aiming for repairs that provide minimal repair, thus a minimality criterion is used.

In [32], an impact-driven process model repair was proposed. The model repair problem was addressed as an optimization problem where each possible repair had a cost and the task was to find the repaired model that maximize fitness constrained by the cost. A maximum degree of change was considered, in this way the minimality criterion is met.

The approaches in [33–45] use the concept of Logic Petri net to repair a process model represented as a Petri net. They differ mainly by the conformance checking technique used (e.g., alignment-based [34], token-replay-based [35] and footprint-based [36]), requirements for the input Petri net (e.g., concurrency structure [37] or non-free choice structures [43]), and proposal repair (e.g., building choice structure [36] or loop structure [44]). In all the approaches, the authors argue that by only changing the structure in the Petri net associated with the deviations the initial model is minimally changed.

In [46,47], the principle of divide and conquer was used to decompose the initial process model in several fragments. Then, each fragment is classified as a good or bad fragment, depending on whether they conform to the event log or not, respectively. Repair operations are applied to the bad fragments and the generated repair fragments are then composed with the good ones generating a final repaired model. The changes to the fragments are made locally. In [48], the approach is extended to also consider non-local changes to the process model by allowing changes that connect fragments. These works align partially with (i) decomposing the model in changeable and unchangeable fragments. However, the choice is based on the conformance to the event log and not as a prior decision based on the understanding of the business and the needs of the practitioners as stated in this paper. Moreover, it can be the case that a part of the model that conforms with the data should not. By focusing on changing only the parts that did not conform with the event log keeping as much as possible of the initial model, the minimality criterion is considered.

In [49], the process model (workflow net) and the event log are both represented as footprint matrix. The repair approach implements modification in the model based on differences found between the two footprint matrices. The approach searches for a minimal change in the model and considers all the traces of the event log as desirable traces.

In [50], the authors presented the task of generalized conformance checking. A level of quality trust is associated to the log and to the model and this quality is used to repair both the log and the model. Although the authors acknowledge the possibility of the event log not representing all the possible behaviors or representing undesirable behavior, the model repair does not consider these issues in a different way, also because they are not distinguished in the event log. Therefore the consideration of positive and negative event logs is not taken into account by this approach. The alignment between the original model and the final model is calculated, but the trust value associated to the model defines the amount of change accepted. If the trust on the model is high (i.e., the model is approximately correct), then the model will be changed minimally, similarly to theory revision therefore the approach follows a minimality criterion.

The approach proposed in [15,29] is the most related to the concepts presented in this paper. It uses conformance checking to find the sequence of the model most similar to the traces that did not conform. Then the parts that did not conform are separated and process discovery technique are used to learn the correspondent sub-processes that are then included in the initial model repairing it. They separated the traces that should not be replayed by the model in a different log. Therefore, they partially considered negative and positive traces. The approach satisfies a minimality criterion.

In [51], an approach for repairing a process model by including missing edges is proposed. The event log is used to identify flow sequences that are observed in the event log, but not in the process model. The approach focuses on only including the missing edges, changing the model as minimal as possible, therefore following a minimality criterion.

In [52], the problem of process model repair was built as a multi-objective problem with the goal of changing the initial process model by maximizing the coverage of new behavior observed in the event log and minimizing the cost of the changes. The approach aims for reducing the complexity of the repaired model by not focusing only on increasing the

fitness of the model in face of the new behavior observed in the event log. The search for changes that minimize the cost in terms of simplicity of the model and only focusing on changing the model where it is necessary follows a minimality criterion.

As a result of the analysis of the existing techniques for process model repair it is possible to observe that the techniques use a minimality criterion to guarantee that the final model resembles as much as possible the initial model. However, the techniques can be improved by the involvement of the practitioners for fragmentation of the initial model and partitioning of the event log. The approaches proposed in [46–48] and [15,29] partially fulfilled these two points, respectively.

6 Related work

The contribution of this paper has been on process model repair in which the model follows the procedural paradigm, and the repair is done automatically. In Sect. 5, it was shown that the use of practitioners knowledge to support process model repair under these requirements has been modest. In this section, I describe related work, i.e., work that consider similar ideas for process model repair. I classify this prior research to two streams: (i) non data-aware process model repair that consider undesirable behavior and/or unchangeable fragments of the model; and (ii) data-aware declarative process model repair that consider undesirable behavior and/or unchangeable fragments of the model.

For what concerns stream (i), [53] proposes to identify incorrect free-choice segments in a process model and repair it based on a transition system that represent the same segment as a non free-choice segment. The approach keeps the fitness of the model with the event log used to discover the process model and improves precision, given that prevents the model from modeling undesirable behavior previously represented in the free-choice segment. This work implements the concept of undesirable behaviors and the necessity of the process model to avoid covering these behaviors, however these behaviors are not explicit in the event log and therefore the repair is not data oriented.

For what concerns stream (ii), given the analyzed sources of the literature, it can be observed that the concept of theory revision has not yet been considered for process model repair when the model follows a procedural paradigm. When the process model is represented with a declarative paradigm, work such as [54] can be mentioned. Theory revision concepts were used to improve DECLARE [55] rules, where a set of positive and negative event logs were built and used for the revision of the set of DECLARE rules. In the context of process discovery, the approach presented in [56] used a partitioning of the event log into positive and negative event logs to learn declarative process model.

7 Conclusion

Process models support practitioners when it comes to managing various workflows around software development such as issue-to-resolution processes, bug-fixing, version release-planning, etc. If these models become outdated over time, the monitoring of the process becomes inaccurate. Process model repair proposes to change the model in order to cope with the latest changes in the data recorded in event log. From the area of ILP, theory revision techniques allow the revision of a logical theory, i.e., a set of rules, guided by positive and negative observations. The logical theory is minimally changed in order to explain all the positive observations and none of the negative observations.

This paper explored the area of process model repair framing it as a theory revision problem. It identified two main points from theory revision that can contribute to process model repair: (i) the fragmentation of the initial process model in a way that the practitioners may indicate fragments that should not be considered for repair and (ii) the definition of two event logs, namely positive and negative logs. The first event log includes behavior that should be replayed by the model and the second one includes behavior that should be avoided by the model. To use theory revision for process model repair, two main challenges should be addressed, namely the fragmentation of the initial process model and the partitioning of the event log.

As future work, I intend to (i) formalize other aspects of theory revision and (ii) explore other facets of the broader area of *theory refinement* [7]. For what concerns (i), as existing works are already implicitly using the concepts of revision points and revision operators, I plan to formally define them. For what concerns (ii), the automatic improvement of logic knowledge bases, known as *theory refinement*, can be divided into two classes: *theory revision* and *theory restructuring*. Both aim at improving the quality of the logical theory. The revision task involves changing the answer set of the given theory, i.e., improving its inferential capabilities by adding previously missing answers (generalization) or by removing incorrect answers (specialization). On the other hand, the task of restructuring does not change the answer set of the given theory; its objective is to improve performance and/or user understandability of the theory. As a follow-up work, I intend to investigate how concepts of theory restructuring can support the task of process model repair, e.g., process simplification [20].

Furthermore, this paper was based on the assumption that the input is a event log in XES format. However, the source can be a database and approaches such as [57] which extract

event logs with specific characteristics following the goal of business analysts can also be used. The extraction could be made following requirements of desirable and undesirable events generating in this way the negative and positive event logs.

Lastly, I intend to develop of a prototype to be used to collect feedback from the practitioners concerning ease of use, usability, and trustworthiness when incorporating their knowledge into process model repair.

Acknowledgements This work was partially supported by WU Wien via the WU-Project Projekt-IA 27001663 and by Teaming.AI project in the European Union's Horizon 2020 research and innovation program under Grant Agreement No 95740.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, 2nd edn. Springer, New York (2018)
2. Schmidt, D.C.: Model-driven engineering. Comput.-IEEE Comput. Soc. **39**(2), 25 (2006)
3. van der Aalst, W.M.P.: Process Mining - Data Science in Action, 2nd edn. Springer, New York (2016)
4. Armas-Cervantes, A.: Process model repair. In: Encyclopedia of Big Data Technologies. Springer, New York (2019)
5. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer, New York (2018)
6. Barriga, A., Heldal, R., Iovino, L., Marthinsen, M., Rutle, A.: An extensible framework for customizable model repair, pp. 24–34. In: MoDELS. ACM (2020)
7. Wrobel, S.: First order theory refinement. In: De Raedt, L. (ed.) Advances in Inductive Logic Programming. IOS Press (1996)
8. Taylor, C., Nakhaeizadeh, G.: Learning in dynamically changing domains: theory revision and context dependence issues. In: ECML, vol. 1224, pp. 353–360. Springer, New York (1997)
9. Muggleton, S.: Inductive logic programming. New Gener. Comput. **8**(4), 295–318 (1991). https://doi.org/10.1007/BF03037089
10. Sato, D.M.V., Freitas, S.C.D., Barddal, J.P., Scalabrin, E.E.: A survey on concept drift in process mining. ACM Comput. Surv. **54**(9), 1–38 (2021)
11. Muggleton, S., Paes, A., Costa, V.S., Zaverucha, G.: Chess revision: acquiring the rules of chess variants through FOL theory revision from examples. In: ILP, vol. 5989, pp. 123–130. Springer, New York (2009)
12. Guimarães, V., Paes, A., Zaverucha, G.: Online probabilistic theory revision from examples with ProPPR. Mach Learn. **108**(7), 1165–1189 (2019)
13. Raedt, L.D., Kersting, K., Kimmig, A., Revoredo, K., Toivonen, H.: Compressing probabilistic Prolog programs. Mach Learn. **70**(2–3), 151–168 (2008)
14. Revoredo, K.: Process model repair meets theory revision - initial ideas. In: PoEM, vol. 432, pp. 184–194. Springer, New York (2021)
15. Fahland, D., van der Aalst, W.M.P.: Repairing process models to reflect reality. In: BPM, vol. 7481, pp. 229–245. Springer, New York (2012)
16. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: On the role of fitness, precision, generalization and simplicity in process discovery. In: OTM Conferences (1), vol. 7565, pp. 305–322. Springer, New York (2012)
17. Armas-Cervantes, A., van Beest, N.R.T.P., Rosa, M.L., Dumas, M., Raboczi, S.: Incremental and interactive business process model repair in apromore. In: BPM (Demos), vol. 1920. CEUR-WS.org (2017)
18. Armas-Cervantes, A., van Beest, N.R.T.P., Rosa, M.L., Dumas, M., García-Bañuelos, L.: Interactive and incremental business process model repair. In: OTM Conferences (1), vol. 10573, pp. 53–74. Springer, New York (2017)
19. Ferilli, S.: Incremental declarative process mining with woman. In: EAIS, pp. 1–8. IEEE (2020)
20. Fahland, D., van der Aalst, W.M.P.: Simplifying discovered process models in a controlled manner. Inf Syst. **38**(4), 585–605 (2013)
21. Schuster, D., van Zelst, S.J., van der Aalst, W.M.P.: Freezing submodels during incremental process discovery. In: ER, vol. 13011, pp. 14–24. Springer, New York (2021)
22. van der Aalst, W.M.P.: Decomposing Petri nets for process mining: a generic approach. Distrib. Parallel Databases **31**(4), 471–507 (2013)
23. Martin, N., Van Houdt, G., Janssenswillen, G.: DaQAPO: supporting flexible and fine-grained event log quality assessment. Expert Syst. Appl. **191**, 116274 (2022). https://doi.org/10.1016/j.eswa.2021.116274
24. Ackermann, L., Schönig, S.: MuDePS: Multi-perspective declarative process simulation. In: BPM (Demos), vol. 1789, pp. 12–16. CEUR-WS.org (2016)
25. Ackermann, L., Schönig, S., Jablonski, S.: Simulation of multi-perspective declarative process models. In: Business Process Management Workshops. vol. 281, pp. 61–73 (2016)
26. Goedertier, S., Martens, D., Baesens, B., Haesen, R., Vanthienen, J.: Process mining as first-order classification learning on logs with negative events. In: Business Process Management Workshops, vol. 4928, pp. 42–53. Springer, New York (2007)
27. Vidgof, M., Djurica, D., Bala, S., Mendling, J.: Cherry-picking from spaghetti: multi-range filtering of event logs. In: BPMDS/EMMSAD@CAiSE, vol. 387, pp. 135–149. Springer, New York (2020)
28. Dees, M., de Leoni, M., Mannhardt, F.: Enhancing process models to improve business performance: a methodology and case studies. In: OTM Conferences (1), vol. 10573, pp. 232–251. Springer, New York (2017)
29. Fahland, D., van der Aalst, W.M.P.: Model repair - aligning process models to reality. Inf. Syst. **47**, 220–243 (2015)
30. Kitchenham, B., Charters, S.: Guidelines for Performing Systematic Literature Reviews in Software Engineering (2007)
31. Buijs, J.C.A.M., La Rosa, M., Reijers, H.A., van Dongen, B.F., van der Aalst, W.M.P.: Improving business process models using observed behavior. In: Cudre-Mauroux, P., Ceravolo, P., Gašević, D. (eds.) Data-Driven Process Discovery and Analysis, pp. 44–59. Springer, Berlin Heidelberg (2013)

32. Polyvyanyy, A., van der Aalst, W.M.P.: ter Hofstede AHM, Wynn MT impact-driven process model repair. ACM Trans. Softw. Eng. Methodol. **25**(4), 28:1–28:60 (2017)
33. Xu, Y., Du, Y., Qi, L., Luan, W., Wang, L.: A logic petri net-based model repair approach by constructing choice bridges. IEEE Access **7**, 18531–18545 (2019). https://doi.org/10.1109/ACCESS.2019.2896079
34. Zhang, X., Du, Y., Qi, L., Sun, H.: An approach for repairing process models based on logic petri nets. IEEE Access **6**, 29926–29939 (2018)
35. Teng, Y., Du, Y., Qi, L., Luan, W., Wang, L.: A simple logic transition repair method for business process models via logic petri nets. IEEE Access **7**, 76628–76644 (2019)
36. Xu, Y., Du, Y., Luan, W., Qi, L.: A process model repair approach by constructing choice structures via logic petri nets. IEEE Access **7**, 172387–172402 (2019)
37. Zheng, W., Du, Y., Qi, L., Wang, L.: A method for repairing process models containing a choice with concurrency structure by using logic petri nets. IEEE Access **7**, 13106–13120 (2019)
38. Teng, Y., Du, Y., Qi, L.: A logic petri net-based repair method of process models with incomplete choice and concurrent structures. Comput. Inf. **39**(1), 264–297 (2020)
39. Teng, Y., Du, Y., Qi, L., Luan, W.: A logic petri net-based method for repairing process models with concurrent blocks. IEEE Access **7**, 8266–8282 (2019)
40. Xu, Y., Du, Y., Luan, W., Qi, L., Sun, H.: Repairing process models with logical concurrent and casual relations via logical petri nets. IEEE Access **6**, 56340–56355 (2018)
41. Zhang, X., Du, Y., Qi, L., Sun, H.: Repairing process models containing choice structures via logic petri nets. IEEE Access **6**, 53796–53810 (2018)
42. Bai, E., Su, N., Liang, Y., Qi, L., Du, Y.: Method for repairing process models with selection structures based on token replay. Comput. Inf. **40**(2), 446–468 (2021)
43. Zheng, W., Du, Y., Wang, S., Qi, L.: Repair process models containing non-free-choice structures based on logic petri nets. IEEE Access **7**, 105132–105145 (2019)
44. He, Z., Du, Y., Qi, L., Du, H.: A model repair approach based on petri nets by constructing free-loop structures. IEEE Access **7**, 24214–24230 (2019)
45. Qi, H., Du, Y., Qi, L., Wang, L.: An approach to repair Petri net-based process models with choice structures. Enterp. Inf Syst. **12**(8–9), 1149–1179 (2018)
46. Mitsyuk, A.A., Lomazova, I.A., Shugurov, I.S., van der, Aalst W.M.P.: Process model repair by detecting unfitting fragments. In: AIST (Supplement), vol. 1975, pp. 301–313. CEUR-WS.org (2017)
47. Mitsyuk, A.A., Lomazova, I.A., van der Aalst, W.M.P.: Using event logs for local correction of process models. Autom. Control Comput. Sci. **51**(7), 709–723 (2017)
48. Mitsyuk, A.A.: Non-local correction of process models using event logs. In: 2017 Ivannikov ISPRAS open conference (ISPRAS), pp. 6–11 (2017)
49. Sun, Y., Du, Y., Li, M.: A repair of workflow models based on mirroring matrices. Int. J. Parallel Program. **45**(4), 1001–1020 (2017)
50. Rogge-Solti, A., Senderovich, A., Weidlich, M., Mendling, J., Gal, A.: In log and model we trust? A generalized conformance checking framework. In: BPM, vol. 9850, pp. 179–196. Springer, New York (2016)
51. Fernández-Ropero, M., Reijers, HA., Pérez-Castillo, R., Piattini, M.: Repairing business process models as retrieved from source code. In: BMMDS/EMMSAD, vol. 147, pp. 94–108. Springer, New York (2013)
52. Francescomarino, C.D., Tiella, R., Ghidini, C., Tonella, P.: A multi-objective approach to business process repair. In: ICSOC, vol. 8831, pp. 32–46. Springer, New York (2014)
53. Kalenkova, A.A., Carmona, J., Polyvyanyy, A., Rosa, M.L.: Automated repair of process models using non-local constraints. In: Petri Nets, vol. 12152, pp. 280–300. Springer, New York (2020)
54. Cattafi, M., Lamma, E., Riguzzi, F., Storari, S.: Incremental declarative process mining. In: Smart Information and Knowledge Management, vol. 260, pp. 103–127. Springer, New York (2010)
55. Pesic, M., Schonenberg, H., van der, Aalst W.M.P.: DECLARE: full support for loosely-structured processes. In: EDOC, pp. 287–300. IEEE Computer Society (2007)
56. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting inductive logic programming techniques for declarative process mining. Trans. Petri Nets Other Model Concurr. **2**, 278–295 (2009)
57. de Murillas, E.G.L., Reijers, H.A., van der Aalst, W.M.P.: Connecting databases with process mining: a meta model and toolset. Softw. Syst. Model. **18**(2), 1209–1247 (2019)

Kate Revoredo is a research associate at the Chair of Process Management and Information Systems at the Department of Computer Science at Humboldt-Universität zu Berlin, Germany, and previously an assistant professor at the Institute for Information Business at Wirtschaftsuniversit Wien (WU Vienna), Austria. Her main research interest is on data-centric approaches for information systems. This includes topics in the area of business process management, data science, semantic web and knowledge representation. She has published more than 80 research papers in various outlets, among others in Computers in Industry, Information Systems and Machine Learning. She is a member of the Brazilian Commission in Artificial Intelligence and a reviewer for several journals and conferences, among others Business & Information Systems Engineering, The Knowledge Engineering Review, International Joint Conference on Artificial Intelligence (IJCAI) and Conference on Artificial Intelligence (AAAI).