Using process mining to learn from process changes in evolutionary systems

Using Process Mining to Learn from Process Changes in Evolutionary Systems

Christian W. Günther[1], Stefanie Rinderle[2],
Manfred Reichert[3], Wil van der Aalst[1,4], Jan Recker[4]

[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
[4] Queensland University of Technology, Australia
j.recker@qut.edu.au

Abstract. Traditional information systems struggle with the requirement to provide flexibility and process support while still enforcing some degree of control. Accordingly, *adaptive* process management systems (PMSs) have emerged that provide some flexibility by enabling dynamic process changes during runtime. Based on the assumption that these process changes are recorded explicitly, we present two techniques for mining change logs in adaptive PMSs; i.e., we do not only analyze the execution logs of the operational processes, but also consider the adaptations made at the process instance level. The change processes discovered through process mining provide an aggregated overview of all changes that happened so far. This, in turn, can serve as basis for integrating the extrinsic drivers of process change (i.e., the stimuli for flexibility) with existing process adaptation approaches (i.e., the intrinsic change mechanisms). Using process mining as an analysis tool we show in this paper how better support can be provided for truly flexible processes by understanding *when* and *why* process changes become necessary.

1 Introduction

The notion of flexibility has emerged as a pivotal research topic in Business Process Management (BPM) over the last years [7, 8, 34, 48]. The need for more flexibility, in general, stems from the observation that organisations often face continuous and unprecedented changes in their business environment [32, 51]. Such disturbances and perturbations of business routines need to be reflected within the business processes, in the sense that these processes as well as their supporting information systems need to be quickly adaptable to environmental changes.

In this context, business process flexibility denotes the capability to reflect externally triggered change by modifying only those aspects of a process that

need to be changed, while keeping the other parts stable; i.e., the ability to change or evolve the process without completely replacing it [33]. In particular, we have to deal with the essential requirement for maintaining a close "fit" between the real-world business processes and the workflows as supported by Process Management Systems (PMSs), the current generation of which is known under the label of *Process-aware Information Systems* (PAISs) [20]. As real-world processes are continuously subjected to disruptions, deviations and changes, their supporting systems should provide sufficient flexibility and quick adaptability in order to cope with process changes [48, 8].

1.1 Problem Statement

Recently, many efforts have been undertaken to make PAISs more flexible and several approaches for *adaptive* process management have emerged (for an overview see [38]), resulting in more flexible PAISs (like ADEPT [34], CBRFlow [55] or WASA [57]). The basic idea behind these approaches is to enable users to dynamically *evolve* or *adapt* process schemas such that they fit to changed real-world situations. More precisely, adaptive PMSs support dynamic changes of different process aspects (e.g., control and data flow) at different levels (e.g., the process instance and the process type level). In particular, ad-hoc changes conducted at the process instance level (e.g., to add, delete or move process steps during runtime) allow to flexibly adapt single process instances to exceptional or changing situations [34]. Usually, such ad-hoc deviations are recorded in *change logs* (see [39]), which results in more meaningful log information when compared to traditional PAISs.[5]

So far, adaptive process management technology has not addressed the fundamental question what we can learn from the additional log information (e.g., how to derive an optimized process schema from a collection of individually adapted process instances [52]). In principle, *process mining* techniques [2] offer promising perspectives for this. However, current process mining algorithms have not been designed with adaptive processes in mind, but have focused on the analysis of pure execution logs instead (i.e., taking a behavioral and operational perspective).

Obviously, mining ad-hoc changes in adaptive PMSs offers promising perspectives as well. By enhancing adaptive processes with advanced mining techniques we aim at a PMS framework, which enables full *process life cycle support*. However, the practical implementation of such a framework in a coherent architecture, let alone the integration of process mining and adaptive processes is far from trivial. In particular, we have to deal with the following challenges: First, we have to determine which runtime information about ad-hoc deviations has to be logged and how this information should be represented in order to achieve

[5] PAIS which do not provide support for ad-hoc changes are usually bypassed in exceptional situations (e.g., by executing unplanned activities outside the scope of the PAIS). Consequently, the PAIS is unaware of the performed deviations and thus unable to log (let alone analyze) information about them.

optimal mining results. Second, we have to develop advanced mining techniques that utilize *change logs* in addition to pure execution logs. The concurrent consideration of execution logs and change logs in the context of change mining facilitates learning more about the *context* and the *reasons* of a change. Third, we have to integrate the new mining techniques with existing adaptive process management technology. This requires the provision of integrated tool support allowing us to evaluate our framework and to compare different mining variants.

1.2 Contribution

In our previous work, with ADEPT [34] and ProM [19] we have developed two separate frameworks for adaptive processes and for process mining respectively. While ADEPT has focused on the support of dynamic process changes at different levels, ProM has offered a wide variety of process mining techniques, e.g., for discovering a Petri Net model or an Event Process Chain (EPC) describing the behavior observed in an execution log. However, so far no specific ProM extension has been developed to mine for process changes.

This paper contributes new techniques for the mining of ad-hoc process changes in adaptive PMSs and discusses the challenges arising in this context. We first describe what constitutes a process change, how process change information can be represented in respective logs, and how these change logs have to be mined to deliver insights into the scope and context of changes. This enables us, for example, to better understand how users deviate from predefined processes. We import change logs from adaptive PMSs like ADEPT and introduce two mining approaches for discovering change knowledge from these logs. As result, we obtain an abstract *change process* represented as a Petri Net model. This abstract process reflects all changes applied to the instances of a particular process type so far. More precisely, a change process comprises *change operations* (as process steps) and the causal relations between them. We introduce two different mining approaches which are based on different assumptions and techniques. The first approach uses multi–phase mining, but utilizes further information about the semantics of change operations (i.e., commutativity) in order to optimize results. The second approach maps change logs to labeled state transition systems, and then constructs a compact Petri Net model from it.

We additionally incorporate in this paper information about *process context* in order to better understand why process changes occur and what causes them (i.e., the semantic reasons for a change). Simply put, context is the relevant subset of the entire situation of a business process that requires it to adapt to potential changes. We introduce a basic notion of context and show how context information can be analyzed using machine learning techniques. Altogether, the developed mining techniques provide valuable knowledge about the process changes happened so far, which serves as basis for deriving process optimizations.

The remainder of this paper is organized as follows: Section 2 provides background information on process mining and adaptive process management. In Section 3 we present a general framework for integrating these two technologies. Section 4 deals with the representation of process changes and corresponding

change logs. Based on these considerations, Section 5 introduces a format for representing change logs, and discusses to which degree existing mining techniques can be applied to these logs and which limitations exist in this context. In Section 6 we introduce two new techniques for mining change logs along a running example and give a comparison of them. Section 7 discusses how learning mechanisms can be devised based on a comprehensive understanding of the context of business processes. In Section 8 we present details of our proof-of-concept implementation and show which tool support is provided. Section 9 discusses related work and Section 10 concludes with a summary and an outlook.

2 Background Information

The ideas and approaches presented in this paper are based on the application and integration of existing technologies, namely *process mining* analysis techniques and *adaptive process management*. In order to understand the implications and leverages of their combination, it is helpful to gain a basic understanding. Section 2.1 introduces process mining, followed by an introduction to adaptive process management in Section 2.2.

2.1 Process Mining

Although the focus of this paper is on analyzing change processes in the context of adaptive process management systems, *process mining* is applicable to a much wider range of information systems. There are different kinds of Process-Aware Information Systems (PAISs) that produce *event logs* recording events. Examples are classical workflow management systems (e.g. Staffware), ERP systems (e.g. SAP), case handling systems (e.g. FLOWer), PDM systems (e.g. Windchill), CRM systems (e.g. Microsoft Dynamics CRM), middleware (e.g. IBM's WebSphere), hospital information systems (e.g. Chipsoft), etc. These systems all provide very detailed information about the activities that have been executed. The goal of process mining is to extract information (e.g., process models, or schemas) from these logs, i.e., process mining describes a family of *a-posteriori* analysis techniques exploiting the information recorded in event 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 using 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 predefined 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 schemas, and ultimately be used in a process redesign effort.

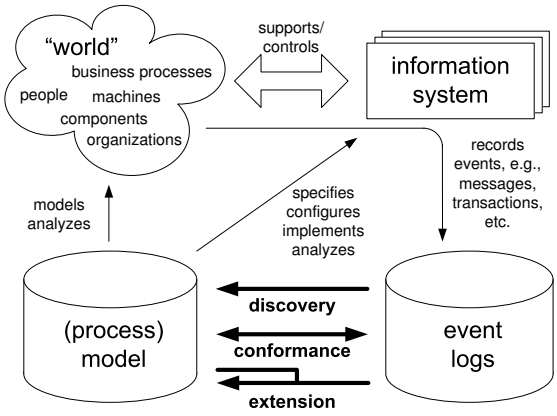

Fig. 1. Overview showing three types of process mining: (1) Discovery, (2) Conformance, and (3) Extension.

The idea of process mining is to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs. Clearly process mining is relevant in a setting where much flexibility is allowed and/or needed and therefore this is an important topic in this paper. The more ways in which people and organizations can deviate, the more variability and the more interesting it is to observe and analyze processes as they are executed. We consider three basic types of process mining (cf. Figure 1):

- **Discovery**: There is no a-priori process schema, i.e., based on an event log some schema is constructed. For example, using the alpha algorithm a process schema can be discovered based on low-level events.
- **Conformance**: There is an a-priori process schema. This schema is used to check if reality conforms to the schema. For example, there may be a process schema indicating that purchase orders of more than one million Euro require two checks. Another example is the checking of the four-eyes principle. Conformance checking may be used to detect deviations, to locate and explain these deviations, and to measure the severity of these deviations.
- **Extension**: There is an a-priori process schema. This schema is extended with a new aspect or perspective, i.e., the goal is not to check conformance but to enrich the schema. An example is the extension of a process schema with performance data, i.e., some a-priori process schema is used to project the bottlenecks on.

Traditionally, process mining has been focusing on *discovery*, i.e., deriving information about the original process schema, 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 [15] describing the behavior observed in the event log. The

multi-phase mining approach [17] can be used to construct an Event-driven Process Chain (EPC) based on similar information. However, process mining is not limited to process schemas (i.e., control flow) and recent process mining techniques are more and more focusing on other perspectives, e.g., the organizational perspective or the data perspective. For example, there are approaches to extract social networks from event logs and analyze them using social network analysis [1]. This allows organizations to monitor how people and groups are working together.

Conformance checking compares an a-priori schema with the observed behavior as recorded in the log. In [43] it is shown how a process schema (e.g., a Petri net) can be evaluated in the context of a log using metrics such as "fitness" (Is the observed behavior possible according to the schema?) and "appropriateness" (Is the schema "typical" for the observed behavior?). However, it is also possible to check conformance based on organizational models, predefined business rules, temporal formula's, Quality of Service (QoS) definitions, etc.

There are different ways to extend a given process schema with additional perspectives based on event logs. For this paper, decision mining [44] is the most interesting technique. Decision mining, also referred to as decision point analysis, aims at the detection of data dependencies that affect the routing of a case. Starting from a process schema, one can analyze how data attributes influence the choices made in the process based on past process executions. Classical data mining techniques such as decision trees can be leveraged for this purpose. The machine learning technique for inducing a decision tree from data is called decision tree learning. A leaf node in the decision tree corresponds to taking specific path, i.e, the equivalent of a decision in a process schema. The splits in the decision tree are based on data values in the log, e.g., the values appearing in all events of a case before the decision point is reached. Similarly, the process schema can be extended with timing information (e.g., bottleneck analysis).

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 Figure 1 [19]. Any of the analysis techniques of ProM can be applied to change logs (i.e., event logs in the context of adaptive process management systems). Moreover, this paper also presents two new process mining techniques exploiting the particularities of change logs.

2.2 Adaptive Process Management

A key requirement for BPM technology becoming more and more important in practice is (runtime) adaptivity; i.e., the ability of the PMS to support (dynamic) changes at the process type as well as the process instance level. Several approaches have been discussed in literature (e.g. [57, 34, 9, 22, 38]), and a number of prototypes demonstrating the high potential of adaptive PMS have emerged [36, 57]. With the introduction of such adaptive PMSs we obtain additional runtime information about process executions not explicitly captured in

current execution logs. This information can be useful in different context and is usually managed in respective change logs. Change log entries may contain information about the type of a change, the applied change operations and their parameterizations, the time the change happened, etc.

Basically, process changes can take place at the type as well as the instance level: Changes of single process instances may have to be carried out in an ad-hoc manner (e.g., to deal with an exceptional situation) and must not affect system robustness and consistency. Process type changes, in turn, must be quickly accomplished in order to adapt the PAIS to business process changes. The latter could be triggered, for example, by analyzing the changes which were individually applied to a collection of process instances. In the context of long-running business processes, process type changes may additionally require the migration of thousands of instances to the new process schema. Important requirements are to perform respective migrations on-the-fly, to preserve correctness, and to avoid performance penalties.

PMS frameworks like ADEPT [34, 36] support both ad-hoc changes of single process instances and the propagation of process type changes to running instances. Examples of *ad-hoc deviations* from the prescribed process schema include the insertion, deletion, movement, or replacement of activities. In ADEPT, such ad-hoc changes do not lead to an unstable system behavior, i.e., none of the system guarantees achieved by formal checks at buildtime are violated due to the dynamic change. ADEPT offers a complete set of operations for defining instance changes at a high semantic level and ensures correctness by introducing pre-/post-conditions for these operations. Finally, all complexity associated with the adaptation of instance states, the remapping of activity parameters, or the problem of missing data (e.g., due to activity deletions) is hidden from users. In order to deal with business process changes ADEPT also enables quick and efficient schema adaptations at the process type level. In particular, it is possible to propagate type changes to running instances (of this type). In this context, comprehensive correctness criteria are provided for deciding on the compliance of process instances with a modified process type schema. ADEPT enables efficient compliance checks and allows to quickly and correctly adapt the states of instance when migrating them to the new schema.

3 A Framework for Integrating Process Mining and Adaptive Process Management

Both process mining and adaptive process management 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 only be harnessed by tight integration. While process mining can deliver concrete and reliable information about how process schemas 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*. On the other side, adaptive PMSs need to be equipped with functionality to exploit this feedback information.

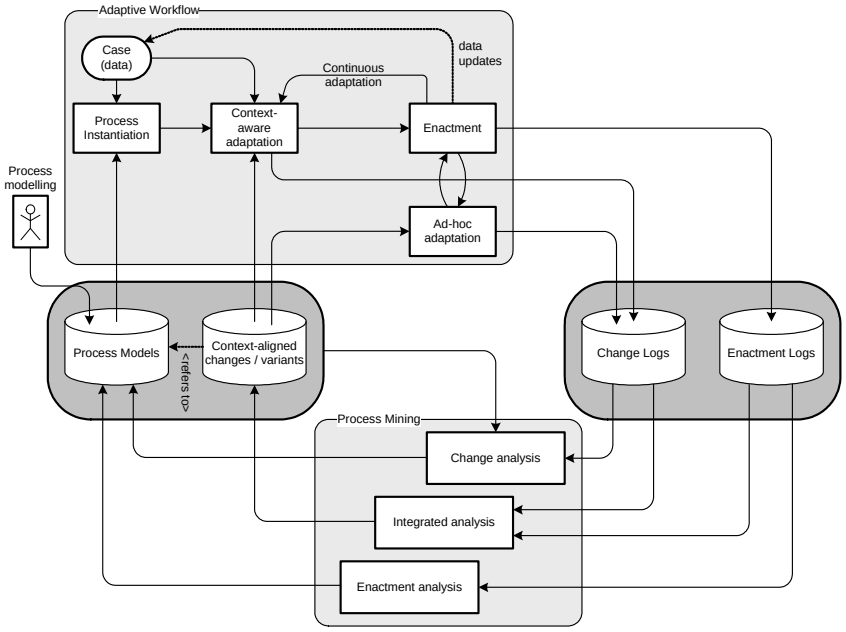

Fig. 2. Integration of Process Mining and Adaptive Process Management

The framework depicted in Figure 2 illustrates how such an integration could look like. Adaptive PMSs, visualized in the upper part of this model, operate on pre-defined process schemas. The evolution of these schemas 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 can additionally log the sequence of change operations imposed on a process schema for every executed case, producing a set of *change logs*. Process mining techniques that integrate into such system in form of a feedback cycle may be positioned in one of three major categories:

Change analysis: Process mining techniques from this category make use of change log information, besides the original process schemas and their variants. One goal is to determine common and popular variants for each process schema, which may be promoted to replace the original schema. Possible

ways to pursue this goal are through statistical analysis of changes or their abstraction to higher-level schemas. From the initially used process schema and a sequence of changes, it is possible to trace the evolution of a process schema for each case. Based on this information, change analysis techniques can derive abstract and aggregate representations of changes in a system. These are valuable input for analysis and monitoring, and they can serve as starting point for more involved analysis (e.g., determining the circumstances in which particular classes of change occur, and thus reasoning about the driving forces for change).

Integrated analysis: This analysis uses both change and enactment logs in a combined fashion. Possible applications in this category could perform a context-aware categorization of changes as follows. Change process instances, as found in the change logs, are first clustered into coherent groups, e.g. based on the similarity of changes performed, or their environment. Subsequently, change analysis techniques may be used to derive aggregate representations of each cluster. Each choice in an aggregate change representation can then be analyzed by comparing it with the state of each clustered case, i.e. the values of case data objects at the time of change, as known from the original process schema 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 schema which need to be changed, e.g. paths having become obsolete. Traditional process mining techniques like control flow mining and conformance checking can be adapted with relative ease to provide valuable information in this context. For example, conformance checking, i.e. determining the "fit" of the originally defined process schema and the recorded enactment log, can show when a specific alternative of a process schema has never been executed. Consequently, the original process schema may be simplified by removing that part. Statistical analysis of process enactment can also highlight process definitions, or variants thereof, which have been rarely used in practice. These often clutter the user interface, by providing too many options, and they can become a maintenance burden over time. Removing (or hiding) them after a human review can significantly improve the efficiency of a process management system.

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 schemas to a specialist before their deletion or change, rather than having the system perform these tasks autonomously. Also, some users may find it useful to have the system provide active assistance in adapting a process definition, while others would prefer the system not to intervene without their explicit request.

In every case, the change feedback cycle should provide and store extensive *history information*, i.e. an explicit log of actions performed in the feedback

cycle, and their intermediate results. This enables users and administrators to trace the motivation for a change, and thereby to understand the system. Also, in the long run it allows both administration staff to supervise the progress of autonomous adaption in the system, and enable the system itself to monitor its own performance.

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.

4 Anatomy of Change

Adaptive PMSs do not only create process enactment logs, but *they also log the sequence of changes applied to a process schema*. 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 approaches described in Section 6.

Logically, a process change is accomplished by applying a sequence of change operations to the respective process schema [34]. 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 schema 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 schema and this can be overcome by conducting concomitant changes. Generally, 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 respective *change process instance* (for details see [37]).

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. Process schemas can be defined in a number of different notations, e.g. EPCs, YAWL,

BPEL, BPMN, UML Activity Diagrams, or proprietary notations for systems like Staffware and FLOWer. In this paper we use a simplified notation borrowed from the ADEPT system and assume a transition system for formalization and expressing semantics. However, it is important to point out that the properties and approaches introduced in this paper are applicable to any process modelling notation. Also, tools like ProM can be used to handle a large number of different process modelling notations and translate between them.

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) [14]. Note that s_{end} denotes the *correct*, and thus desirable, final state of a process. If the process schema is incorrectly specified or executed, there may be further possible final states. However, we take the correctness of process schemas as precondition, and therefore the assumption of a single final state is valid. A simple example of a process schema is shown in Figure 3, consisting of a sequence of five activities (Enter Order, Inform Patient, Prepare Patient, Examine Patient, and Deliver Report).

Change logs can now be formally defined as follows:

Definition 2 (Change Log). *Let* $\mathcal{P}$ *be the set of possible process schemas and* $\mathcal{C}$ *the set of possible process changes, i.e., any process change* Δ *is an element of* $\mathcal{C}$*. A* change log instance σ *is a sequence of process changes performed on some initial process schema* PS*, i.e.,* $\sigma \in \mathcal{C}^*$ *(where* $\mathcal{C}^*$ *is the powerset of* $\mathcal{C}$*). A* change log L *is a set of change log instances, i.e.,* $L \subseteq \mathcal{C}^*$*.*

Note that a change log is defined as a set of instances. It would be more natural to think of a log as a multi-set (i.e., bag) of instances because the same sequence of changes may appear multiple times in a log. We abstract from the number of times the same sequence occurs in the change log for the sake of

simplicity, because in this paper we only consider the presence of changes and not their frequency. Note that in tools like ProM the frequency definitely plays a role and is used to deal with noise and to calculate probabilities.

Figure 3 shows an example of a change log in column *b)*, which is composed of nine change log instances cl_{I1} – cl_{I9}. The first change log instance cl_{I1}, for example, consists of two consecutive change operations $op1$ and $op2$.

Changes can be characterized as operations, which are transforming one process schema into another one. The same holds for sequences of change operations, i.e. change log instances. This can be formalized as follows:

Definition 3 (Change in Process Schemas). *Let $PS, PS' \in \mathcal{P}$ be two process schemas, let $\Delta \in \mathcal{C}$ be a process change, and $\sigma = \langle \Delta_1, \Delta_2, \ldots \Delta_n \rangle \in \mathcal{C}^*$ be a change log instance.*

- *$PS[\Delta\rangle$ if and only if Δ is applicable to PS, i.e., Δ is possible in PS.*
- *$PS[\Delta\rangle PS'$ if and only if Δ is applicable to PS (i.e., $PS[\Delta\rangle$) and PS' is the process schema resulting from the application of Δ to PS.*
- *$PS[\sigma\rangle PS'$ if and only if there are process schemas $PS_1, PS_2, \ldots PS_{n+1} \in \mathcal{P}$ with $PS = PS_1$, $PS' = PS_{n+1}$, and for all $1 \leq i \leq n$: $PS_i[\sigma\rangle PS_{i+1}$.*
- *$PS[\sigma\rangle$ if and only if there is a $PS' \in \mathcal{P}$ such that $PS[\sigma\rangle PS'$.*

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 PS, between the node sets A and B, if these node sets are indeed contained in PS and the activity X is not already contained in PS. Note that we do not allow duplicate tasks, i.e. an activity can be contained only once in a process schema.

For an example, consider the first process instance I_1, and its associated change log instance cL_{I1}, in Figure 3 . The first change performed, $op1$, is inserting a new activity "Lab test" between activities "Examine patient" and "Deliver report". Obviously, this change is applicable to the original process schema (the horizontal sequence of five activities), the resulting process schema being a sequence of six activities including "Lab test". The second change operation, $op2$, moves the second activity "Inform Patient" one position to the right, between activities "Prepare patient" and "Examine patient". This change is applicable to the process schema resulting from change operation $op1$, which in turn makes the sequence cL_{I1} applicable to the original process schema.

Any change log refers to a specific process schema, which has been the subject of change. Thus, whether a specific change log is *valid* can only be determined by also taking into account the original process schema:

Definition 4 (Valid Change Log). *Let $PS \in \mathcal{P}$ be a process schema and $L \subseteq \mathcal{C}^*$ a change log for PS. L is valid with respect to PS if for all $\sigma \in L$: $PS[\sigma\rangle$.*

Figure 3 shows an example for a valid change log in column *b)*, consisting of nine change log instances cL_{I1} – cL_{I9}, which are all applicable to the original process schema.

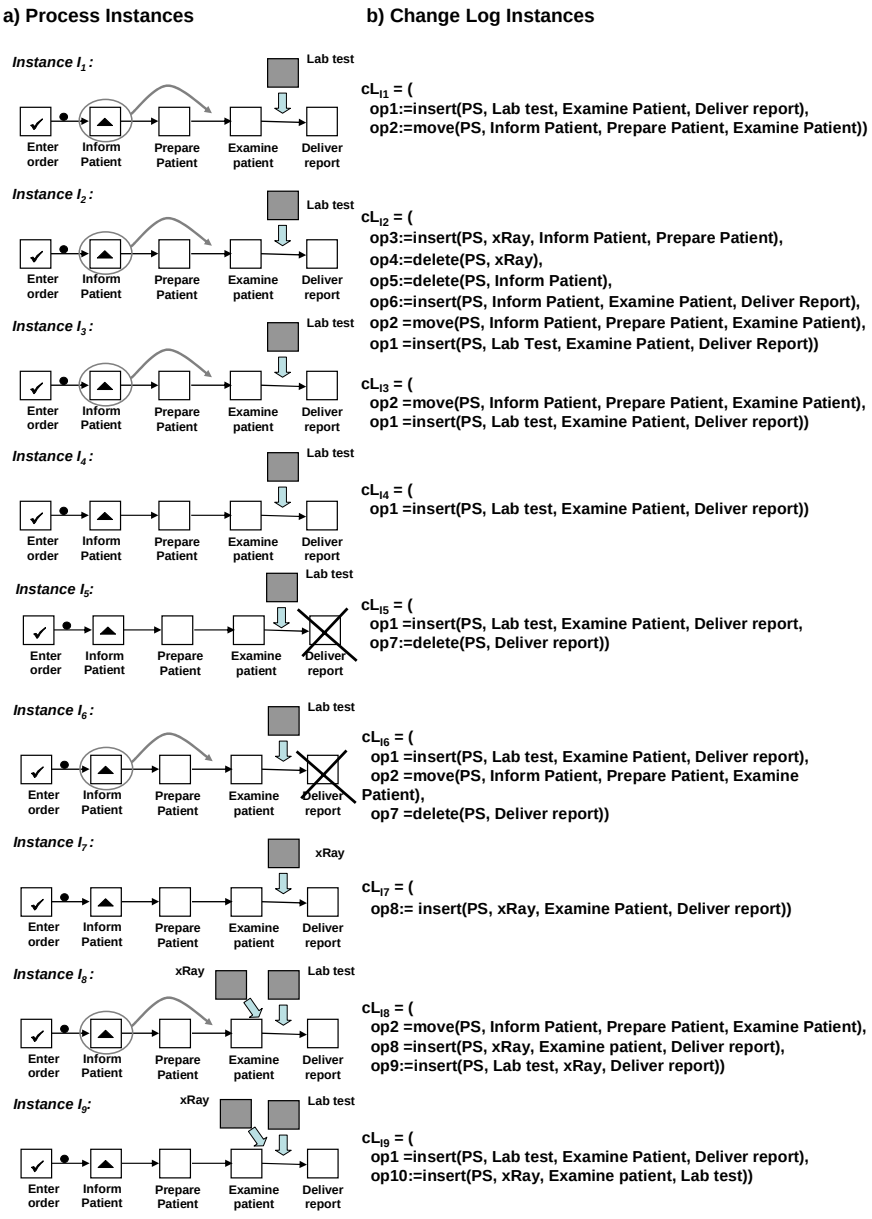

Fig. 3. Modified Process Instances and Associated Change Log Instances

In the following we represent change log entries by means of high-level change operations since we want to exploit their semantical content (see Figure 3 for an example). However, basically, the mining approaches introduced later can be adapted to change primitives as well. Table 1 presents examples of *high-level change operations* on process schemas which can be used at the process type as well as at the process instance level to create or modify process schemas. Although the change operations are exemplarily defined on the ADEPT meta model (see [34] for details) they are generic in the sense that they can be easily transferred to other meta models as well (e.g. [35]).

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

Change Operation **Δ Applied to S**	opType	subject	paramList
insert(PS, X, A, B, [*sc*])	insert	X	PS, A, B, [*sc*]
Effects on PS: 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 PS, and X must not be contained in PS yet (i.e., no duplicate activities!)			
delete(PS, X)	delete	X	PS
Effects on PS: deletes activity X from PS **Preconditions:** activity X must be contained exactly once in PS			
move(PS, X, A, B, [*sc*])	move	X	PS, A, B, [*sc*]
Effects on PS: 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 PS			
replace(PS, X, Y)	replace	X	Y
Effects on PS: replaces activity X by activity Y **Preconditions:** activity X must be contained exactly once in PS, and activities X and Y must be of the same type (e.g., have the same input / output parameters)			

5 Applying Existing Process Mining Techniques

Change logs bear a strong structural resemblance to regular process enactment logs. They are essentially also sequences of events which have occurred in the realm of a process, and in a specific case. As shown in Figure 4, mining the meta-process of change in an adaptive PMS is also strikingly similar to traditional process mining. Both start from comparably structured log files and strive to derive a model describing the causal relations between observed activities.

This similarity between traditional process mining (i.e. focusing on the execution control flow) and change mining raises the question of whether well-known process mining techniques can be applied to change logs, and if so, how meaningful the results yielded by them are in this context.

Section 5.1 introduces the MXML format for storing event logs, and defines a mapping of change logs to MXML. Once mapped to MXML, change logs can be analyzed by regular process mining techniques. Thus, Section 5.2 investigates

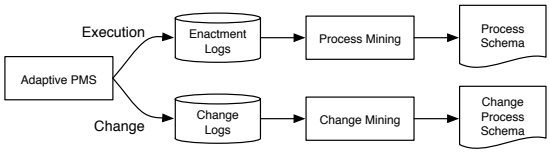

Fig. 4. Traditional Process Mining and Change Mining

the usefulness of traditional control flow mining techniques in the context of change logs.

5.1 Change Logs

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 each 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 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 the *Timestamp* and the *Originator*. The timestamp holds the exact 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.

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 (i.e. into change process instances). 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 schema, 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 schema, as e.g. seen for Instance I_1 in Figure 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 process schema 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 predefined process schema, this information is

not available. Thus, a concatenation of change type and subject is used for the workflow model element field.

Once the change log information is mapped and converted to MXML, it can be readily mined by any process mining algorithm in the ProM framework. The next section investigates the appropriateness of traditional process mining algorithms in the context of change logs.

5.2 Evaluation of Existing Mining Techniques

As discussed in the previous section, mapping process change logs to the existing MXML format for execution logs enables the use of existing mining techniques, such as those implemented within the ProM framework, also for mining change logs. The question is how "well" these algorithms perform when being applied to change logs. We have evaluated a selection of mining algorithms implemented within ProM with respect to change log mining.

First of all, we extended the ADEPT demonstrator by automatic generation and export of change logs [53]. When generating change logs, users can specify how many process instances are to be modified and how often single change operations or change sequences shall occur[6]. The evaluation itself was carried out based on two sample processes of different complexity [53].

To be able to compare the resulting change processes produced by the different mining algorithms afterwards we need to specify respective quality criteria: The most important criterion is how "well" the change process reflects the actual dependencies between change operations. As for process instance I_2 depicted in Figure 3, for example, change operation op_4 depends on previous change operation op_3. This dependency should be reflected as a sequence $op_3 \longrightarrow op_3$ within the change process. Contrary, non-existing dependencies between change operations should not be reflected as sequences, i.e., changes which do not depend on each other should be ordered in parallel within the change process.

α Algorithm: When mostly similar changes are applied to the process instances (i.e., low variation within the applied change operations) the α algorithm is able to detect actually existing dependencies between changes. However, the α algorithm also generates dependencies between changes which do not exist in reality. This is caused by their order within the change logs. If more and more different change operations are applied, the resulting change process becomes less meaningful, i.e., no conclusions about dependencies between changes can be drawn anymore.

Multi-Phase Miner: This algorithm first represents the change operations which have been observed for each process instance separately. However, at this instance level the Multi-Phase Miner only produces change sequences. Within the aggregated workflow graph the annotations reflecting the frequency of the change operations may be helpful to find actual dependencies between the change operations. As for the α algorithm, with increasing complexity of the changes

[6] Within the evaluation we focused on insert, delete, and move operations.

and the processes the number of dependencies between changes which are not existent in reality increases as well.
HEURISTICS MINER: The Heuristic Miner annotates the resulting change process model with observation frequencies as well. Contrary to Multi Phase Mining, the Heuristic Miner only displays the most frequent edges within the change process such that it is less likely to include unnecessary dependencies. Furthermore, if change operations are correctly detected as being independent they are displayed in parallel. However, even the Heuristics Miner still produces unnecessary dependencies between change operations. Furthermore, the detection of actually existing dependencies based on frequencies can be expensive and even wrong.

The fundamental problem is that change is an activity far less observed in an adaptive PMS than regular execution. Therefore, the *completeness* of change logs, i.e. their property to record independent (i.e. parallel) activities in any possible order, cannot be taken for granted due to their limited availability. This has been simulated by using an incomplete subset of change logs, as can be expected in a real-life situation.

Our experiments with applying existing process mining algorithms to change logs have shown that their suitability in this context is limited. In the following section, we explore the nature of change in an adaptive system and the associated logs in more detail to find a more suitable means for detecting whether an observed ordering relation is actually necessary.

6 Change Mining

In this section we describe novel approaches 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 Section 6.2. This commutativity relation provides the foundation for our first mining algorithm for change processes, as introduced in Section 6.3. A second algorithm for mining change processes based on the theory of regions is presented in Section 6.4. Finally, Section 6.5 compares both approaches.

6.1 Motivation: 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 schema (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 Section 5.1 it has been defined that each change operation refers to the original process schema through three associations, namely the *subject*, *pre-*, and *post-set* of the change. As all these three associations can theoretically be bound

to any subset from the original process schema's set of activities[7], the set of possible change operations grows exponentially with the number of activities in the original process schema. This situation is fairly different from mining a regular process schema, where the number of activities is usually rather limited (e.g., up to 50–100 activities). Hence, the mining of change processes 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*. In order to come to a more compact representation of change processes, it is helpful to abstract from a certain subset of ordering 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. 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 can also be 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.

6.2 Commutative and Dependent Change Operations

When traditional process mining algorithms are applied to change logs, they often return very unstructured, "spaghetti-like" models of the change process (cf. Section 5.2). This problem is due to a large number of ordering relations which do not reflect actual dependencies between change operations. The concept of *commutativity* is an effective tool for determining, whether there indeed exists a causal relation between two consecutive change operations.

As it has been introduced in Section 4 (cf. Definition 3), change operations (and sequences thereof) can be characterized as transforming one process schema into another one. Thus, 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.

[7] Here we assume that the subset describing the *subject* field is limited to a size of one.

Definition 5 (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* [23, 27, 38], which regards two process schemas as equivalent if the sets of observable traces they can execute are identical. Since the number of traces a process schema 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 [24]. 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.

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

Definition 6 (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 effectively captures the ordering relation between two consecutive change operations. If two change operations are commutative according to Definition 6 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).

6.3 *Approach 1:* Enhancing Multi-phase Mining with Commutativity

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 first approach on an existing process mining technique. Among the available algorithms, the *multi-phase* algorithm [17, 18] 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 [17, 18]. 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 [17, 18] 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 Figure 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.

Figure 5 shows a Petri net model [15] of the change process mined from the example change log instances in Figure 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, respective 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 Figure 3, where change log instances hardly have common subsequences, this model delivers a high degree of abstraction.

When 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. 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

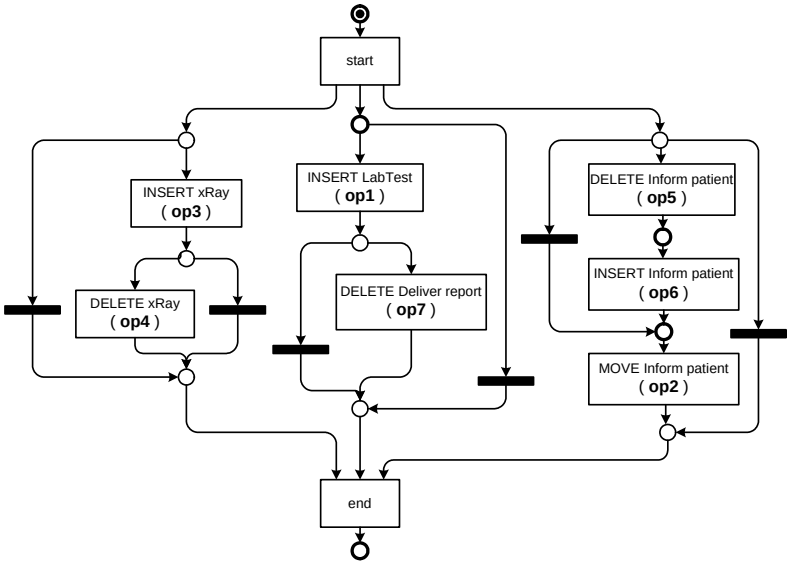

Fig. 5. Mined Example Process (Petri net notation)

relation, and thus no direct link between these change operations in the model shown in Figure 5.

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 6). 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 Figure 5.

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.

6.4 *Approach 2:* Mining Change Processes with Regions

The second approach towards mining change logs uses an approach based on the *theory of regions* [13] and exploits the fact that *a sequence of changes defines a state*, i.e., the application of a sequence of changes to some initial process schema results in another process schema. The observation that a sequence of changes uniquely defines a state and the assumption that changes are "memoryless" (i.e., the process schema resulting after the change is assumed to capture all relevant information) are used to build a *transition system*. Using the theory or regions, this transition system can be mapped onto a process model (e.g., a Petri net) describing the change process. To explain our second approach we first define what a transition system is.

Definition 7 (Transition System). *A (labeled) transition system is a tuple* $TS = (S, E, T, s_i)$ *where* S *is the set of states,* E *is the set of events,* $T \subseteq S \times E \times S$ *is the transition relation, and* $s_i \in S$ *is the initial state.*

A example of a simple transition system is $TS = (S, E, T, s_i)$ with $S = \{a, b, c\}$ (three states), $E = \{x, y, z\}$ (three events), $T = \{(a, x, a), (a, y, b), (b, z, a), (b, y, c), (c, z, b), (c, y, c)\}$ (six transitions), and $s_i = a$. Figure 6 shows this transition system graphically. The semantics of a transition system are simple, i.e., starting from the initial state, any path through the transition system is possible. For example, $\langle x, x, x, y, z, x\rangle$, $\langle y, y, y, z, z, x\rangle$, and $\langle\rangle$ (the empty sequence) are possible behaviors of the transition system depicted in Figure 6.

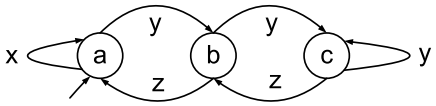

Fig. 6. Example transition system

Transition systems are the most basic representation of processes, e.g., simple processes tend to have many states (cf. "state explosion" problem in verification). However, using the theory of regions and tools like Petrify [13], transition systems can be "folded" into more compact representations, e.g., Petri nets. Especially transition systems with a lot of concurrency (assuming interleaving semantics) can be reduced spectacularly through the folding of states into regions, e.g., transition systems with hundreds or even thousands of states can be mapped onto compact Petri nets. However, before using the theory of regions to fold

transition systems into Petri nets, we first need to derive a transition system from a change log. To do this we first introduce some useful notation.

Definition 8 (Useful Notations). *Let $PS \in \mathcal{P}$ be a process schema and let $\sigma = \langle \Delta_1, \Delta_2, \ldots \Delta_n \rangle \in L$ be a change log instance from some valid log L.*

- *$\sigma(k) = \Delta_k$ is the k^{th} element of σ $(1 \leq k \leq n)$,*
- *$hd(\sigma, k) = \langle \Delta_1, \Delta_2, \ldots \Delta_k \rangle$ is the sequence of the first k elements $(0 \leq k \leq n)$ of σ (with $hd(\sigma, 0) = \langle\ \rangle$),*
- *$state(PS, \sigma, k) = PS'$ where $PS[hd(\sigma, k)\rangle PS'$, i.e., PS' is the process schema after the first k changes have been applied.*

Definition 8 shows that given an initial process schema PS and a change log instance σ, it is possible to construct the process schema resulting after executing the first k changes in σ. $state(PS, \sigma, k)$ is the state after applying $\Delta_1, \Delta_2, \ldots \Delta_k$. Note that $state(PS, \sigma, 0) = PS$ is the initial process and $state(PS, \sigma, n)$ is the state after applying all changes listed in σ.

Using the notations given in Definition 8 it is fairly straightforward to construct a transition system based on an initial process schema and a log. The basic idea is as follows. The states in the transition system correspond to all process schemas visited in the log, i.e., the initial process schema is a possible state, all intermediate process schemas (after applying some but not all of the changes) are possible states, and all final process schemas are possible states of the resulting transition system. There is a transition possible from a state PS_1 in the transition system to another state PS_2 if in at least one of the change log instances PS_1 is changed into PS_2.

Definition 9 (Transition System of a Change Log). *Let $PS \in \mathcal{P}$ be a process schema and L a valid change log for PS. $TS_{(PS,L)} = (S, E, T, s_i)$ is the* corresponding transition system*, where $S = \{state(PS, \sigma, k) \mid \sigma \in L \ \wedge\ 0 \leq k \leq |\sigma|\}$ is the state space, $E = \{\sigma(k) \mid \sigma \in L \ \wedge\ 1 \leq k \leq |\sigma|\}$ is the set of events, $T = \{(state(PS, \sigma, k), \sigma(k), state(PS, \sigma, k+1)) \mid \sigma \in L \ \wedge\ 1 \leq k < |\sigma|\}$ is the transition relation, and $s_i = PS$ is the initial state (i.e., the original process schema).*

Note that this approach assumes that changes are *memoryless*, i.e., the set of possible changes depends on the current process schema and not on the path leading to the current process schema. This means that if there are multiple "change paths" leading to a state PS', then the next change in any of these paths is possible when one is in state PS'. In other words: only the current process schema for the change process and not the way it was obtained. Note that this assumption is similar, but also different, from the assumption in the first approach: there commutative changes are assumed to occur in any order even when this has not been observed.

For technical reasons it is useful to add a unique start event *start* and state s_0 and a unique end event *end* and state s_e. This can be achieved by adding *start* and *end* to respectively the start and end of any change log instance. It can also be added directly to the transition system.

Definition 10 (Extended Transition System of a Change Log). *Let $PS \in \mathcal{P}$ be a process schema and L a valid change log for PS. $TS_{(PS,L)} = (S, E, T, s_i)$ is as defined in Definition 9. Let s_0, s_e, start, and end be fresh identifiers. $TS^*_{(PS,L)} = (S^*, E^*, T^*, s_i^*)$ is the* extended transition system*, where $S^* = S \cup \{s_0, s_e\}$, $E^* = E \cup \{start, end\}$, $T^* = T \cup (\bigcup_{\sigma \in L} \{(s_0, start, state(PS, \sigma, 0)), (state(PS, \sigma, |\sigma|), end, s_e)\})$, and $s_i = s_0$.*

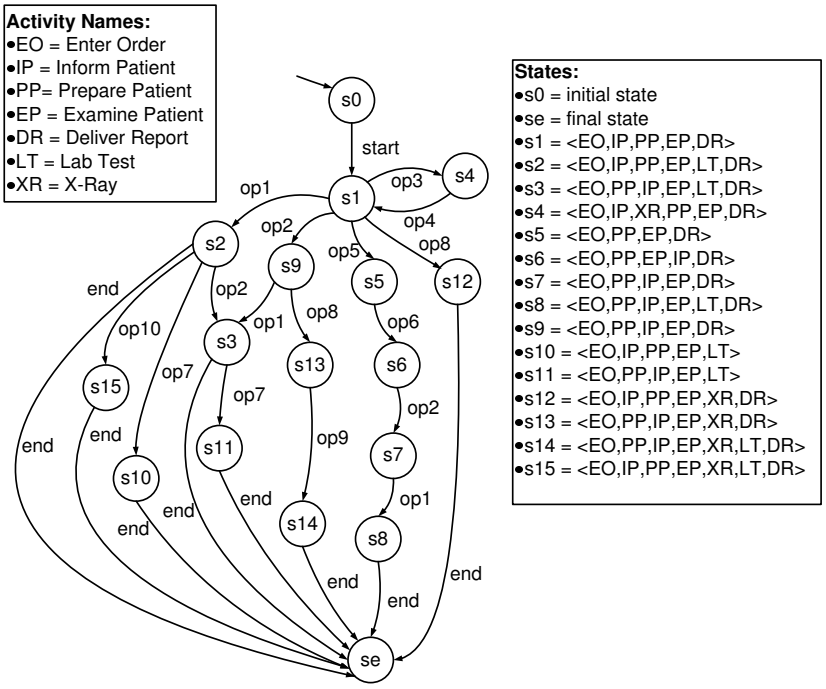

Fig. 7. Transition system based on the change log shown in Figure 3

Figure 7 shows the transition system obtained by applying Definition 10 to the running example, i.e., the change log depicted in Figure 3 (consisting of nine change log instances and ten different change operations) is used to compute $TS^*_{(PS,L)}$. For convenience we use shorthands for activity names: *EO = Enter Order*, *IP = Inform Patient*, *PP = Prepare Patient*, *EP = Examine Patient*, *DR = Deliver Report*, *LT = Lab Test*, and *XR = X-Ray*. Figure 3 defines ten different change operations, these correspond to the events in the transition system. Moreover, the artificial start and end are added. Hence $E = \{start, op1, op2, \ldots, op10\}$ is the set of events. The application of a change operation to some process schema, i.e., a state in Figure 7, results in a new state. Since in this particular example all process schemas happen to be sequen-

tial, we can denote them by a simple sequence as shown in Figure 7. For example, $s1 = \langle EO, IP, PP, EP, DR \rangle$ is the original process schema before applying any changes. When in the first change log instance $op1$ is applied to s_1 the resulting state is $s2 = \langle EO, IP, PP, EP, LT, DR \rangle$ (i.e., the process schema with the lab test added). When in the same change log instance $op2$ is applied to s_2 the resulting state is $s3 = \langle EO, PP, IP, EP, LT, DR \rangle$ (i.e., the process schema where *Inform Patient* is moved). This can be repeated for all nine instances, resulting in fifteen states plus the two artificial states, i.e., $S = \{s_0, s_1, \ldots, s_{15}, s_e\}$. Using the approach defined in definitions 9 and 10, the transition system shown in Figure 7 is obtained.

As indicated at the start of this section, transition systems can be mapped onto Petri nets using synthesis techniques based on the so-called *regions* [13, 21]. An example of a tool that can create a Petri net for any transition system using regions is *Petrify* [12]. In this paper we do not elaborate on this and only present the basic idea.

Given a transition system $TS = (S, E, T, s_i)$, a subset of states $S' \subseteq S$ is a *region* if for all events $e \in E$ one of the following properties holds:

- all transitions with event e *enter the region*, i.e., for all $s_1, s_2 \in S$ and $(s_1, e, s_2) \in T$: $s_1 \notin S'$ and $s_2 \in S'$,
- all transitions with event e *exit the region*, i.e., for all $s_1, s_2 \in S$ and $(s_1, e, s_2) \in T$: $s_1 \in S'$ and $s_2 \notin S'$, or
- all transitions with event e *do not "cross" the region*, i.e., for all $s_1, s_2 \in S$ and $(s_1, e, s_2) \in T$: $s_1, s_2 \in S'$ or $s_1, s_2 \notin S'$.

The basic idea of using regions is that each region S' corresponds to a place in the corresponding Petri net and that each event corresponds to a transition in the corresponding Petri net. Given a region all the events that *enter* the region are the transitions producing tokens for this place and all the events that *leave* the region are the transitions consuming tokens from this place. In the original theory of regions many simplifying assumptions are made, e.g., elementary transitions systems are assumed [21] and in the resulting Petri net there is one transition for each event. Many transition systems do not satisfy such assumptions. Hence many refinements have been developed and implemented in tools like Petrify [12, 13]. As a result it is possible to synthesize a suitable Petri net for any transition system. Moreover, tools such as Petrify [12] provide different settings to navigate between compactness and readability and one can specify desirable properties of the target model. For example, one can specify that the Petri net should be free-choice. For more information we refer to [12, 13].

Figure 8 shows the Petri net corresponding to the transition system of Figure 7. This Petri net was constructed using regions. It can be observed that the Petri net is more or less identical to the transition system. One can use Petrify with different settings. This way it is possible to construct a more compact Petri net, however, this process model is less readable. At first sight, it may be disappointing to see the Petri net shown in Figure 8. However, one should note that the change log depicted in Figure 3 only has nine change log instances, i.e., compared to the number of change operations, the number of instances is

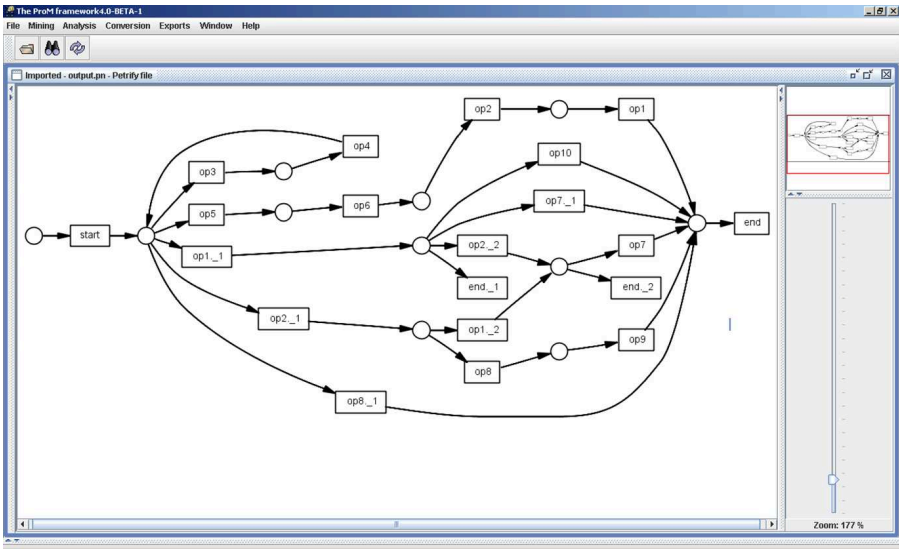

Fig. 8. Screenshot of ProM showing the Petri net obtained for the change log depicted in Figure 3

rather low. It seems that only a few of the possible interleavings are present in Figure 3. As a result, the transition system in Figure 8 seems to be the result of a number of *particular* examples rather than a description of the full behavior. The strength and also the weakness of our approach using regions is that the corresponding Petri net has a behavior identical to the transition system. The transition system is branching bisimilar to the Petri net. Hence there is no generalization for going from Figure 7 (transition system) to Figure 8 (Petri net).

Figure 9 shows another example of a transition system that exhibits more parallelism (i.e., there are more interleavings). When we apply our approach to this transition system we obtain the Petri net shown in Figure 10. Clearly this Petri net reveals the behavior implied by Figure 9 in a compact and readable manner. This example shows the potential of applying logs to transition systems with more parallelism.

6.5 Comparing Both Approaches

We have introduced two new process mining approaches based on the characteristics of change logs. The first approach is based on the *multi-phase* algorithm [17, 18]. However, the original algorithm has been enhanced to exploit information about commutativity of change operations. If there are independent changes (i.e., changes that operate on different parts of the schema), it is not necessary to see all permutations to conclude that they are in parallel. The second approach is

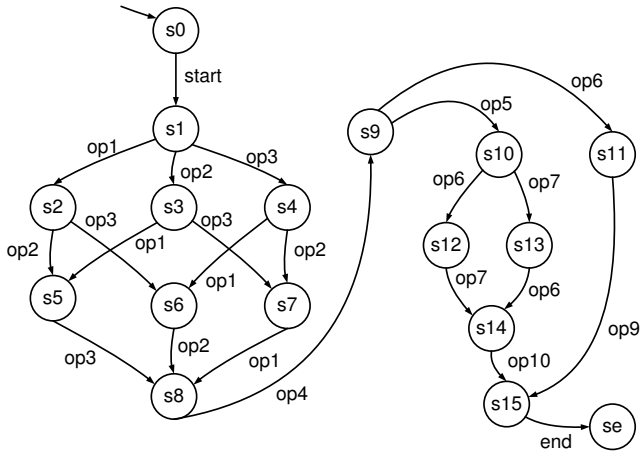

Fig. 9. A transition system with more parallelism

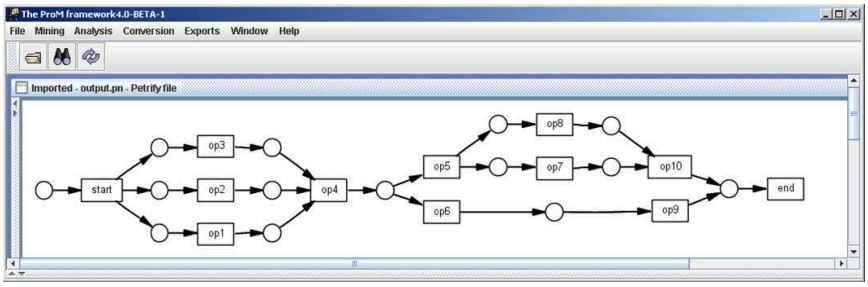

Fig. 10. Screenshot of ProM showing the Petri net obtained for the transition system depicted in Figure 9

based on the observation that given an original schema and a sequence of change operations, it is possible to reconstruct the resulting process schema. This can be used to derive a *transition system* where the states are represented by possible (intermediate) process schemas. Using *regions* such a transition model can be translated into an equivalent Petri net describing the change process.

Both approaches make *assumptions* and *generalize* on the basis of observed behavior.

The first approach assumes that *commutativity implies parallelism*, i.e., even if change operations are only observed in a specific order, commutativity still suggests that potentially these change operations could be performed concurrently. This way the approach generalizes things dramatically. Consider for example the Petri net in Figure 5 which has been derived from the change log depicted in

Figure 3. This Petri net allows for much more sequences of changes than the ones that happen to be present in the change log.

The second approach also makes an assumption to generalize things: changes are *memoryless*. It is assumed that the set of possible changes only depends on the current process schema and not on the path leading to this schema. This allows for some form of generalization quite different from the assumption that commutativity implies parallelism. The second approach first constructs the transition system. In this step some generalization takes place due to the memoryless state encoding. Then this transition system is mapped onto an *equivalent* Petri net, i.e., when constructing the Petri net no further generalizations take place. Figure 8 shows the Petri net which has been derived from the change log depicted in Figure 3.

The Petri net in Figure 8 is very different from the one in Figure 5. This illustrates that both approaches produce different results, i.e., *they provide two fundamentally different ways of looking at change processes*. It seems that in this particular example, the first approach performs better than the second. This seems to be a direct consequence of the small amount of change log instances (just nine) in comparison with the possible number of change operations. When there is an abundance of change log instances, the second approach performs better because it more precisely captures the observed sequences of changes. Moreover, the second step could be enhanced by generalization operations at the transition system level, e.g., using commutativity.

7 Context and Learning

So far, we have discussed what constitutes a process change, how change information can be represented in logs, and how these logs can be mined to deliver valuable insights into the scope of change. The latter enables us to understand *how* processes deviate from predefined routines. However, what is still needed is an understanding of *why* changes occur and *what* causes change (i.e., the semantic reasons for a change). The answers to these questions can be found in the *context* of a process [42, 46]. In other words, context is the relevant subset of the entire situation of a business process that makes change necessary. This section attempts to arrive at a useful understanding of context (Section 7.1). We further show how context information can be elicited and stored (Section 7.2) and, finally, how it can be exploited by machine learning techniques to learn from change (Section 7.3).

7.1 Context Information

Understanding the environment in which a process is embedded as the context of the process raises the question what exactly constitutes this context. Definitions of context in related disciplines such as Web systems engineering [26] and mobile applications research [29] focus around the users and their interaction with the systems [16, 45]. They typically include only reduced information such

as locality (e.g., what is the closest restaurant and how do I make a booking?) and user characteristics (e.g., what type of food does the user of the mobile application like?). However, attempting to introduce these interaction-focussed approaches to the area of process flexibility requires that the process is aware of its surroundings *irrespective* of user interactions. In order to facilitate this general awareness in a structured manner, the challenge is to identify the *relevant* context of business process, i.e., those elements or variables in the context that have impact on process design or execution (e.g., location, but not legislation).

The context of a business process can generally be divided into two distinct parts. The *intrinsic context* is constituted by all data and information that is directly accessible by the business process. This will usually resolve to the data structures defined in the process definition itself, i.e., all data created and modified during its execution (e.g., the name and address of a customer). Such a process definition would at least include the control flow logic, involved informational data, and organizational resources [25]. On the other hand, the *extrinsic context* of a process cannot be defined as concisely. It contains all information that is available at some stage during the execution of a business process, and that could potentially have influenced decisions in this process. As such, the extrinsic context is very large, containing basically all knowledge available, both in the immediate environment of a process (e.g., high system load or crash) and in its external environment (e.g., the weather or stock market situation during execution).

From the complete extrinsic context of a process, it is necessary to determine the subset that is *relevant*, i.e., that has had an actual chance of influencing the execution. One obvious criterion for this decision is *time*, i.e. only that set of information which was available at a certain point in time may be relevant for decisions at that time. A second criterion is *involvement*, which means that for context information to be relevant, at least one person or system involved in the process must have had access to it so to be able to react upon it. Obviously it is not possible to determine involvement with absolute certainty, e.g., one cannot know for certain all knowledge of persons involved in a process. However, by using suitable heuristics (e.g., based on locality or other spatial data) the involvement criterion can restrict the extrinsic context to a reasonable subset.

The above relevancy criteria can be determined more or less automatically, i.e. they are of a technical nature. Any further selection of relevant context, however, needs to be based on semantics of information, i.e. their level of contribution to the *objectives* (or goals) of the process. In general terms, goals, when applied to process modeling, specify the final state that a process seeks to reach (e.g., "complete an order") [49]. Thus, from a semantic viewpoint, the relevant extrinsic context is the set of extrinsic variables that, when changed, impacts the transition between the predefined process execution steps in a way that affects the extent of achieving the goal [42].

A consideration of goals thus allows identifying relevant context information. This can be achieved by decomposing the process into information that is contained in traditional process descriptions, (e.g., control flow, data, resources,

applications), i.e., information intrinsic to the process definition, and extrinsic information that impacts goal achievement but goes beyond this traditional layer of description. This way, the notion of goals makes it possible to reduce the extrinsic context to a subset of relevance (see Figure 11.

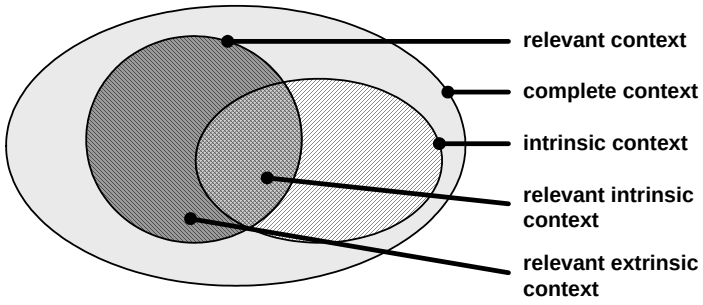

Fig. 11. The Context of a Business Process

In a basic format, a methodology for deriving relevant context information should consist of the following steps:

1. Determine process goal and identify appropriate measures.
2. Decompose process in accordance to goal in a set of goal-relevant information that is *intrinsic* (i.e., information that is contained in the process specification) and a set of goal-relevant information that is not fully dependent on the process definition, i.e., *extrinsic*.
3. Determine impact of goal-relevant, extrinsic information on the achievement of the goal to determine relevancy.
4. Type context and identify relevant state value ranges.

The accomplishment of each of these steps may benefit from existing research. Goal identification, for instance, is an important topic in the requirements engineering discipline. The basic idea for determining goals and relevant context is centered around the notion of a requirement chunk, which is a pair ⟨Goal,Case⟩ and denotes a potential way of achieving a goal in a given scenario (i.e., case) [41]. The identification process then starts with a given goal and describes a process case as a possible concretization of the goal. Clearly, this step results in one complete requirement chunk, made up of goal-relevant information that is either intrinsic or extrinsic to the process definition. Re-iterations of this process for various cases can then be used to elicit the set of information that is relevant to all cases but not intrinsic — and thus denotes relevant extrinsic context that, together with the relevant intrinsic context information available from the process description, can then be typed, conceptualized and utilized.

7.2 Elicitation and Storage of Context Information

The definition of the relevant context of a business process is necessary for learning from it, both from a practical perspective (i.e., limitation is necessary to preserve abstraction), as well as from a technical one (i.e., being able to cope with technical limitations of memory and computation time). In order to make this relevant context available for learning, suitable solutions have to be found for its elicitation (i.e., retrieval) and storage.

Context information can be retrieved from a wide range of potential data sources. Large parts of the intrinsic context can be extracted from enactment logs, which often include information about time and value of a data modification. Also, queries can be executed on databases, which hold relevant context information for the process. The latter, however, poses the problem that most conventional databases do not provide history information of data, i.e., once a new value is provided it usually erases the previous one, and the time of change is usually unknown. Thus, context information can be divided into *timed history* and the *static context*, with the latter often being identified as a snapshot of context data at the end of execution (e.g., the final state of context information stored in a database).

As context information is most useful when it is timed, an obvious means of storage is to enhance change logs with context data. This makes it possible to structure the context history in suitable chunks, i.e., the structural states of a process (between change operations). Technically, this is accomplished by examining each change event, acquiring timed context information for the time of its occurrence, and then enriching it with the elicited context information.

The static context of a process can obviously not be used to learn from single changes in a case, as it remains unchanged. However, it can point out to drivers for change when the static context of a number of cases are being compared. Then, it would, for instance, be possible to find *patterns* in the static context that always appear together with a certain type of change. Thus, the static context may be stored in an unordered fashion, associated to a specific case (e.g., listing the static context fields at the beginning of each case).

Approaches for structuring, understanding and describing semantic context information in a meaningful and applicable manner could again benefit from research in related disciplines. We found that in the area of context modeling and description a substantial amount of research has already been conducted, e.g. in the form of context architectures [47] or context ontologies [10]. For instance, the Context Ontology Language [50] is designed to accommodate selected aspects of context such as temperature, scales, the relative strengths of aspects and further meta data. A promising approach stems from the work of Analyti et al. [6] who build directed graphs and use relationships such as generalization and classification to describe context.

7.3 Learning from Context Information

Assuming access to context information, it would become possible to investigate changes in a process together with the reasons for the change decisions take

along the execution of a process. This can be achieved by looking at change process models and the decision points contained within. Decision points in a change process model are the points where the process splits into alternative paths. They describe crucial points where a set of cases, which have so far been subject to the same modifications, is split into subsets which exhibit a different set of changes from then on. The most significant of these decision points is the implicit choice, whether a case is subject to modification in the first place, or whether it can be executed without applying changes to the process model.

While a change process model itself is already immensely helpful, for instance for monitoring an adaptive PMS, it can not be used to learn from process change. Learning shall be interpreted as deriving information from an adaptive PMS, which can be used to facilitate future changes, or make them obsolete in the first place. The fundamental premise is that cases in which a certain change (or, pattern of changes) has been applied will exhibit distinct *patterns* in their context information. For example, whenever a customer's record has been stored for more than two years in the customer database, the otherwise necessary credit check is removed from the process schema. Note that these context data characteristics need not represent the reason for change per se, they can also describe further effects of the original reason for change, which itself is not found in the context information.

As sketched above, the set of context information can be very large. Thus, identifying the pivotal data elements, or patterns thereof, which are unique for a specific change, is like looking for a needle in a haystack. Fortunately, existing Machine Learning (ML) [30, 58] techniques can solve exactly this problem in an automatic fashion. *Classification algorithms*, which are a subset of ML algorithms, take for input a classified set of examples, the so-called *training set*. Once this set has been analyzed, the algorithm is capable of classifying previously unknown examples.

In the example process in Figure 5, let us assume that we are interested in the decision point after change operation "INSERT LabTest". The question is what made it necessary for some cases to delete the activity "Deliver report", while this was skipped for others. The first step in answering this question is to select the training set, i.e., the subset of cases for which "LabTest" has been inserted. Subsequently, all cases in the training set are classified according to whether "Deliver report" has been deleted later on.

Training a *decision tree* algorithm [31, 44] with this classified set may for example yield the decision tree shown in Figure 12. The root node first tests the value for the "Urgency" context attribute; if it has a value of "high", the case is classified as "TRUE", meaning that "Deliver report" is deleted. Correspondingly, cases with low urgency do not have this activity deleted ("FALSE"). For those cases where the urgency was set to "medium", the classification is performed based on the context attribute "Time". Thus, for cases with medium urgency, only those running at night had activity "Deliver report" deleted.

Note that while "Urgency" can be considered a static context attribute, "Time" will naturally change during execution, and should thus be determined

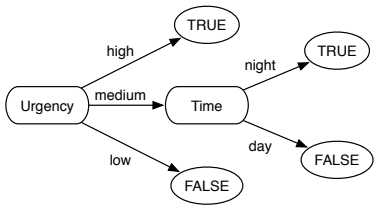

Fig. 12. A decision tree

in a *timed* fashion (i.e., with respect to the decision point under consideration, it may hold different values for the same case). It is further important to point out that classification algorithms may use only a subset of context attributes to generate their knowledge base. If a certain data field cannot be related to the classification under consideration (i.e., the path taken), it is automatically excluded from the decision tree.

Other classification algorithms from ML can generate a set of classification rules [30]. For example, one classification rule for the above example would be "`IF Urgency = medium AND Time = day THEN DELETE.DeliverReport = FALSE`". While it is also possible to generate rules from a decision tree, directly generated classification rules are often a more compact representation, as they can be defined more flexibly.

Note that both decision trees and classification rules are explicit representations of the algorithm's generated *knowledge base*, i.e., they can be easily stored and are understandable by humans. Other ML classification techniques like *neural networks* are also highly efficient in classifying unknown cases, after they have been trained on an existing set. However, their design makes it virtually impossible to comprehend why a specific case has been assigned to a certain class, i.e. they remain a "black box".

Classification algorithms are always focused on specific decisions, i.e. one branching point, and are thus dependent on the mining of a change process model in the first place. This means that for analyzing or predicting changes to a specific process schema, one has to identify all decision points in the change process model. For each of these decision points, the training set needs to be determined and enriched with specific context information before the classification algorithm can be applied to it.

An alternative to this approach is the *mining of association rules* [4]. Here, every case is regarded as a set of facts, where a fact can both be the occurrence of a change operation as well as a context attribute having a specific value. After identifying *frequent item sets*, i.e., facts that are often found together in a case, the algorithm can derive association rules. These rules describe, for instance, that for a large fraction of cases where an additional x-ray was inserted, the patient was older than 65 years and the doctor was female, an additional blood screening was inserted. Association rules are derived in a *global* manner,

viz., the order in which change operations occur is not taken into account. Depending on the circumstances, this feature can be beneficial or detrimental. For example, when there are hardly any causal relations between change operations, association rules may deliver more enlightening results. They can discover tacit relationships between change operations and context data, which cannot be captured by classification.

Knowledge gained from both classification and association rule algorithms can be used for a number of purposes. One important application is the prediction of future changes for a running case. Whether predicted changes are applied autonomously by the system (*self-adaption*), merely proposed to the user (*decision support*), or whether they are only used to identify abnormal cases (*monitoring*) is strongly dependent on the application scenario at hand.

The artificial limitation of the training set can also be used to highlight specific aspects. One example is to limit the case base used for learning to cases performed by a specific user, in order to customize the system to his behavior. Another possibility is to limit the set of context information under consideration, e.g. for highlighting the resource perspective by only looking at *who* was responsible for a certain change pattern.

One significant downside of most ML techniques is their high demand of computation time and memory. However, both resources are becoming less expensive every year, while the amount of data to be processed remains relatively stable. The complete process of learning from change using context can also be described as successively filtering the available context. While the time and involvement criteria deliver the first, coarse restriction of the context set, subsequent testing for interference with process goals further narrows this set down significantly. The application of ML techniques delivers the final identification of the drivers for change from the relevant context, and is able to relate them to one another (e.g., generating a decision tree). We believe that this structured approach can deliver precise results while still remaining feasible in practical settings, and is thus a foundation for the design of self-adapting PMSs.

8 Implementation and Tool Support

To enable experimentation with change logs and their analysis, an import plugin 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 [36]. ProM*import*[8] 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 schema in the ADEPT demonstrator prototype, after which a

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

respective change log can be imported and written to the MXML-based change log format described in Section 5.1.

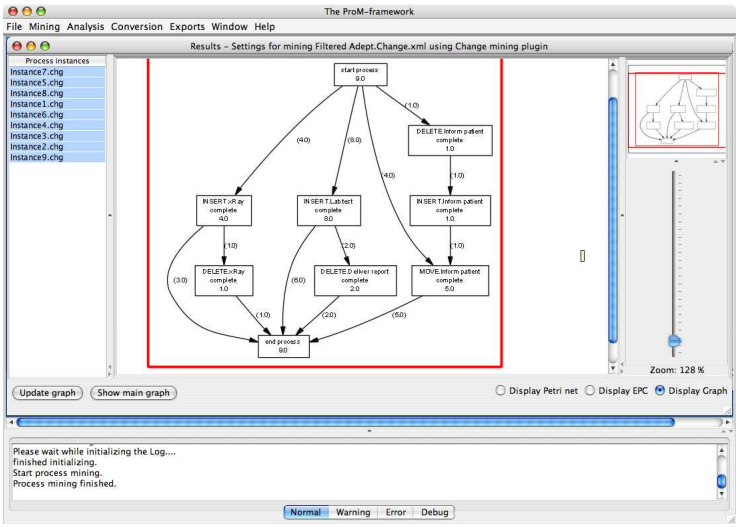

Fig. 13. Change Mining Plugin within ProM.

These change logs can then be loaded into the ProM framework[9]. A dedicated change mining plugin has been developed, which implements the commutativity-enhanced multi-phase algorithm described in Section 6.3. It is also possible to mine only a selection of the change log instances found in the log. The resulting change process can be visualized in the form of a workflow graph, Petri net, or EPC.

Figure 13 shows the change mining plugin implementing the approach based on commutativity, introduced in Section 6.3, within the ProM framework. It displays the example process introduced in Figure 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.

In Section 6.4 we proposed a second approach exploiting the specific nature of change logs. This approach builds a transition system assuming that change processes are memoryless, i.e., only the current schema is relevant and not the different intermediate process schemas. After building the transition system, the theory of regions can be applied to construct a Petri net of the change process. ProM has different plugins to construct, visualize, and analyze transition systems. However, there is not yet a plugin dedicated to change processes. Once there is a transition system ProM can create a Petri net using the Petrify import

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

and export plugin. Figures 8 and 10 show two screenshots of ProM. Both process models have been obtained using a combination of ProM and Petrify as described in Section6.4.

For determining the drivers for change, as proposed in Section 7.3, ProM features a *decision mining* plugin [44]. It uses a decision-tree algorithm, which is employed to unveil data properties used in decision points.

9 Related Work

Although process mining techniques have been intensively studied in recent years [2, 3, 11, 17, 18], no systematic research on analyzing process change logs has been conducted so far. Existing approaches mainly deal with the discovery of process schemas from execution logs, conformance testing, and log-based verification (cf. Section 2.1). The theory of regions [12, 13] has also been exploited to mine process schemas from execution logs [28], e.g. from logs describing software development processes. 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 [44]).

In recent years, several approaches for adaptive process management have emerged [38], most of them supporting changes of certain process aspects and changes at different levels. Examples of adaptive PMSs include ADEPT [36], CBRflow [55], and WASA [57]. 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 schemas from them.

CBRFlow [55] has focused on the question how to facilitate exception handling in adaptive PMSs. More precisely, it applies conversational case-based reasoning (CCBR) in order to assist users in defining ad-hoc changes and in capturing contextual knowledge about them (see our discussion in Section 7). This, in turn, increases the quality of change logs and change case bases respectively, and therefore provides new perspectives. CBRFlow, for example, supports the reuse of previous ad-hoc changes when defining new ones [55]. The retrieval of similar changes is based on CCBR techniques. CCBR itself is an extension of the original case-based reasoning (CBR) paradigm, which actively involves users in the inference process [5]. A CCBR system can be characterized as an interactive system that, via a mixed-initiative dialogue, guides users through a question-answering sequence in a case retrieval context. In [56] the authors further improve the support for change reuse by additionally discovering dependencies between different ad-hoc changes.

In [40, 54] the CBRFlow approach has been extended to a framework for integrated process life cycle support. In particular, knowledge from the change

case base is applied to continuously derive improved process schemas. This is similar to our goal for discovering optimized process schemas from change log. However, we have provided more advanced and more general mining techniques in this context, whereas [40] particularly makes use of semantically enriched log-files (e.g., information about the frequency of a particular change, user ratings, etc.). We will consider respective semantical and statistical information in our future work as well.

10 Summary and Outlook

This paper gave an overview of how comprehensive support for true process flexibility can be provided by combining adaptive process management systems with advanced process mining techniques. The integration of process mining with adaptive PMS enables the exploitation of knowledge about process changes from change logs, which in turn enables us to reason about the reason for the change, i.e., the contextual drivers for flexibility.

We have developed two mining techniques and implemented them as plugins for the ProM framework, taking ADEPT change logs in the mapped MXML format 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 so far. This discovery is based on the analysis of the (temporal) dependencies existing between the change operations applied to the respective process instance. Meaningful, compact representations of the change process can be derived by either making use of the concept of commutativity, or by application of the theory of regions, as has been shown. Altogether, the presented approaches 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 [44]) to change log information. In a related stream of work we continue our research on the identification and description of contextual information. We envision that based on an appropriate way of conceptualizing and identifying context, support can be developed to monitor, mine and control contextual variables in the environment of a process, which would in effect allow for true process agility, decreased reaction-time, and overall more flexible support in process design and execution.

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. Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, 1993.
5. D.W. Aha and H. Munoz-Avila. Introduction: Interactive case-based reasoning. *Applied Intelligence*, 14(7-8), 2001.
6. Anastasia Analyti, Manos Theodorakis, Nikos Spyratos, and Panos Constantopoulos. Contextualization as an independent abstraction mechanism for conceptual modeling. *Information Systems*, 32(1):24–60, 2007.
7. Pavel Balabko, Alain Wegmann, Alain Ruppen, and Nicolas Clement. Capturing design rationale with functional decomposition of roles in business processes modeling. *Software Process: Improvement and Practice*, 10(4):379–392, 2005.
8. I Bider. Masking flexibility behind rigidity: Notes on how much flexibility people are willing to cope with. In Jaelson Castro and Ernest Teniente, editors, *CAiSE'05 Workshops*, pages 7–18, Porto, Portugal, 2005. FEUP.
9. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. *Data and Knowledge Engineering*, 24(3):211–238, 1998.
10. Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review*, 18(3):197–207, 2003.
11. 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.
12. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325, March 1997.
13. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 47(8):859–882, August 1998.
14. 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.
15. 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.

16. Anind K. Dey. Understanding and using context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.
17. 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.
18. 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.
19. 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.
20. 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.
21. A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Informatica*, 27(4):315–368, 1989.
22. C.A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In N. Comstock, C. Ellis, R. Kling, J. Mylopoulos, and S. Kaplan, editors, *Proceedings of the Conference on Organizational Computing Systems*, pages 10 – 21, Milpitas, California, August 1995. ACM SIGOIS, ACM Press, New York.
23. R. van Glabbeek and U. Goltz. Refinement of Actions and Equivalence Notions for Concurrent Systems. *Acta Informatica*, 37(4–5):229–327, 2001.
24. R.J. van Glabbeek and W.P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600, 1996.
25. Stefan Jablonski and Christoph Bussler. *Workflow Management. Modeling Concepts, Architecture, and Implementation*. Thomson Computer Press, London, UK, 1996.
26. J. Wolfgang Kaltz, Jürgen Ziegler, and Steffen Lohmann. Context-aware web engineering: Modeling and applications. *Revue d'Intelligence Artificielle*, 19(3):439–458, 2005.
27. 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/)*.
28. Ekkart Kindler, Vladimir Rubin, and Wilhelm Schäfer. Process mining and petri net synthesis. In Johann Eder and Schahram Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 105–116. Springer Verlag, Vienna, Austria, 2006.
29. Marius Mikalsen and Anders Kofod-Petersen. Representing and reasoning about context in a mobile environment. In Stefan Schulz and Thomas Roth-Berghofer, editors, *First International Workshop on Modeling and Retrieval of Context*, volume 114 of *CEUR Workshop Proceedings*, pages 25–35, Ulm, Germany, 2004. CEUR.
30. T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
31. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
32. James Brian Quinn. *Intelligent Enterprise: A Knowledge and Service Based Paradigm for Industry*. Free Press, New York, NY, 1992.

33. Gil Regev and Alain Wegmann. A regulation-based view on business process and supporting system flexibility. In Jaelson Castro and Ernest Teniente, editors, *Proceedings of the CAiSE'05 Workshops. Vol. 1*, pages 91–98. FEUP, Porto, Portugal, 2005.
34. M. Reichert and P. Dadam. ADEPTflex - Supporting Dynamic Changes of Workflows Without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
35. 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)*, LNCS 2888, pages 407–425, Catania, Italy, November 2003.
36. 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.
37. S. Rinderle. *Schema Evolution in Process Management Systems*. PhD thesis, University of Ulm, 2004.
38. 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.
39. 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.
40. S. Rinderle, B. Weber, M. Reichert, and W. Wild. Integrating Process Learning and Process Evolution - A Semantics Based Approach. In *Proc. 3rd Int. Conf. on Business Process Management (BPM'05)*, pages 252–267, Nancy, 2005.
41. Colette Rolland, Carine Souveyet, and Camille Ben Achour. Guiding goal modeling using scenarios. *IEEE Transactions on Software Engineering*, 24(12):1055–1071, 1998.
42. Michael Rosemann, Jan Recker, Peter Ansell, and Christian Flender. Context-awareness in business process design. In Andy Koronios and Ed Fitzgerald, editors, *Australasian Conference on Information Systems*, Adelaide, Australia, 2006. Australasian Chapter of the Association for Information Systems.
43. A. Rozinat and W.M.P. van der Aalst. Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In C. Bussler et al., editor, *BPM 2005 Workshops (Workshop on Business Process Intelligence)*, volume 3812 of *Lecture Notes in Computer Science*, pages 163–176. Springer-Verlag, Berlin, 2006.
44. A. Rozinat and W.M.P. van der Aalst. Decision Mining in ProM. In S. Dustdar, J.L. Faideiro, and A. Sheth, editors, *International Conference on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 420–425. Springer-Verlag, Berlin, 2006.
45. Bill N. Schilit and Marvin M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22–32, 1994.
46. Albrecht Schmidt. Implicit human computer interaction through context. *Personal Technologies*, 4(2-3):191–199, 2000.
47. Johanneke Siljee, Sven Vintges, and Jos Nijhuis. A context architecture for service-centric systems. In Thomas Strang and Claudia Linnhoff-Popien, editors, *Location- and Context-Awareness: First International Workshop LoCA 2005*, volume 3479 of *Lecture Notes in Computer Science*, pages 16–25. Springer, Oberpfaffenhofen, Germany, 2005.

48. Pnina Soffer. On the Notion of Flexibility in Business Processes. In Jaelson Castro and Ernest Teniente, editors, *Proceedings of the CAiSE'05 Workshops. Vol. 1*, pages 35–42. FEUP, Porto, Portugal, 2005.
49. Pnina Soffer and Yair Wand. On the notion of soft-goals in business process modeling. *Business Process Management Journal*, 11(6):663–679, 2005.
50. Thomas Strang, Claudia Linnhoff-Popien, and Korbinian Frank. Cool: A context ontology language to enable contextual interoperability. In Jean-Bernard Stefani, Isabelle Demeure, and Daniel Hagimont, editors, *Distributed Applications and Interoperable Systems - DAIS 2003*, volume 2893 of *Lecture Notes in Computer Science*, pages 236–247. Springer, Paris, France, 2003.
51. D.M. Strong and S.M. Miller. Exceptions and exception handling in computerized information processes. *ACM Transactions on Information Systems*, 13(2):206–233, 1995.
52. Wil M. P. van der Aalst, Christian Günther, Jan Recker, and Manfred Reichert. Using process mining to analyze and improve process flexibility - position paper -. In Thibaud Latour and Michael Petit, editors, *The 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortium*, pages 168–177. Namur University Press, Luxembourg, Grand-Duchy of Luxembourg, 2006.
53. Marco Waimer. Integration of adaptive process management technology and process mining, 2006. (in German).
54. 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.
55. B. Weber, W. Wild, and R. Breu. CBRFlow: Enabling adaptive workflow management through conversational case-based reasoning. In *Proc. Eurpean Conf. on Case–based Reasoning (ECCBR'04)*, pages 434–448, Madrid, 2004.
56. B. Weber, W. Wild, M. Lauer, and M. Reichert. Improving Exception Handling by Discovering Change Dependencies in Adaptive Process Management Systems. In *Proc. 2nd Int'l Workshop on Business Process Intelligence (BPI'06)*, pages 93 – 104, Vienna, 2006.
57. 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.
58. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, 2nd Edition.* Morgan Kaufmann, 2005.