Change Mining in Adaptive Process Management Systems

Christian W. Günther[1], Stefanie Rinderle[2], Manfred Reichert[3], Wil van der Aalst[1]

[1] Eindhoven University of Technology, The Netherlands
{c.w.gunther, w.m.p.v.d.aalst}@tm.tue.nl
[2] University of Ulm, Germany
stefanie.rinderle@uni-ulm.de
[3] University of Twente, The Netherlands
m.u.reichert@ewi.utwente.nl

Abstract. The wide-spread adoption of process-aware information systems has resulted in a bulk of computerized information about real-world processes. This data can be utilized for process performance analysis as well as for process improvement. In this context process mining offers promising perspectives. So far, existing mining techniques have been applied to operational processes, i.e., knowledge is extracted from execution logs (process discovery), or execution logs are compared with some a-priori process model (conformance checking). However, execution logs only constitute one kind of data gathered during process enactment. In particular, adaptive processes provide additional information about process changes (e.g., ad-hoc changes of single process instances) which can be used to enable organizational learning. In this paper we present an approach for mining change logs in adaptive process management systems. The change process discovered through process mining provides an aggregated overview of all changes that happened so far. This, in turn, can serve as basis for all kinds of process improvement actions, e.g., it may trigger process redesign or better control mechanisms.

1 Introduction

The striking divergence between modeled processes and practice is largely due to the rigid, inflexible nature of commonplace *Process-Aware Information Systems* (PAISs) [10]. Whenever a small detail is modeled in the wrong manner, or external changes are imposed on the process (e.g. a new legislation or company guideline), users are forced to *deviate* from the prescribed process model. However, given the fact that process (re-)design is an expensive and time-consuming task, this results in employees working "behind the system's back". In the end, the PAIS starts to become a burden rather than the help it was intended to be. In recent years many efforts have been undertaken to deal with these drawbacks and to make PAISs more flexible. In particular, several approaches for *adaptive* process management have emerged (for an overview see [17]). Adaptive processes enable users to *evolve* process definitions, such that they fit to changed situations. Adaptability can be supported by dynamic changes of different process aspects (e.g., control and data flow) at different levels (e.g., instance and type level). For example, ad-hoc changes conducted at the instance level (e.g., to add or delete process steps) allow to flexibly adapt single process instances to exceptional or changing situations [14]. Usually, such deviations are recorded in change logs (see [18]), which results in more meaningful log information when compared to traditional *Process Management Systems* (PMSs).

Adaptive PMSs like ADEPT or WASA offer flexibility at both process type level and process instance level [14, 17, 21]. So far, adaptive PMSs have not systematically addressed the fundamental question what we can learn from this additional information and how we can derive optimized process models from it. Process mining techniques [2], in turn, offer promising perspectives for learning from changes, but have focused on the analysis of pure execution logs (i.e., taking a behavioral and operational perspective) so far.

This paper presents a framework for integrating adaptive process management and process mining: Change information gathered within the adaptive PMS is exploited by process mining techniques. The results can be used to learn from previously applied changes and to optimize running and future processes accordingly. For this integration, first of all, we determine which runtime information about ad-hoc deviations is necessary and how it should be represented in order to achieve optimal mining results. Secondly, we develop new mining techniques based on existing ones which utilize change logs in addition to execution logs. As a result we obtain an abstract *change process* which reflects all changes applied to the instances of a particular process type so far. More precisely, a change process comprises *change operations* (as activities) and the causal relations between them. We further utilize information about the semantics of change operations (e.g., commutativity) in order to optimize our mining results. The resulting change process provides valuable knowledge about the process changes happened so far, which may serve as basis for deriving process optimizations in the sequel. Finally, the approach is implemented within a prototype integrating process mining framework ProM and ADEPT.

Sect. 2 introduces our framework for integrating process mining and adaptive process management. Sect. 3 describes how we import change log information into this framework and how changes are represented. Sect. 4 deals with our approach for discovering change processes from these logs. In Sect. 5 we discuss details of our implementation and show which tool supported is provided. Sect. 6 discusses related work and Sect. 7 concludes with a summary and an outlook.

2 Process Optimization by Integrating Process Mining and Adaptive Process Management

In this section we argue that the value of adaptive PMSs can be further leveraged by integrating them with process mining techniques. After introducing basics related to process mining, we present our overall integration framework.

2.1 Process Mining

Process-Aware Information Systems (PAISs), such as WfMS, ERP, and B2B systems, need to be configured based on *process models*. The latter specify the order in which process steps are to be executed and therefore enable the information system to ensure and control the correct execution of operational processes.

Usually, relevant events occurring in a PAIS (e.g., regarding the execution of tasks or the modification of data) are recorded in *event logs*. *Process mining* describes a family of *a-posteriori* analysis techniques exploiting the information recorded in these logs. Typically, respective approaches assume that it is possible to sequentially record events such that each event refers to an activity (i.e., a well-defined step in the process) and is related to a particular case (i.e., a process instance). Furthermore, there are other mining techniques making use of additional information such as the performer or originator of the event (i.e., the person / resource executing or initiating the activity), the timestamp of the event, or data elements recorded with the event (e.g., the size of an order).

Process mining addresses the problem that most "process owners" have very limited information about what is actually happening in their organization. In practice there is often a significant gap between what is prescribed or supposed to happen, and what *actually* happens. Only a concise assessment of the organizational reality, which process mining strives to deliver, can help in verifying process models, and ultimately be used in a process redesign effort.

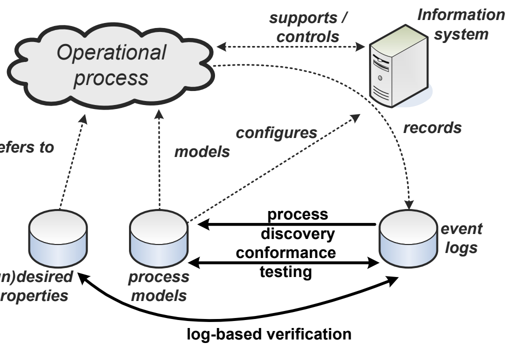


Fig. 1. Process Mining and its relation to BPM.

There are three major classes of process mining techniques as indicated in Fig. 1. Traditionally, process mining has focused on *process discovery*, i.e. deriving information about the original process model, the organizational context, and execution properties from enactment logs. An example of a technique addressing the control flow perspective is the alpha algorithm [2], which can construct a Petri net model [6] describing the behavior observed in the event log. The multi-phase mining approach [7] can be used to construct an Event-driven

Process Chain (EPC) based on similar information. Finally, first work regarding the mining of other model perspectives (e.g., organizational aspects [1]) and data-driven process support systems (e.g., case handling systems) has been done.

Another line of process mining research is *conformance testing*. Its aim is to analyze and measure discrepancies between the model of a process and its actual execution (as recorded in event logs). This can be used to indicate problems. Finally, *log-based verification* does not analyze enactment logs with respect to the original model, but rather checks the log for conformance with certain desired or undesired properties, e.g., expressed in terms of Linear Temporal Logic (LTL) formulas. This makes it an excellent tool to check a case for conformance to certain laws or corporate guidelines (e.g. the four-eyes principle).

At this point in time there are mature tools such as the ProM framework, featuring an extensive set of analysis techniques which can be applied to real process enactments while covering the whole spectrum depicted in Fig. 1 [9].

2.2 Integration Framework

Both process mining and adaptive workflow address fundamental issues that are widely prevalent in the current practice of BPM implementations. These problems stem from the fact that the *design*, *enactment*, and *analysis* of a business process are commonly interpreted, and implemented, as *detached phases*.

Although both techniques are valuable on their own, we argue that their full potential can be only harnessed by tight integration. While process mining can deliver concrete and reliable information about how process models need to be changed, adaptive PMSs provide the tools to safely and conveniently implement these changes. Thus, we propose the development of process mining techniques, integrated into adaptive PMSs as a *feedback cycle*. In the sequel, adaptive PMSs need to be equipped with functionality to exploit this feedback information.

The framework depicted in Fig. 2 illustrates how such an integration could look like. Adaptive PMSs, visualized in the upper part of this model, operate on pre-defined process models. The evolution of these models over time spawns a set of process changes, i.e., results in multiple *process variants*. Like in every PAIS, *enactment logs* are created which record the sequence of activities executed for each case. On top of that, adaptive PMSs additionally log the sequence of change operations imposed on a process model for every executed case, producing a set of *change logs*. Process mining techniques that integrate into such system, in form of a feedback cycle, fall into one of three major categories:

Change analysis: Process mining techniques from this category make use of change log information, besides the original process models and their variants. Their goal is to determine common and popular variants for each process model, which may be promoted to replace the original model. Possible ways to pursue this goal are through statistical analysis of changes or their abstraction to higher-level models.

Integrated analysis: This analysis uses both change and enactment logs in a combined fashion. Possible applications in this category could perform a

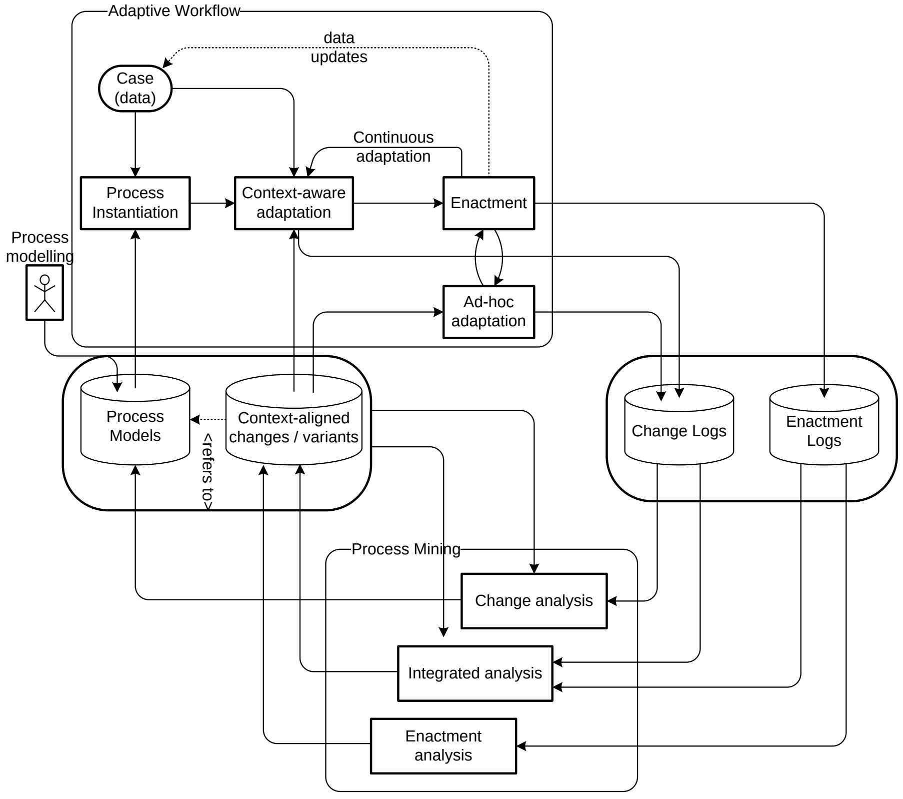


Fig. 2. Integration of Process Mining and Adaptive Process Management

context-aware categorization of changes as follows. After clustering change sequences, as found in the change logs, into groups, the incentive for these changes can be derived. This is performed by inspecting the state of each case, i.e. the values of case data objects, at the time of change, as known from the original process model and the enactment logs. A decision-tree analysis of these change clusters provides an excellent basis for guiding users in future process adaptations, based on the peculiarities of their specific case.

Enactment analysis: Based solely on the inspection of enactment logs, techniques in this category can pinpoint parts of a process model which need to be changed. For example, when a specific alternative of a process model has never been executed, the original process model may be simplified by removing that part. Further techniques may also clean the model repository from rarely used process definitions.

These examples give only directions in which the development of suitable process mining techniques may proceed. Of course, their concrete realization depends on the nature of the system at hand. For example, it may be preferable to present highlighted process models to a specialist before their deletion or change, rather than having the system performing these tasks autonomously.

When such feedback cycle is designed and implemented consistently, the resulting system is able to provide user guidance and autonomous administration to an unprecedented degree. Moreover, the tight integration of adaptive PMSs and process mining technologies provides a powerful foundation, on which a new generation of truly intelligent and increasingly autonomous PAISs can be built.

3 Change Logs

Adaptive PMSs do not only create process enactment logs, but *they also log the sequence of changes applied to a process model.* This section introduces the basics of these change logs. We first discuss the nature of changes and then introduce MXML as general format for event logs. Based on this we show how change logs can be mapped onto the MXML format. MXML-based log files constitute the basic input for the mining approach described in Sect. 4.

3.1 General Change Framework

Logically, a process change is accomplished by applying a sequence of change operations to the respective process model [14]. The question is how to represent this change information within change logs. In principle, the information to be logged can be represented in different ways. The goal must be to find an adequate representation and appropriate analysis techniques to support the three cases described in the previous section.

Independent from the applied (high–level) change operations (e.g., adding, deleting or moving activities), for example, we could translate the change into a set of basic change primitives (i.e., basic graph primitives like `addNode` or `deleteEdge`). This still would enable us to restore process structures, but also result in a loss of information about change semantics and therefore limit traceability and change analysis. As an alternative we can explicitly store the applied high–level change operations, which combine basic primitives in a certain way.

High–level change operations are based on formal pre-/post-conditions. This enables the PMS to guarantee model correctness when changes are applied. Further, high-level change operations can be combined to *change transactions*. This becomes necessary, for example, if the application of a high-level change operation leads to an incorrect process model and this can be overcome by conducting concomitant changes. During runtime several change transactions may be applied to a particular process instance. All change transactions related to a process instance are stored in the *change log*[4] of this instance (cf. [18]).

[4] A change log is an ordered series $cL{:=}< \Delta_1, \ldots, \Delta_n >$ of change operations Δ_i (i = 1, ..n); i.e., when applying the change operations contained in cL to a correct process schema S, all intermediate process schemas S_i with $S_i := S_{i-1} + \Delta_i$ (i = 1,. . . , n; $S_0 := S$) are correct process schemas.

In the following we represent change log entries by means of high-level change operations since we want to exploit their semantical content (see Fig. 3 for an example). However, basically, the mining approach introduced later can be adapted to change primitives as well. Table 1 presents examples of *high-level change operations* on process models which can be used at the process type as well as at the process instance level to create or modify models. Although the change operations are exemplarily defined on the ADEPT meta model (see [14] for details) they are generic in the sense that they can be easily transferred to other meta models as well (e.g. [15]).

Table 1. *Examples of High-Level Change Operations on Process Schemas*

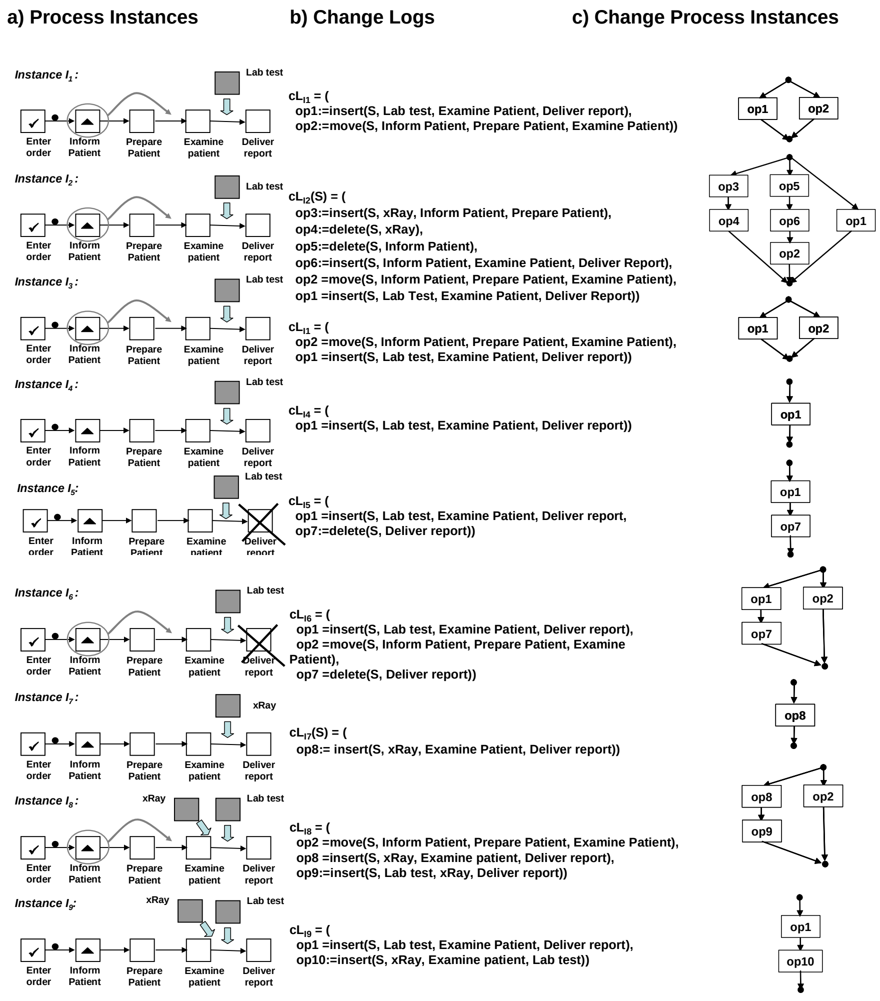
Change Operation Δ **Applied to S**	opType	subject	paramList
insert(S, X, A, B, [*sc*])	insert	X	S, A, B
Effects on S: inserts activity X between node sets A and B (it is a conditional insert if *sc* is specified)			
Preconditions: node sets A and B must exist in S, and X must not be contained in S yet (i.e., no duplicate activities!)			
delete(S, X)	delete	X	S
Effects on S: deletes activity X from S			
Preconditions: activity X must be contained exactly once in S			
move(S, X, A, B, [*sc*])	move	X	S, A, B
Effects on S: moves activity X from its original position between node sets A and B (it is a conditional insert if *sc* is specified)			
Preconditions: activity X and node sets A and B must be contained exactly once in S			

3.2 The MXML Format for Process Event Logs

MXML is an XML-based format for representing and storing event log data, which is supported by the largest subset of process mining tools, such as ProM. While focusing on the core information needed for process mining, the format reserves generic fields for extra information potentially provided by a PAIS. Due to its outstanding tool support and extensibility, the MXML format has been selected for storing change log information in our approach.

The root node of a MXML document is a *WorkflowLog*. It represents a log file, i.e. a logical collection of events having been derived from one system. Every workflow log can potentially contain one *Source* element, which is used to describe that system the log has been imported from. Apart from the source descriptor, a workflow log can contain an arbitrary number of *Processes* as child elements, each grouping events that occurred during the execution of a specific process definition. The single executions of a process are represented by child elements of type *ProcessInstance*, each representing one case in the system.

Finally, process instances group an arbitrary number of *AuditTrailEntry* elements as child elements. Each of these child elements refers to one specific event which has occurred in the system. Every audit trail entry must contain at least two child elements: The *WorkflowModelElement* describes the abstract process definition element to which the event refers, e.g. the name of the activity that



```
cL_I1 = (
  op1:=insert(S, Lab test, Examine Patient, Deliver report),
  op2:=move(S, Inform Patient, Prepare Patient, Examine Patient))

cL_I2(S) = (
  op3:=insert(S, xRay, Inform Patient, Prepare Patient),
  op4:=delete(S, xRay),
  op5:=delete(S, Inform Patient),
  op6:=insert(S, Inform Patient, Examine Patient, Deliver Report),
  op2 =move(S, Inform Patient, Prepare Patient, Examine Patient),
  op1 =insert(S, Lab Test, Examine Patient, Deliver Report))

cL_I1 = (
  op2 =move(S, Inform Patient, Prepare Patient, Examine Patient),
  op1 =insert(S, Lab test, Examine Patient, Deliver report))

cL_I4 = (
  op1 =insert(S, Lab test, Examine Patient, Deliver report))

cL_I5 = (
  op1 =insert(S, Lab test, Examine Patient, Deliver report,
  op7:=delete(S, Deliver report))

cL_I6 = (
  op1 =insert(S, Lab test, Examine Patient, Deliver report),
  op2 =move(S, Inform Patient, Prepare Patient, Examine
Patient),
  op7 =delete(S, Deliver report))

cL_I7(S) = (
  op8:= insert(S, xRay, Examine Patient, Deliver report))

cL_I8 = (
  op2 =move(S, Inform Patient, Prepare Patient, Examine Patient),
  op8 =insert(S, xRay, Examine patient, Deliver report),
  op9:=insert(S, Lab test, xRay, Deliver report))

cL_I9 = (
  op1 =insert(S, Lab test, Examine Patient, Deliver report),
  op10:=insert(S, xRay, Examine patient, Lab test))
```

Fig. 3. Modified Process Instances and Associated Change Logs

was executed. The second mandatory element is the *EventType*, describing the nature of the event, e.g. whether a task was scheduled, completed, etc. The optional child elements of an audit trail entry are *Timestamp* and *Originator*. The timestamp holds the date and time of when the event has occurred, while the originator identifies the resource, e.g. person, which has triggered the event.

To enable the flexible extension of this format with extra information, all mentioned elements (except the child elements of *AuditTrailEntry*) can also have a generic *Data* child element. The data element groups an arbitrary number of *Attributes*, which are key-value pairs of strings. The following subsection describes the mapping of change log information to MXML, which is heavily based on using custom attributes of this sort.

3.3 Mapping Change Log Information to MXML

With respect to an adaptive PAIS, change log information can be structured on a number of different levels. Most of all, change events can be grouped by the process definition they address. As we are focusing on changes applied to *cases*, i.e. executed instances of a process definition, the change events referring to one process can be further subdivided with respect to the specific case in which they were applied. Finally, groups of change events on a case level are naturally sorted by the order of their occurrence.

The described structure of change logs fits well into the common organization of enactment logs, with instance traces then corresponding to *consecutive changes* of a process model, in contrast to its execution. Thus, change logs can be mapped to the MXML format with minor modifications. Listing 1 shows an MXML audit trail entry describing the insertion of a task “Lab Test” into a process definition, as e.g. seen for Instance I_1 in Fig. 3.

```
<AuditTrailEntry>
    <Data>
        <Attribute name="CHANGE.postset">Deliver_report</Attribute>
        <Attribute name="CHANGE.type">INSERT</Attribute>
        <Attribute name="CHANGE.subject">Lab_test</Attribute>
        <Attribute name="CHANGE.rationale">Ensure that blood values
             are within specs.</Attribute>
        <Attribute name="CHANGE.preset">Examine_patient</Attribute>
    </Data>
    <WorkflowModelElement>INSERT.Lab_test</WorkflowModelElement>
    <EventType>complete</EventType>
    <Originator>N.E.Body</Originator>
</AuditTrailEntry>
```

Listing 1: Example of a change event in MXML.

Change operations are characterized by the *type* (e.g., “INSERT”) of change, the *subject* which has been primarily affected (e.g., the inserted task), and the *syntactical context* of the change. This syntactical context contains the change operation’s *pre-* and *post-set*, referring to adjacent model elements that are either directly preceding or following the change subject in the process definition. These specific change operation properties are not covered by the MXML format, therefore they are stored as attributes in the “Data” field. The set of attributes for a change event is further extended by an optional *rationale* field, storing a human-readable reason, or incentive, for this particular change operation.

The *originator* field is used for the person having applied the respective change, while the *timestamp* field obviously describes the concise date and time of occurrence. Change events have the *event type* “complete” by default, because they can be interpreted as atomic operations. In order to retain backward compatibility of MXML change logs with traditional process mining algorithms, the *workflow model element* needs to be specified for each change event. As the change process does not follow a prescribed process model, this information is not available. Thus, a concatenation of change type and subject is used for the workflow model element field.

On top of having a different set of information, change logs also exhibit specific properties making them different from enactment logs. The next section investigates these specific properties and uses them for a first mining approach.

4 Mining Compact Change Processes

In this section we describe an approach for analyzing change log information, as found in adaptive PMSs. First, we explore the nature of change logs in more detail. This is followed by an introduction to the concept of commutativity between change operations in Sect. 4.2. This commutativity relation provides the foundation for our mining algorithm for change processes, as introduced in Sect. 4.3.

4.1 A Characterization of Change Logs

Change logs, in contrast to regular enactment logs, do not describe the execution of a defined process. This is obvious from the fact that, if the set of potential changes would have been known in advance, then these changes could have already been incorporated in the process model (making dynamic change obsolete). Thus, change logs must be interpreted as emerging sequences of activities which are taken from a set of change operations.

In Sect. 3.3 it has been defined that each change operation refers to the original process model through three associations: the *subject*, *pre-*, and *post-set* of the change. As all three associations can theoretically be bound to any subset from the original process model's set of activities[5], the set of possible change operations grows exponentially with the number of activities in the original process model. This situation is fairly different from mining a regular process model, where the number of activities is usually rather limited (e.g., up to 50–100 activities). Hence, change process mining poses an interesting challenge.

Summarizing the above characteristics, we can describe the *meta-process* of changing a process schema as a *highly unstructured* process, potentially involving a *large number of distinct activities.* These properties, when faced by a process mining algorithm, typically lead to overly precise and confusing *"spaghetti-like" models.* For a more compact representation of change processes, it is helpful to abstract from a certain subset of order relations between change operations.

When performing process mining on enactment logs (i.e., the classical application domain of process mining), the actual state of the mined process is treated like a "black box". This is a result of the nature of enactment logs, which typically only indicate transitions in the process, i.e. the execution of activities.

[5] Here we assume that the subset describing the *subject* field is limited to one.

However, the information contained in change logs allows to trace the state of the change process, which is indeed defined by the process schema that is subject to change. Moreover, one can compare the effects of different (sequences of) change operations. From that, it becomes possible to explicitly detect whether two consecutive change operations might as well have been executed in the reverse order, without changing the resulting state.

The next section introduces the concept of *commutativity* between change operations, which is used to reduce the number of ordering relations by taking into account the semantic implications of change events. Since the order of commutative change operations does not matter, we can abstract from the actually observed sequences thus simplifying the resulting model.

4.2 Commutative and Dependent Change Operations

Change operations modify a process schema, either by altering the set of activities or by changing their ordering relations. Thus, each application of a change operation to a process schema results in another, different schema. A process schema can be described formally without selecting a particular notation, i.e., we abstract from the concrete operators of the process modeling language and only describe the set of activities and possible behavior.

Definition 1 (Process Schema). *A process schema is a tuple $PS = (A, TS)$ where*

- *A is a set of activities*
- *$TS = (S, T, s_{start}, s_{end})$ is a labeled transition system, where S is the set of reachable states, $T \subseteq S \times (A \cup \{\tau\}) \times S$ is the transition relation, $s_{start} \in S$ is the initial state, and $s_{end} \in S$ is the final state.*

$\mathcal{P}$ is the set of all process schemas.

The behavior of a process is described in terms of a transition system TS with some initial state s_{start} and some final state s_{end}. Note that any process modeling language can be mapped onto a labeled transition system. The transition system does not only define the set of possible traces (i.e., execution orders); it also captures the moment of choice. Moreover, it allows for "silent steps". A silent step, denoted by τ, is an activity within the system which changes the state of the process, but is not observable in the execution logs. This way we can distinguish between different types of choices (internal/external or controllable/uncontrollable) [5]. While all change operations modify the set of states S and the transition relation T, the "move" operation is the only one not changing the set of activities A.

In order to compare sequences of change operations, and to derive ordering relations between these changes, it is helpful to define an equivalence relation for process schemas.

Definition 2 (Equivalent Process Schemas). *Let $\equiv$ be some equivalence relation. For $PS_1, PS_2 \in \mathcal{P}$: $PS_1 \equiv PS_2$ if and only if PS_1 and PS_2 are considered to be equivalent.*

There exist many notions of process equivalence. The weakest notion of equivalence is *trace equivalence* [11, 13, 17], which regards two process models as equivalent if the sets of observable traces they can execute are identical. Since the number of traces a process model can generate may be infinite, such comparison may be complicated. Moreover, since trace equivalence is limited to comparing traces, it fails to correctly capture the moment at which choice occurs in a process. For example, two process schemas may generate the same set of two traces $\{ABC, ABD\}$. However, the process may be very different with respect to the moment of choice, i.e. the first process may already have a choice after A to execute either BC or BD, while the second process has a choice between C and D just after B. *Branching bisimilarity* is one example of an equivalence, which can correctly capture this moment of choice. For a comparison of branching bisimilarity and further equivalences the reader is referred to [12]. In the context of this paper, we abstract from a concrete notion of equivalence, as the approach described can be combined with different process modeling notations and different notions of equivalence.

As stated above, each application of a change operation transforms a process schema into another process schema. This can be formalized as follows:

Definition 3 (Change in Process Schemas). *Let $PS_1, PS_2 \in \mathcal{P}$ be two process schemas and let Δ be a process change.*

- *$PS_1[\Delta\rangle$ if and only if Δ is applicable to PS_1, i.e., Δ is possible in PS_1.*
- *$PS_1[\Delta\rangle PS_2$ if and only if Δ is applicable to PS_1 (i.e., $PS_1[\Delta\rangle$) and PS_2 is the process schema resulting from the application of Δ to PS_1.*

The applicability of a change operation to a specific process schema is defined in Table 1, and is largely dictated by common sense. For example, an activity X can only be inserted into a schema S, between the node sets A and B, if these node sets are indeed contained in S and the activity X is not already contained in S. Note that we do not allow duplicate tasks, i.e. an activity can be contained only once in a process schema.

Based on the notion of process equivalence we can now define the concept of *commutativity* between change operations.

Definition 4 (Commutativity of Changes). *Let $PS \in \mathcal{P}$ be a process schema, and let Δ_1 and Δ_2 be two process changes. Δ_1 and Δ_2 are commutative in PS if and only if:*

- *There exist $PS_1, PS_2 \in \mathcal{P}$ such that $PS[\Delta_1\rangle PS_1$ and $PS_1[\Delta_2\rangle PS_2$,*
- *There exist $PS_3, PS_4 \in \mathcal{P}$ such that $PS[\Delta_2\rangle PS_3$ and $PS_3[\Delta_1\rangle PS_4$,*
- *$PS_2 \equiv PS_4$.*

Two change operations are commutative, if they have exactly the same effect on a process schema, regardless of the order in which they are applied. If two change operations are not commutative, we regard them as *dependent*, i.e., the effect of the second change depends on the first one. The concept of commutativity captures the ordering relation between two consecutive change operations. If two change operations are commutative according to Def. 4 they can be applied in any given order, therefore there exists no ordering relation between them.

In the next subsection we demonstrate that existing process mining algorithms can be enhanced with the concept of commutativity, thereby abstracting from ordering relations that are irrelevant from a semantical point of view (i.e., their order does not influence the resulting process schema).

4.3 Mining Change Processes

Mining change processes is to a large degree identical to mining regular processes from enactment logs. Therefore, we have chosen not to develop an entirely new algorithm, but rather to base our approach on an existing process mining technique. Among the available algorithms, the *multi-phase* algorithm [7, 8] has been selected, which is very robust in handling fuzzy branching situations (i.e., it can employ the "OR" semantics to split and join nodes, in cases where neither "AND" nor "XOR" are suitable). Although we illustrate our approach using a particular algorithm, it is important to note that any process mining algorithm based on explicitly detecting causalities can be extended in this way (e.g., also the different variants of the α-algorithm).

The multi-phase mining algorithm is able to construct basic workflow graphs, Petri nets, and EPC models from the causality relations derived from the log. For an in-depth description of this algorithm, the reader is referred to [7, 8]. The basic idea of the multi-phase mining algorithm is to discover the process schema in two steps. First a model is generated for each individual process instance. Since there are no choices in a single instance, the model only needs to capture causal dependencies. Using causality relations derived from observed execution orders and the commutativity of specific change operations, it is relatively easy to construct such instance models. In the second step these instance models are aggregated to obtain an overall model for the entire set of change logs.

The causal relations for the multi-phase algorithm [7, 8] are derived from the change log as follows. If a change operation A is followed by another change B in at least one process instance, and no instance contains B followed by A, the algorithm assumes a possible causal relation from A to B (i.e., "A may cause B"). In the example log introduced in Fig. 3, instance I_2 features a change operation deleting "Inform Patient" followed by another change, inserting the same activity again. As no other instance contains these changes in reverse order, a causal relation is established between them.

Fig. 4 shows a Petri net model [6] of the change process mined from the example change log instances in Fig. 3. The detected causal relation between deleting and inserting "Inform patient" is shown as a directed link between these activities. Note that in order to give the change process explicit start and end points, artificial activities have been added. Although the model contains only seven activities, up to three of them can be executed concurrently. Note further that the process is very flexible, i.e. all activities can potentially be skipped. From the very small data basis given in Fig. 3, where change log instances hardly have common subsequences, this model delivers a high degree of abstraction.

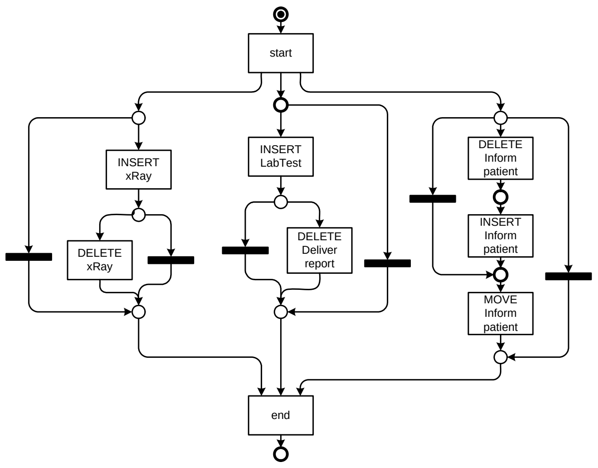


Fig. 4. Mined Example Process (Petri net notation)

If two change operations are found to appear in both orders in the log, it is assumed that they can be executed in any order, i.e. concurrently. (Note that there might be some order between concurrent changes determined by context factors not directly accessible to the system. We aim at integrating such information in our future work). An example for this is inserting “xRay” and inserting “Lab Test”, which appear in this order in instance I_8, and in reverse order in instance I_9. As a result, there is no causal relation, and thus no direct link between these change operations in the model shown in Fig. 4.

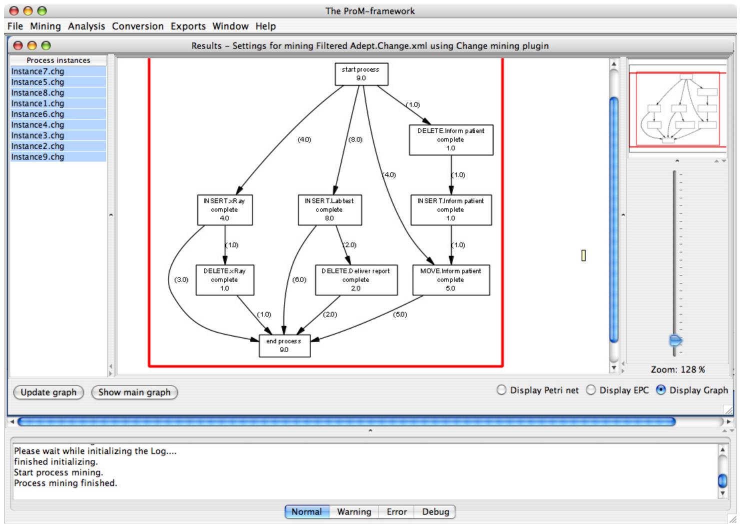
Apart from observed concurrency, as described above, we can introduce the concept of *commutativity-induced concurrency*, using the notion of commutativity introduced in the previous subsection (cf. Definition 4). From the set of observed causal relations, we can exclude causal relations between change operations that are commutative. For example, instance I_2 features deleting activity “xRay” directly followed by deleting “Inform Patient”. As no other process instance contains these change operations in reverse order, a regular process mining algorithm would establish a causal relation between them.

However, it is obvious that it makes no difference in which order two activities are removed from a process schema. As the resulting process schemas are identical, these two changes are *commutative*. Thus, we can safely discard a causal relation between deleting “xRay” and deleting “Inform Patient”, which is why there is no link in the resulting change process shown in Fig. 4.

Commutativity-induced concurrency removes unnecessary causal relations, i.e. those causal relations that do not reflect actual dependencies between change operations. Extending the multi-phase mining algorithm with this concept significantly improves the clarity and quality of the mined change process. If it were not for commutativity-induced concurrency, every two change operations would need to be observed in both orders to find them concurrent. This is especially significant in the context of change logs, since one can expect changes to a process schema to happen far less frequently than the actual execution of the schema, resulting in less log data.

5 Implementation and Tool Support

To enable experimentation with change logs and their analysis, an import plug-in has been implemented for the ProM*import* framework, which allows to extract both enactment and change logs from instance files of the ADEPT demonstrator prototype [16]. ProM*import*[6] is a flexible and open framework for the rapid prototyping of MXML import facilities from all kinds of PAISs. The ADEPT demonstrator prototype provides the full set of process change facilities found in the ADEPT distribution, except for implementation features like work distribution and the like. The combination of both makes it possible to create and modify a process model in the ADEPT demonstrator prototype, after which a respective change log can be imported and written to the MXML-based change log format described in Sect. 3.3.



```
Please wait while initializing the Log....
finished initializing.
Start process mining.
Process mining finished.
```

Fig. 5. Change Mining Plug-in within ProM.

These change logs can then be loaded into the ProM framework[7]. A dedicated change mining plug-in has been developed, which implements the commutativity-

[6] ProM*import* is available under an Open Source license at http://promimport.sourceforge.net/.

[7] ProM is available under an Open Source license at http://prom.sourceforge.net/.

enhanced multi-phase algorithm described in Sect. 4.3. It is also possible to mine only a selection of the change logs found in the log. The resulting change process can be visualized in the form of a workflow graph, Petri net, or EPC.

Figure 5 shows the change mining plug-in within the ProM framework, displaying the example process introduced in Fig. 3 in terms of a process graph. The activities and arcs are annotated with frequencies, indicating how often the respective node or path has been found in the log.

6 Related Work

Although process mining techniques have been intensively studied in recent years [2–4, 7, 8], no systematic research on analyzing process change logs has been conducted so far. Existing approaches mainly deal with the discovery of process models from execution logs, conformance testing, and log-based verification (cf. Sect. 2.1). However, execution logs in traditional PMSs only reflect what has been modeled before, but do not capture information about process changes. While earlier work on process mining has mainly focused on issues related to control flow mining, recent work additionally uses event-based data for mining model perspectives other than control flow (e.g., social networks [1], actor assignments, and decision mining [19]).

In recent years, several approaches for adaptive process management have emerged [17], most of them supporting changes of certain process aspects and changes at different levels. Examples of adaptive PMSs include ADEPT [16], CBRflow [20], and WASA [21]. Though these PMSs provide more meaningful process logs when compared to traditional workflow systems, so far, only little work has been done on fundamental questions like what we can learn from this additional log information, how we can utilize change logs, and how we can derive optimized process models from them. CBRflow has focused on the question how to facilitate exception handling in adaptive PMSs. In this context case-based reasoning (CBR) techniques have been adopted in order to capture contextual knowledge about ad-hoc changes in change logs, and to assist actors in reusing previous changes. [20]. This complementary approach results in semantically enriched log-files (e.g., containing information about the frequency of a particular change, user ratings, etc.) which can be helpful for our future work.

7 Summary and Outlook

In this paper we presented an approach for integrating adaptive process management and process mining in order to exploit knowledge about process changes from change logs. For this we have developed a mining technique and implemented it as plug-in of the ProM framework taking ADEPT change logs as input. We demonstrated that change log information (as created by adaptive PMSs like ADEPT) can be imported into the ProM framework. Based on this we have sketched how to discover a (minimal) change process which captures all modifications applied to a particular process instance so far. This discovery is based on the analysis of the (temporal) dependencies existing between the change operations applied to the respective process instance. How single change processes can be combined to one aggregated change process (capturing all instance changes applied) has been presented afterwards. Finally we have described the implementation framework behind our approach. Altogether, the presented approach can be very helpful for process engineers to get an overview about which instance changes have been applied at the system level and what we can learn from them. Corresponding knowledge is indispensable to make the right decisions with respect to the introduction of changes at the process type level (e.g., to reduce the need for ad-hoc changes at the instance level in future).

In our future work we want to further improve user support by augmenting change processes with additional contextual information (e.g., about the reason why changes have been applied or the originator of the change). From this we expect better comprehensibility of change decisions and higher reusability of change knowledge (in similar situations). The detection of this more context-based information will be accomplished by applying advanced mining techniques (e.g., decision mining [19]) to change log information.

Acknowledgements: This research has been supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Dutch Ministry of Economic Affairs.

References

1. W.M.P. van der Aalst, H.A. Reijers, and M. Song. Discovering Social Networks from Event Logs. *Computer Supported Cooperative work*, 14(6):549–593, 2005.
2. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
3. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
4. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
5. J. Dehnert and W.M.P. van der Aalst. Bridging the Gap Between Business Models and Workflow Specifications. *International Journal of Cooperative Information Systems*, 13(3):289–332, 2004.
6. J. Desel, W. Reisig, and G. Rozenberg, editors. *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2004.
7. B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Process Mining: Building Instance Graphs. In P. Atzeni, W. Chu, H. Lu, S. Zhou, and T.W. Ling, editors, *International Conference on Conceptual Modeling (ER 2004)*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, Berlin, 2004.

8. B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Process Mining: Aggregating Instance Graphs into EPCs and Petri Nets. In *Proceedings of the 2nd International Workshop on Applications of Petri Nets to Coordination, Worklflow and Business Process Management (PNCWB) at the ICATPN 2005*, 2005.
9. B.F. van Dongen, A.K. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A new era in process mining tool support. In G. Ciardo and P. Darondeau, editors, *Proceedings of the 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005)*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
10. M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software through Process Technology.* Wiley & Sons, 2005.
11. R. van Glabbeek and U. Goltz. Refinement of Actions and Equivalence Notions for Concurrent Systems. *Acta Informatica*, 37(4–5):229–327, 2001.
12. R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.
13. B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows.* PhD thesis, Queensland University of Technology, Brisbane, 2002. (available via *http://www.workflowpatterns.com/)*.
14. M. Reichert and P. Dadam. ADEPTflex - Supporting Dynamic Changes of Workflows Without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
15. M. Reichert, S. Rinderle, and P. Dadam. On the common support of workflow type and instance changes under correctness constraints. In *Proc. Int'l Conf. on Cooperative Information Systems (CoopIS'03)*, pages 407–425, Catania, 2003.
16. M. Reichert, S. Rinderle, U. Kreher, and P. Dadam. Adaptive process management with ADEPT2. In *Proc. 21st Int'l Conf. on Data Engineering (ICDE'05)*, pages 1113–1114, Tokyo, 2005.
17. S. Rinderle, M. Reichert, and P. Dadam. Correctness Criteria for Dynamic Changes in Workflow Systems – A Survey. *Data and Knowledge Engineering, Special Issue on Advances in Business Process Management*, 50(1):9–34, 2004.
18. S. Rinderle, M. Reichert, M. Jurisch, and U. Kreher. On Representing, Purging, and Utilizing Change Logs in Process Management Systems. In *Proc. Int'l Conf. on Business Process Management (BPM'06)*, Vienna, 2006.
19. A. Rozinat and W.M.P. van der Aalst. Decision mining in prom. In *Proc. Int'l Conf. on Business Process Management (BPM'06)*, Vienna, 2006.
20. B. Weber, S. Rinderle, W. Wild, and M. Reichert. CCBR-Driven Business Process Evolution. In *Proc. Int. Conf. on Cased based Reasoning (ICCBR'05)*, Chicago, 2005.
21. M. Weske. Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In *Proc. Hawaii International Conference on System Sciences (HICSS-34)*, 2001.