Using Process Mining to Learn from Process Changes in Evolutionary Systems

Christian W. Günther*
Faculty of Technology Management,
Eindhoven University of Technology, The Netherlands
E-mail: c.w.gunther@tue.nl
*Corresponding author

Stefanie Rinderle-Ma
Department of Databases & Information Systems,
University of Ulm, Germany
E-mail: stefanie.rinderle@uni-ulm.de

Manfred Reichert
Faculty of Electrical Engineering, Mathematics and Computer Science,
University of Twente, The Netherlands
E-mail: m.u.reichert@ewi.utwente.nl

Wil M.P. van der Aalst
Faculty of Technology Management,
Eindhoven University of Technology, The Netherlands
E-mail: w.m.p.v.d.aalst@tue.nl

Jan Recker
Faculty of Information Technology,
Queensland University of Technology, Australia
E-mail: 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. 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.

Keywords: Process-aware information systems; process mining; change mining.

Reference to this paper should be made as follows: Günther, C.W., Rinderle-Ma, S., Reichert, M., van der Aalst, W.M.P. and Recker, J. (2007) 'Using Process Mining to Learn from Process Changes in Evolutionary Systems', Int. J. Business Process Integration and Management, Vol. 1, Nos. 1/2/3, pp.111–111.

Biographical notes: Christian W. Günther is a PhD Candidate at the Information Systems group at the Technische Universiteit Eindhoven (TU/e). He received his BSc and MSc in Software Engineering from the Hasso Plattner-Institute in Potsdam (Germany). His research interests include process mining, flexible and unstructured processes, and process analysis.
Stefanie Rinderle-Ma obtained her PhD in Computer Science at the Institute of Databases and Information Systems, University of Ulm (Germany) where she is currently teaching and working on her habilitation. During her postdoc Stefanie stayed at the University of Twente (The Netherlands), the University of Ottawa (Canada), and the Technical University of Eindhoven (The Netherlands) where she worked on several projects on process visualization and modeling as well as on process mining. Her research interests include adaptive process management systems, semantic aspects of process management, and the controlled evolution of organizational structures and access rules.
Manfred Reichert holds a Ph.D. in computer science and is currently Associate Profes-

1 INTRODUCTION

The notion of flexibility has emerged as a pivotal research topic in Business Process Management (BPM) over the last years (Bider, 2005; Reichert and Dadam, 1998; Soffer, 2005). The need for more flexibility, in general, stems from the observation that organizations often face continuous and unprecedented changes in their business environment (Quinn, 1992; Strong and Miller, 1995). To deal with such disturbances and perturbations of business routines, corresponding business 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 (Regev and Wegmann, 2005; Soffer, 2005; Bider, 2005). 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) (Dumas et al., 2005).

1.1 Problem Statement

Recently, many efforts have been undertaken to make PAISs more flexible and several approaches for *adaptive* process management, like ADEPT (Reichert and Dadam, 1998), CBRFlow (Weber et al., 2004) or WASA (Weske, 2001), have emerged in this context (an overview is provided by Rinderle et al. (2004)). The basic idea behind these approaches is to enable users to dynamically *evolve* or *adapt* process schemes 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., process instance and process type level). In particular, ad-hoc changes conducted at the instance level (e.g., to add, delete or move process steps during runtime) allow to adapt single process instances to exceptional or changing situations (Reichert and Dadam, 1998). Usually, such ad-hoc deviations are recorded in *change logs* Rinderle et al. (2006), which results in more meaningful log information when compared to traditional PAISs.

So far, adaptive process management technology has not addressed the fundamental question what we can learn from the additional change log information (e.g., how to derive potential process schema optimizations from a collection of individually adapted process instances (Aalst et al., 2006)). In principle, *process mining* techniques (Aalst et al., 2004) offer promising perspectives for this. However, current 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 three challenges. First, we have to determine which runtime information about ad-hoc deviations has to be logged and how this information should be represented to achieve optimal mining results. Second, we have to develop advanced mining techniques that utilize *change logs* in addition to execution logs. 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 (Reichert and Dadam, 1998) and ProM (Dongen et al., 2005) 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 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. So far, no specific ProM extension has been developed to mine for process changes.

This paper contributes new techniques for mining ad-hoc process changes in adaptive PMSs and discusses the challenges arising in this context. We first describe what constitutes a process change, how respective information can be represented in change 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 ADEPT change logs in ProM, and introduce mining techniques 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. More precisely, a change process comprises *change operations* (as meta process steps) and the causal relations between them. We introduce two different mining approaches 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). The second approach maps change logs to a labeled state transition system, and then constructs a compact Petri Net model from it.

The remainder of this paper is organized as follows: Section 2 provides background information on process mining and adaptive process management, which is needed for the understanding of this paper. 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 this, Section 5 introduces two different approaches for mining changs logs. Section 6 discusses related work and Section 7 concludes with a summary and an outlook.

2 BACKGROUND INFORMATION

This paper is based on the integration of two existing technologies: *process mining* and *adaptive process management*. This section gives background information needed to understand the implications and leverages of their combination.

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.

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.

As indicated, process mining starts with the existence of event logs. The events recorded in such a logs should be ordered (e.g., based on timestamps) and each event should refer to a particular case (i.e., a process instance) and a particular activity. This is the minimal information needed. However, in most event logs more information is present, e.g., 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). In this paper, we assume that event logs are stored in the MXML format (Dongen et al., 2005). *MXML* is an XML-based format for representing and storing event log data, which is supported by process mining tools such as ProM. Using our ProM*import* tool, it is easy to convert data originating from a wide variety of systems to MXML (Günther and Aalst, 2006). For more information about the MXML format we refer to (Dongen et al., 2005) and (Günther and Aalst, 2006).

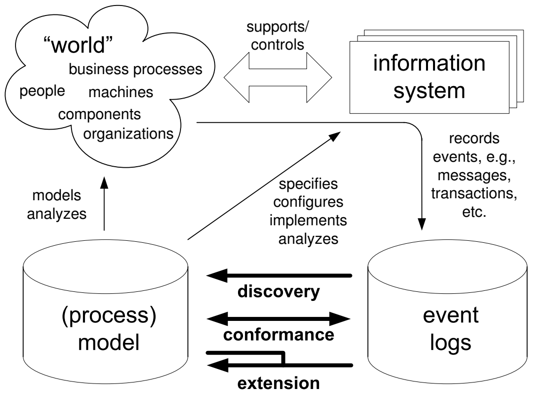


Figure 1: Overview showing three types of process mining: (1) Discovery, (2) Conformance, and (3) Extension.

The idea of process mining is to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs (e.g., in MXML format). 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 (Aalst et al., 2004).

- **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 (Rozinat and Aalst, 2006a).

- **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. Another example is the detection of data dependencies that affect the routing of a case and adding this information to the model in the form of decision rules (Rozinat and Aalst, 2006b).

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 (Dongen et al., 2005). 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

In recent years several approaches for realizing adaptive processes have been proposed and powerful proof-of-concept prototypes have emerged (Weske, 2001; Reichert and Dadam, 1998; Casati et al., 1998; Ellis et al., 1995; Rinderle et al., 2004). Adaptive PMSs like ADEPT2 (Reichert et al., 2005) or WASA (Weske, 2001), for example, provide comprehensive runtime information about process changes not explicitly captured in current execution logs. 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 to deal with an unforeseen or exceptional situation. Process type changes, in turn, refer to the change of a process schema at the type level in order to adapt the PAIS to evolving business processes. Especially for long-running processes, such type changes often require the migration of a collection of running process instances to the new process schema.

PMS frameworks like ADEPT2 (Reichert and Dadam, 1998; Reichert et al., 2005) support both ad-hoc changes of single process instances and the propagation of process type changes to running instances. Examples of *ad-hoc changes* are 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 guarantees achieved by formal checks at build-time 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 is hidden from users. To deal with business process changes ADEPT also enables schema adaptations at the process type level. In particular, it is possible to efficiently and correctly propagate type changes to running instances.

3 A FRAMEWORK FOR INTEGRATION

Both process mining and adaptive processes address fundamental issues 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 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.

The framework depicted in Figure 2 illustrates how such an integration could be realized. 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

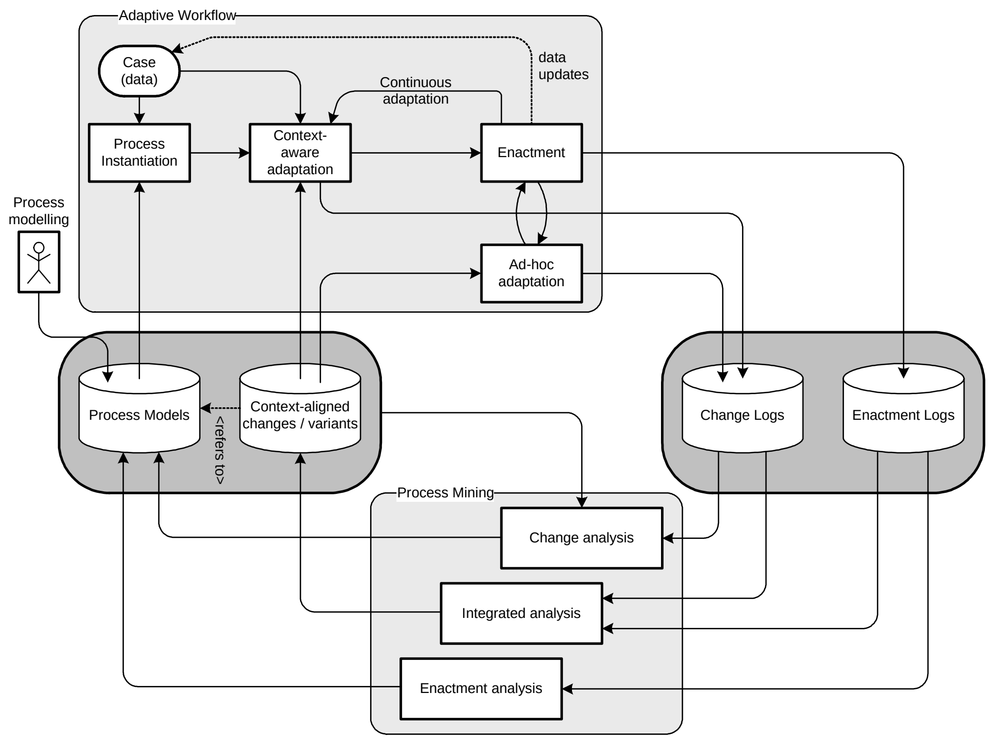


Figure 2: Integration of Process Mining and Adaptive Process Management

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. The progress of autonomous adaptation can thereby be monitored both by administrative staff, and ultimately by the system itself.

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

In this section we provide basic definitions for *process schema*, *schema change*, and *change log*. We assume that a process change will be accomplished by applying a *sequence of change operations to the respective process schema over time* (Reichert and Dadam, 1998). Respective 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 schema. The challenging question is how to represent this change information within change logs. Independent from the applied (high-level) change operations (e.g., adding, deleting or moving activities) we could translate the change into a sequence 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 change analysis significantly. Therefore, change logs should explicitly maintain information about high-level change operations, which combine basic primitives in a certain way.

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 use *transition systems* to define the possible behavior.

Definition 1 (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.*

An 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 3 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 3.

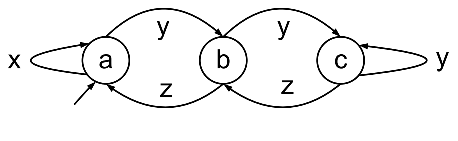


Figure 3: Example transition system

A process schema is defined in terms of a transition system with a designated final state.

Definition 2 (Process Schema)**.** *A process schema is a tuple $PS = (A, s_{start}, s_{end}, TS)$ where*

- *A is a set of activities,*
- *s_{start} is the initial state of the process,*
- *s_{end} is the final state of the process, and*
- *$TS = (S, E, T, s_i)$ is labeled transition system where S is the set of states, $E = A \cup \{\tau\}$ is the set of events (i.e., all activities extended with the so-called "silent step" τ), $T \subseteq S \times E \times S$ is the transition relation, $s_{start} = s_i$ is the initial state, and $s_{end} \in S$ is the final state of the process.*

$\mathcal{P}$ 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}. 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 execution logs. This way we can distinguish between different types of choices (internal/external or controllable/uncontrollable). 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, consisting of a sequence of five activities, is shown in Figure 4.

Note that Figure 4 uses some ad-hoc notation inspired by the ADEPT system. This does not mean that we advocate a particular modeling language. *Any process modeling language having operational semantics can be mapped onto a labeled transition system.* We only assume that such a mapping exists. The choice of a suitable language is a topic by itself. For example, some authors advocate a more goal driven approach (Bider et al., 2002; Soffer and Wand, 2004) while others stress concurrency aspects (Glabbeek and Weijland, 1996; Kiepuszewski, 2002). However, in this paper we abstract from these issues and focus on process changes independent of the language chosen.

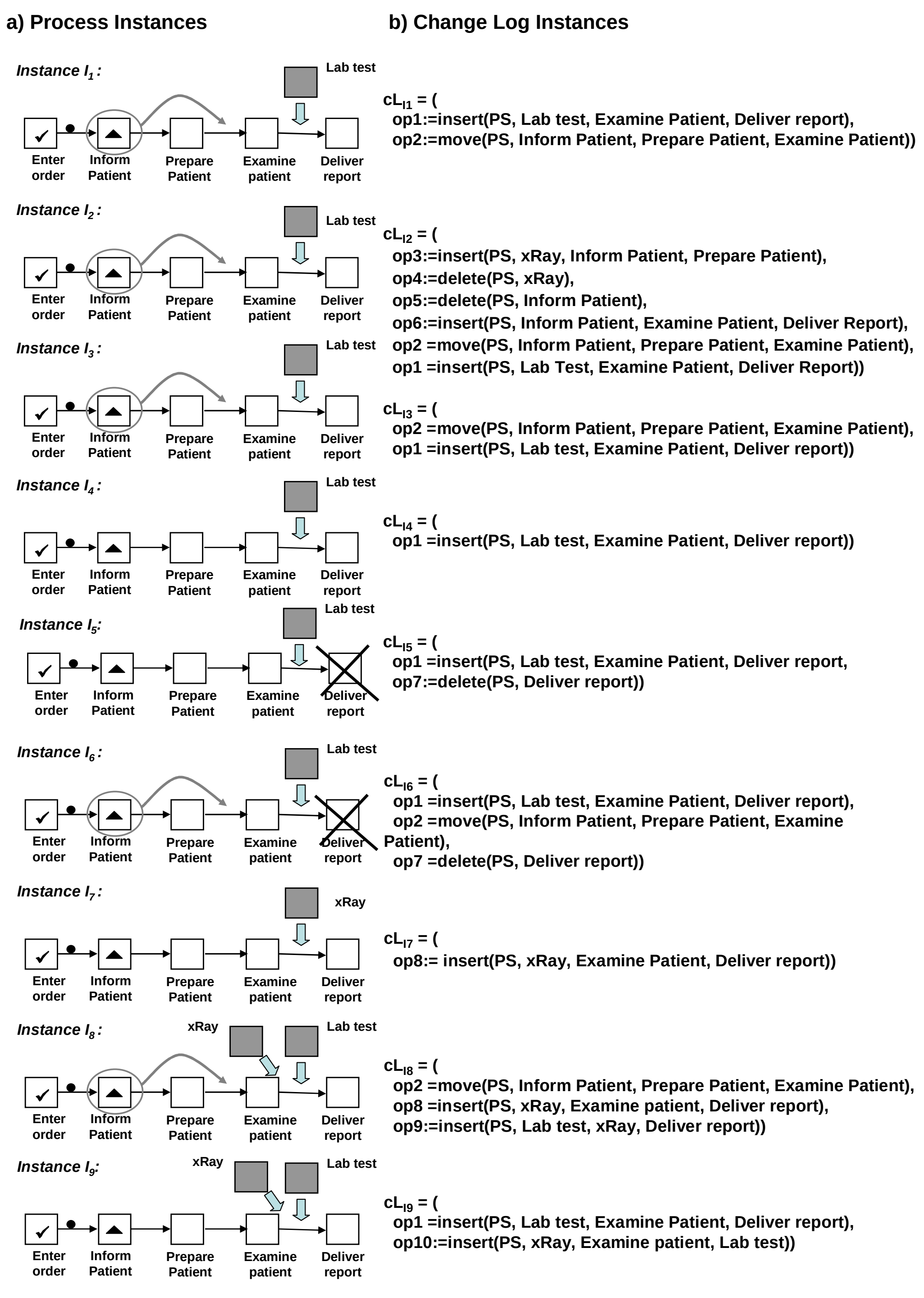
Based on Definition 2, change logs can be defined as follows:

Definition 3 (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 $PS \in \mathcal{P}$, i.e., $\sigma \in \mathcal{C}^*$ (where $\mathcal{C}^*$ is the set of all possible sequences over $\mathcal{C}$).* 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 multiset (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. In this paper we only consider the presence of changes and not their frequency, to simplify matters. Note that in tools like ProM the frequency definitely plays a role and is used to deal with noise and to calculate probabilities.

Figure 4 shows an example of a change log in column *b)*. This log 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 *op*1 and *op*2.

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:



cL_{I1} = (
op1:=insert(PS, Lab test, Examine Patient, Deliver report),
op2:=move(PS, Inform Patient, Prepare Patient, Examine Patient))

cL_{I2} = (
op3:=insert(PS, xRay, Inform Patient, Prepare Patient),
op4:=delete(PS, xRay),
op5:=delete(PS, Inform Patient),
op6:=insert(PS, Inform Patient, Examine Patient, Deliver Report),
op2 =move(PS, Inform Patient, Prepare Patient, Examine Patient),
op1 =insert(PS, Lab Test, Examine Patient, Deliver Report))

cL_{I3} = (
op2 =move(PS, Inform Patient, Prepare Patient, Examine Patient),
op1 =insert(PS, Lab test, Examine Patient, Deliver report))

cL_{I4} = (
op1 =insert(PS, Lab test, Examine Patient, Deliver report))

cL_{I5} = (
op1 =insert(PS, Lab test, Examine Patient, Deliver report,
op7:=delete(PS, Deliver report))

cL_{I6} = (
op1 =insert(PS, Lab test, Examine Patient, Deliver report),
op2 =move(PS, Inform Patient, Prepare Patient, Examine
Patient),
op7 =delete(PS, Deliver report))

cL_{I7} = (
op8:= insert(PS, xRay, Examine Patient, Deliver report))

cL_{I8} = (
op2 =move(PS, Inform Patient, Prepare Patient, Examine Patient),
op8 =insert(PS, xRay, Examine patient, Deliver report),
op9:=insert(PS, Lab test, xRay, Deliver report))

cL_{I9} = (
op1 =insert(PS, Lab test, Examine Patient, Deliver report),
op10:=insert(PS, xRay, Examine patient, Lab test))

Figure 4: Modified Process Instances and Associated Change Log Instances

Definition 4 (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 4 . 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 5 (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 4 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.

As mentioned, in this paper we represent change log entries by means of high-level change operations since we want to exploit their semantical content (see Figure 4 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 **?** for details) they are generic in the sense that they can be easily transferred to other meta models as well (e.g. (Reichert et al., 2003)).

5 CHANGE MINING

In this section we describe novel approaches for analyzing change log information, as found in adaptive PMSs. First, we describe how change logs can be mapped onto the MXML format used for process mining. This makes it possible to evaluate the application of traditional process mining algorithms to change logs. Subsequently, 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 5.4. This commutativity relation provides the foundation for our first mining algorithm for change processes, as introduced in Section 5.5. A second algorithm for mining change processes based on the theory of regions is presented in Section 5.6, and Section 5.7 compares both approaches. Finally, Section 5.8 sketches how context information may be used to investigate the drivers for change (i.e., *why* changes occur) in future work.

5.1 Mapping Change Logs to MXML

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 4.

```
<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*, i.e., adjacent process schema elements that are either directly preceding or following the change subject in the process definition. As these specific properties are not covered by the MXML format, they are stored as attributes in the “Data” field. The set of attributes for a change event is completed by an optional *rationale* field, storing a human-readable reason,

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)			

or incentive, for this particular change.

The *originator* field is used for the person having applied the respective change, while the *timestamp* field 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, uniquely identifying the class of change, is used.

Once the change log information is mapped and converted to MXML, it can be mined by any process mining algorithm, e.g. 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 subsection, mapping process change logs to the existing MXML format for execution logs enables the use of existing mining algorithms (e.g., as implemented within the ProM framework) for mining change logs as well. In the following we discuss how "well" these algorithms perform when being applied to change logs. The underlying evaluation has been carried out using an extension of the ADEPT demonstrator (Waimer, 2006).

For evaluation purposes, the change processes generated by the different mining algorithms are compared along selected quality criteria. The most important criterion is how "well" a change process reflects the actual dependencies between the operations contained within the input change log. As for process instance I_2, for example, change operation op_4 depends on previous change operation op_3 (cf. Figure 4). This dependency should be reflected as a sequence $op_3 \longrightarrow op_4$ within the resulting change process. Contrary, independent change operations should be ordered in parallel within the resulting change process.

In our evaluation we analyzed the α Algorithm, the Multi-Phase Miner, and the Heuristics Miner (Waimer, 2006). All of these algorithms reflect the actual dependencies between the change operations quite "well" for simple processes and a restricted set of change operations. The quality of the mined change processes decreases rapidly (i.e., dependencies are generated by the mining algorithms which are actually not existing and the change processes become less and less meaningful) if different change operations are applied and the underlying processes become more complex. The fundamental problem is that process changes tend to be rather infrequent, i.e., compared to regular logs there are relatively few cases to learn from. 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.

5.3 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-set*, 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[1], 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 state of the mined process is treated like a "black box". This is necessary because enactment logs 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 in fact 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.

5.4 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 5.3). 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 discussed in Section 4 (cf. Definition 4), 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.

[1]In this paper we assume that the *subject* field is limited to one activity.

Definition 6 (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* (Kiepuszewski, 2002; Rinderle et al., 2004), 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 (Glabbeek and Weijland, 1996). 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 7 (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 7 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).

5.5 *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 (Dongen and Aalst, 2004) has been selected, which is very robust in handling ambiguous 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 (Dongen and Aalst, 2004). 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 (Dongen and Aalst, 2004) 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 4, 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 (Desel et al., 2004) of the change process mined from the example change log instances in Figure 4. 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 4, 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. An example for this is inserting "xRay" and inserting "Lab Test", which appear in this order in instance I_8, and in reverse order in instance I_9. As a result, there is no causal relation, and thus no direct link between these change operations in the model shown in 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 7). 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.

5.6 *Approach 2:* Mining Change Processes with Regions

The second approach towards mining change logs uses an approach based on the *theory of regions* (Cortadella et al., 1998) 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.

In Definition 2 we already used the concept of a transition system to describe the behavioral aspect of a process

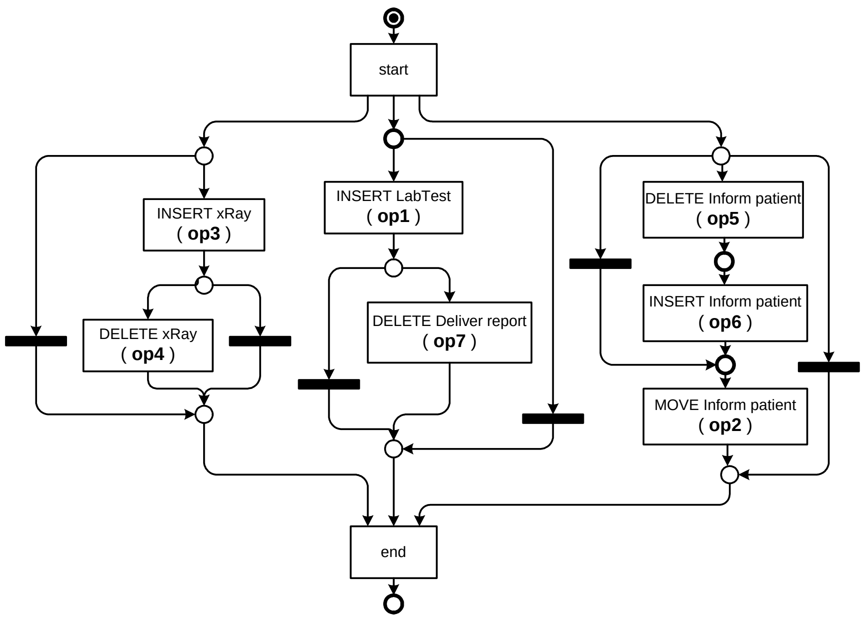

start
INSERT xRay (op3)
INSERT LabTest (op1)
DELETE Inform patient (op5)
DELETE xRay (op4)
DELETE Deliver report (op7)
INSERT Inform patient (op6)
MOVE Inform patient (op2)
end

Figure 5: Mined Example Process (Petri net notation)

schema. However, now we use it as an intermediate format for representing change processes. As indicated in Section 4 we do not advocate transition systems as an end-user language. Any modeling language having formal semantics can be mapped onto a transition system. The reverse is less obvious, but quite essential for our approach. Therefore, we first explain the "theory of regions" (Cortadella et al., 1998; Ehrenfeucht and Rozenberg, 1989) which allows us to translate a transition system into a graphical process model.

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* (Cortadella et al., 1998; Ehrenfeucht and Rozenberg, 1989). An example of a tool that can create a Petri net for any transition system using regions is *Petrify* (Cortadella et al., 1997).

The theory of regions takes a transition system and converts it into an equivalent Petri net. In this paper we do not elaborate on the theory 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. All the events that *enter* a particular region are the transitions producing tokens for the corresponding place and all the events that *leave* the region are the transitions consuming tokens from this place. In the original theory of regions (Ehrenfeucht and Rozenberg, 1989) many simplifying assumptions are made and it was impossible to have multiple Petri net transitions with the same label and silent steps could no be handled. However, tools such as Petrify (Cortadella et al., 1997) based on the approach described in (Cortadella et al., 1997, 1998) can deal with any transition system.

To illustrate the idea consider the transition system shown in Figure 6. Using regions we obtain the Petri net shown in Figure 7. This Petri net was obtained automatically using a combination of ProM and Petrify. Clearly this Petri net reveals the behavior implied by Figure 6 in a compact and readable manner. This example shows the potential of applying logs to transition systems with more parallelism. Note that although the process in Figure 7 is represented in terms of a Petri net, the idea is not limited to Petri nets. Using ProM we can easily map the model onto another notation e.g. EPCs, BPMN, YAWL, BPEL, etc. Note that all of this functionality has been realized in ProM.

Figures 6 and 7 illustrate the idea of folding a transition system into a process model like e.g. a Petri net. Therefore, the challenge of mining changes process is reduced to the construction of a transition system based on a change log. To do this we first introduce some useful notations.

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

- *$\sigma(k) = \Delta_k$ is the k^{th} element of σ $(1 \leq k \leq n)$,*
- *$hd(\sigma, k) = \langle \Delta_1, \Delta_2, \dots \Delta_k \rangle$ is the sequence of the first k elements $(0 \leq k \leq n)$ of σ (with $hd(\sigma, 0) = \langle\ \rangle$),*

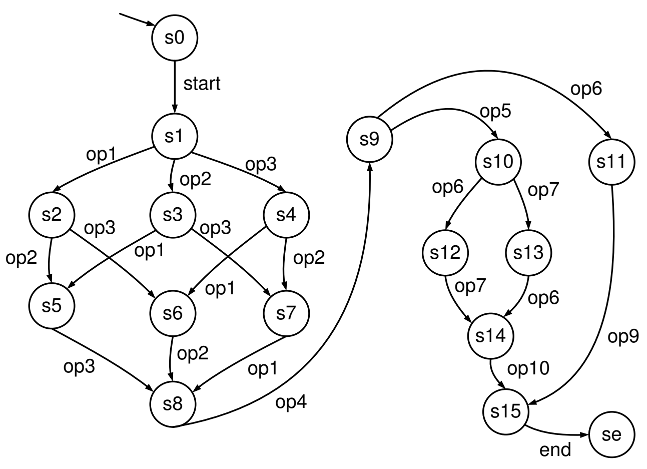


Figure 6: A transition system with more parallelism

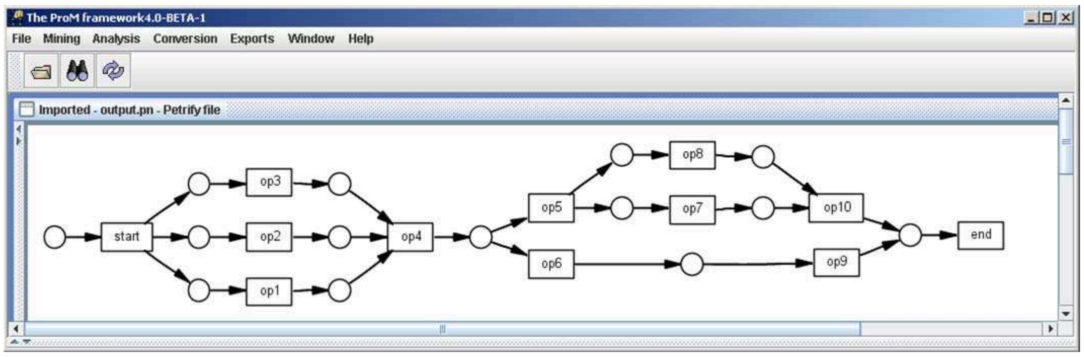

The ProM framework4.0-BETA-1
File Mining Analysis Conversion Exports Window Help
Imported - output.pn - Petrify file
start
op1
op2
op3
op4
op5
op6
op7
op8
op9
op10
end

Figure 7: Screenshot of ProM showing the Petri net obtained for the transition system depicted in Figure 6

- *$state(PS, \sigma, k) = PS'$ 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, $T = \{(state(PS, \sigma, k), \sigma(k+1), state(PS, \sigma, k+1)) \mid \sigma \in L \ \wedge \ 0 \leq k < |\sigma|\}$ 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 matters 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* $s^*_i = s_0$*.*

Figure 8 shows the transition system obtained by applying Definition 10 to the running example, i.e., the change log depicted in Figure 4 (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 4 defines ten different change operations, these correspond to the events in the transition system. Moreover, the artificial start and end are added. Hence $E = \{start, op1, op2, \ldots, op10, end\}$ is the set of events. The application of a change operation to some process schema, i.e., a state in Figure 8, results in a new state. Since in this particular example all process schemas happen to be sequential, we can denote them by a simple sequence as shown in Figure 8. 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 8 is obtained.

After constructing the transition system, the theory of regions can be used to construct a process model. Figure 9 shows the Petri net constructed using this approach. Note that using a combination of ProM and Petrify the log is coverted to the transition system of Figure 8 which is then folded into the Petri net depicted in Figure 9. In this case 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 9. However, one should note that the change log depicted in Figure 4 only has nine change log instances, i.e., compared to the number of change operations, the number of instances is rather low. It seems that only a few of the possible interleavings are present in Figure 4. As a result, the transition system in Figure 9 seems to be the result of a number of *particular* examples rather than a description of the full behavior. Figures 6 and 7 show that using the theory of regions it is possible to dramatically reduce the size of the model if more interleavings are present.

5.7 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 (Dongen and Aalst, 2004). 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 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.

In this section, we applied the two approaches to an example log. This allows us to compare both. The Petri net in Figure 9 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.

5.8 Towards Learning about the Context of Change

Understanding how process change information can be represented in logs and how these logs can be mined to deliver valuable insights into the scope of change delivers insights of *how* processes deviate from predefined routines. This is a significant move towards understanding *why* such changes occur, viz., the drivers for change). These drivers can be found in the *context* of a process (Rosemann et al., 2006).

In general terms, the context of a business process is made up by all the relevant information that is available at some stage during the execution of a business process, and that could potentially have influenced decisions in this process. It can be seen as the set of process data and information that is relevant to the process execution but typically not defined in the process definition itself, which, following existing classification schemes (Jablonski and Bussler, 1996), would at least include the control flow logic, involved informational data, and organizational resources. Context information can be retrieved from a wide range of potential data sources. Enactment logs, for instance, often include information about time and value of a data mod-

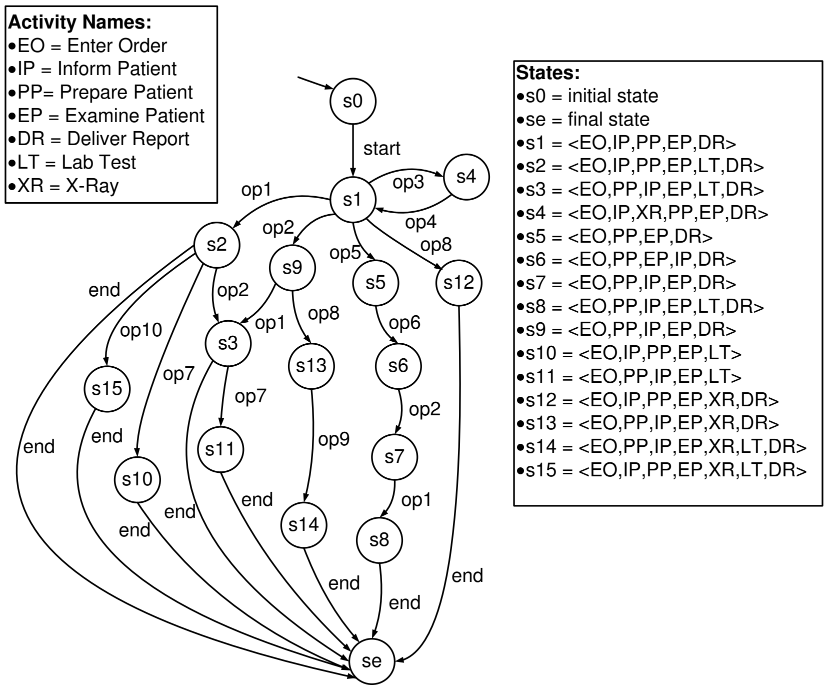
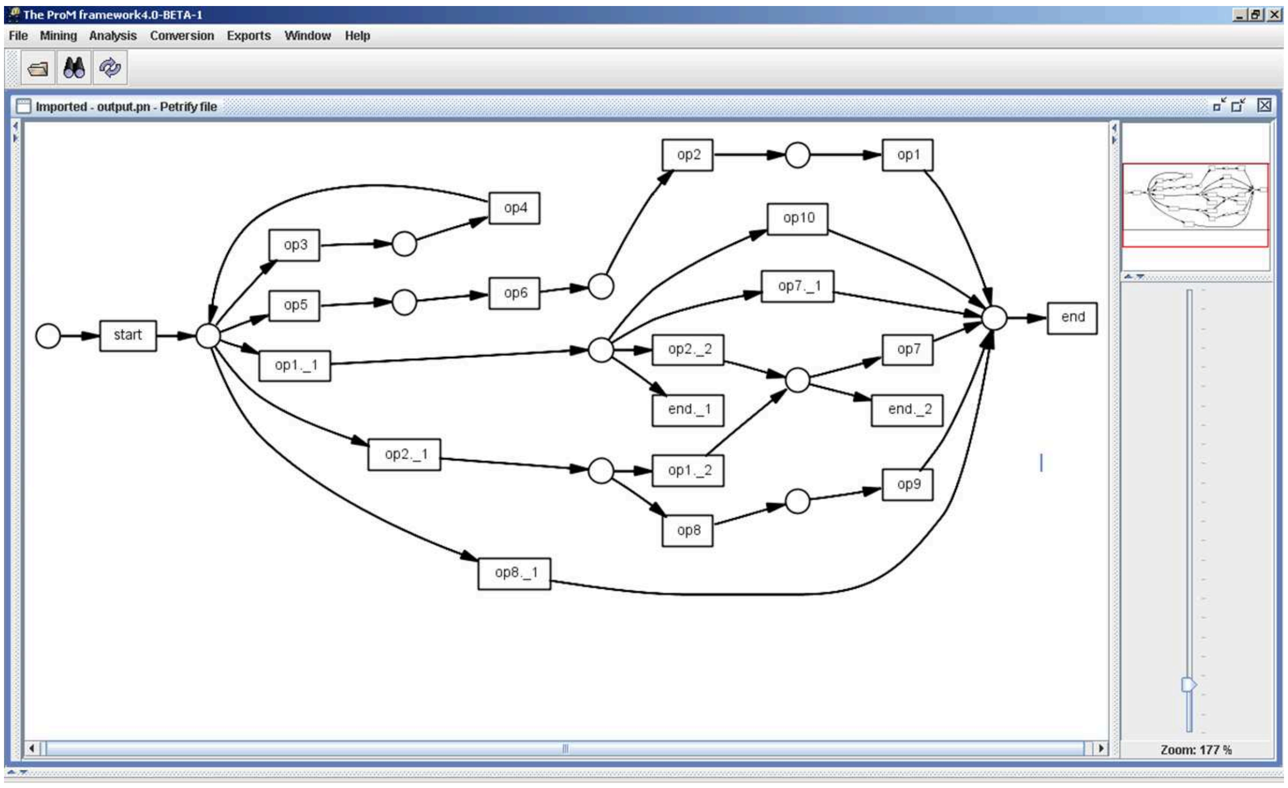


Figure 8: Transition system based on the change log shown in Figure 4



The ProM framework4.0-BETA-1
File Mining Analysis Conversion Exports Window Help
Imported - output.pn - Petrify file
start
op3
op4
op5
op6
op1._1
op2._1
op8._1
op2
op1
op10
op7._1
op2._2
end._1
op7
end._2
op1._2
op8
op9
end
Zoom: 177 %

Figure 9: Screenshot of ProM showing the Petri net obtained for the change log depicted in Figure 4

ification. As context information is obviously most useful when timed, a promising means of storage would be to enhance change logs with context data. This would allow to structure the context history in suitable chunks, i.e., the structural states of a process between change operations. Technically, this can be accomplished by examining each change event, acquiring timed context information for the time of its occurrence, and then enriching it with elicited context information.

Assuming access to context information, changes in a process could be investigated *together* with the reasons for the change decisions taken along the execution of a process. This can be achieved by looking at change process models and the decision points contained within. However, while these change process models themselves are already helpful in developing an understanding of the drivers for change, they can not be used to actually *learn* from the change. Learning can be interpreting as deriving information from an adaptive PMS. The fundamental premise is that cases in which a certain change has been applied will exhibit distinct *patterns* in their context information. As the set of potential context information can be very large, identifying the pivotal data elements, or patterns thereof, which are unique for a specific change, somehow resembles looking for a needle in a haystack. Fortunately, existing Machine Learning (ML) (Mitchell, 1997) techniques can solve this problem. *Classification algorithms*, for instance, 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. Training a *decision tree* algorithm (Rozinat and Aalst, 2006b) with such a classified set may then provide decision trees that visualize how decisions about process change were being made. Other classification algorithms from ML can generate a set of classification rules (Mitchell, 1997).

These classification algorithms by definition focus on specific decisions, i.e., one branching point in the process, and are thus dependent on the mining of a change process model in the first place. An alternative to this approach is the *mining of association rules* (Agrawal et al., 1993). 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, 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. This can be beneficial especially when there are hardly any causal relations between change operations. Association rules may discover tacit relationships between change operations and context data that could not be captured by classification.

In summation, the application of ML techniques appears promising for the identification of the drivers for change from the context of a process, and for relating them to one another. We believe that this structured approach can deliver precise results while still remaining feasible in practical settings, and can thus a be foundation for the future design of self-adapting PMSs.

5.9 Implementation and Tool Support

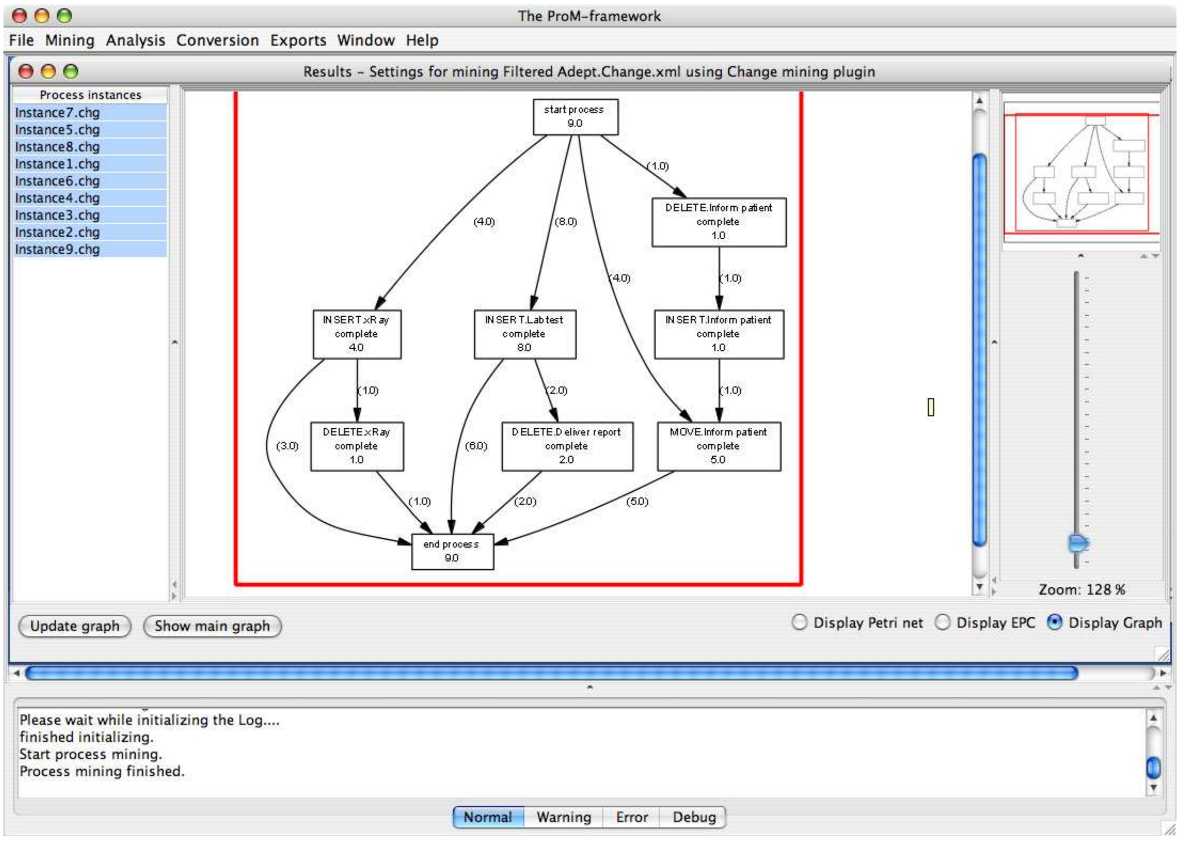
We have implemented a complete toolkit which allows for creating and experimenting with change logs. Process schemas can be designed, executed, and changed in the ADEPT demonstrator prototype (Reichert et al., 2005). The resulting change and enactment logs can then be extracted with a custom plug-in for ProM*import* (Günther and Aalst, 2006). ProM*import* is an open source framework greatly facilitating the implementation of log conversions to MXML for various systems, and can be downloaded from `http://promimport.sf.net/`. Change logs extracted from ADEPT, or any other adaptive system for that matter, can then be loaded into ProM, an open source framework for process mining techniques, which is available at `http://prom.sourceforge.net/`. Our change mining approaches, described in sections 5.5 and 5.6, have both been implemented as plug-ins for ProM. Figure 10 shows the plug-in implementing the approach based on commutativity, introduced in Section 5.5. It displays the example process introduced in Figure 4 in terms of a process graph.

Process mining algorithms have shown to scale well with the amount of input data. The majority of these algorithms, including the two approaches presented in this paper, work in two distinct phases. The first phase scales linearly with the number of events contained in the log. The second phase is of polynomial complexity, where the number of activities in the process corresponds to the problem size. These scalability characteristics make process mining applicable to most real-life problems.

6 RELATED WORK

Although process mining techniques have been intensively studied in recent years (Aalst et al., 2004; Agrawal et al., 1998; Cook and Wolf, 1998; Dongen and Aalst, 2004), 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 (Cortadella et al., 1997, 1998) has also been exploited to mine process schemas from execution logs (Kindler et al., 2006), 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 (Aalst et al., 2005), actor assignments, and decision mining (Rozinat and Aalst, 2006b)).

In recent years, several approaches for adaptive process management have emerged (Rinderle et al., 2004), most of them supporting changes of certain process aspects and changes at different levels. Examples of adaptive PMSs include ADEPT (Reichert et al., 2005), CBRflow (Weber et al., 2004), and WASA (Weske, 2001). 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



Results – Settings for mining Filtered Adept.Change.xml using Change mining plugin

Process instances
Instance7.chg
Instance5.chg
Instance8.chg
Instance1.chg
Instance6.chg
Instance4.chg
Instance3.chg
Instance2.chg
Instance9.chg

Display Petri net Display EPC [✓ Display Graph]

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

Figure 10: Change Mining Plug-in within ProM.

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 (Weber et al., 2004) 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. 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 (Weber et al., 2004). 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 (Aha and Munoz-Avila, 2001). 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. Later, Weber et al. (2006) further improved the support for change reuse by additionally discovering dependencies between different ad-hoc changes. In (Rinderle et al., 2005; Weber et al., 2005) 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 Rinderle et al. (2005) particularly make 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.

7 SUMMARY AND OUTLOOK

This paper gave an overview of how comprehensive support for true process flexibility can be provided by combining adaptive PMSs with advanced process mining techniques. The integration of process mining with adaptive PMS enables the exploitation of knowledge about process changes from change logs.

We have developed two mining techniques and implemented them as plug-ins for the ProM framework, taking ADEPT change logs in the mapped MXML format as input. Based on this we have sketched how to discover a (minimal) change process which captures all modifications applied to a particular process. This discovery is based on the analysis of (temporal) dependencies between change operations that have been applied to a 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 (Rozinat and Aalst, 2006b)) 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.

ACKNOWLEDGMENT

This research has been supported by EIT, NWO, SUPER, and the Technology Foundation STW, applied science division of NWO and the technology program of the Dutch Ministry of Economic Affairs. The contributions of Jan Recker and Wil van der Aalst were partly supported by the Australian Research Council with the ARC Discovery Grant "Next Generation Reference Process Models" (DP0665480).

REFERENCES

Aalst, W., Günther, C., Recker, J., and Reichert, M. (2006). Using Process Mining to Analyze and Improve Process Flexibility (Position Paper). In Latour, T. and Petit, M., editors, *The 18th International Conference on Advanced Information Systems Engineering. Proceedings of Workshops and Doctoral Consortium*, pages 168–177. Namur University Press.

Aalst, W., Reijers, H., and Song, M. (2005). Discovering Social Networks from Event Logs. *Computer Supported Cooperative work*, 14(6):549–593.

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

Agrawal, R., Gunopulos, D., and Leymann, F. (1998). Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483.

Agrawal, R., Imielinski, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216.

Aha, D. and Munoz-Avila, H. (2001). Introduction: Interactive case-based reasoning. *Applied Intelligence*, 14(7-8).

Bider, I. (2005). Masking flexibility behind rigidity: Notes on how much flexibility people are willing to cope with. In Castro, J. and Teniente, E., editors, *CAiSE'05 Workshops*, pages 7–18, Porto, Portugal. FEUP.

Bider, I., Johannesson, P., and Perjons, E. (2002). Goal-Oriented Patterns for Business Processes. In *Proceedings of the HCI Workshop on Goal-Oriented Business Process Modeling (GBMP'02)*, volume 109 of *CEUR Workshop Proceedings*, pages 1–5. CEUR-WS.org.

Casati, F., Ceri, S., Pernici, B., and Pozzi, G. (1998). Workflow Evolution. *Data and Knowledge Engineering*, 24(3):211–238.

Cook, J. and Wolf, A. (1998). Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249.

Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., and Yakovlev, A. (1997). Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D(3):315–325.

Cortadella, J., Kishinevsky, M., Lavagno, L., and Yakovlev, A. (1998). Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 47(8):859–882.

Desel, J., Reisig, W., and Rozenberg, G., editors (2004). *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin.

Dongen, B. and Aalst, W. (2004). Multi-Phase Process Mining: Building Instance Graphs. In Atzeni, P., Chu, W., Lu, H., Zhou, S., and Ling, T., editors, *International Conference on Conceptual Modeling (ER 2004)*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, Berlin.

Dongen, B., Medeiros, A., Verbeek, H., Weijters, A., and Aalst, W. (2005). The ProM framework: A new era in process mining tool support. In Ciardo, G. and Darondeau, P., 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.

Dumas, M., Aalst, W., and Hofstede, A. (2005). *Process-Aware Information Systems: Bridging People and Software through Process Technology.* Wiley & Sons.

Ehrenfeucht, A. and Rozenberg, G. (1989). Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Informatica*, 27(4):315–368.

Ellis, C., Keddara, K., and Rozenberg, G. (1995). Dynamic change within workflow systems. In Comstock, N., Ellis, C., Kling, R., Mylopoulos, J., and Kaplan, S., editors, *Proceedings of the Conference on Organizational Computing Systems*, pages 10 – 21, Milpitas, California. ACM SIGOIS, ACM Press, New York.

Glabbeek, R. and Weijland, W. (1996). Branching Time and Abstraction in Bisimulation Semantics. *Journal of the ACM*, 43(3):555–600.

Günther, C. and Aalst, W. (2006). A generic import framework for process event logs. In Eder, J. and Dustdar, S., editors, *Business Process Management Workshops, Workshop on Business Process Intelligence (BPI 2006)*, volume 4103, pages 81–92. Springer Verlag, Berlin.

Jablonski, S. and Bussler, C. (1996). *Workflow Management. Modeling Concepts, Architecture, and Implementation.* Thomson Computer Press, London, UK.

Kiepuszewski, B. (2002). *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows.* PhD thesis, Queensland University of Technology, Brisbane. (available via *http://www.workflowpatterns.com/).*

Kindler, E., Rubin, V., and Schäfer, W. (2006). Process mining and petri net synthesis. In Eder, J. and Dustdar, S., editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 105–116. Springer Verlag, Vienna, Austria.

Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill.

Quinn, J. B. (1992). *Intelligent Enterprise: A Knowledge and Service Based Paradigm for Industry.* Free Press, New York, NY.

Regev, G. and Wegmann, A. (2005). A regulation-based view on business process and supporting system flexibility. In Castro, J. and Teniente, E., editors, *Proceedings of the CAiSE'05 Workshops. Vol. 1*, pages 91–98. FEUP, Porto, Portugal.

Reichert, M. and Dadam, P. (1998). ADEPTflex - Supporting Dynamic Changes of Workflows Without Loosing Control. *Journal of Intelligent Information Systems*, 10(2):93–129.

Reichert, M., Rinderle, S., and Dadam, P. (2003). 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.

Reichert, M., Rinderle, S., Kreher, U., and Dadam, P. (2005). Adaptive process management with ADEPT2. In *Proc. 21st Int'l Conf. on Data Engineering (ICDE'05)*, pages 1113–1114, Tokyo.

Rinderle, S., Reichert, M., and Dadam, P. (2004). 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.

Rinderle, S., Reichert, M., Jurisch, M., and Kreher, U. (2006). On Representing, Purging, and Utilizing Change Logs in Process Management Systems. In *Proc. Int'l Conf. on Business Process Management (BPM'06)*, Vienna.

Rinderle, S., Weber, B., Reichert, M., and Wild, W. (2005). 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.

Rosemann, M., Recker, J., Ansell, P., and Flender, C. (2006). Context-awareness in business process design. In Koronios, A. and Fitzgerald, E., editors, *Australasian Conference on Information Systems*, Adelaide, Australia. Australasian Chapter of the Association for Information Systems.

Rozinat, A. and Aalst, W. (2006a). Conformance Testing: Measuring the Fit and Appropriateness of Event Logs and Process Models. In Bussler et al., C., editor, *BPM 2005 Workshops (Workshop on Business Process Intelligence)*, volume 3812 of *Lecture Notes in Computer Science*, pages 163–176. Springer-Verlag, Berlin.

Rozinat, A. and Aalst, W. (2006b). Decision Mining in ProM. In Dustdar, S., Faideiro, J., and Sheth, A., editors, *International Conference on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 420–425. Springer-Verlag, Berlin.

Soffer, P. (2005). On the Notion of Flexibility in Business Processes. In Castro, J. and Teniente, E., editors, *Proceedings of the CAiSE'05 Workshops. Vol. 1*, pages 35–42. FEUP, Porto, Portugal.

Soffer, P. and Wand, Y. (2004). Goal-Driven Analysis of Process Model Validity. In Persson, A. and Stirna, J., editors, *Advanced Information Systems Engineering, Proceedings of the 16th International Conference on Advanced Information Systems Engineering (CAiSE'04)*, volume 3084 of *Lecture Notes in Computer Science*, pages 521–535. Springer-Verlag, Berlin.

Strong, D. and Miller, S. (1995). Exceptions and exception handling in computerized information processes. *ACM Transactions on Information Systems*, 13(2):206–233.

Waimer, M. (2006). Integration of adaptive process management technology and process mining. (in German).

Weber, B., Rinderle, S., Wild, W., and Reichert, M. (2005). CCBR-Driven Business Process Evolution. In *Proc. Int. Conf. on Cased based Reasoning (ICCBR'05)*, Chicago.

Weber, B., Wild, W., and Breu, R. (2004). CBRFlow: Enabling adaptive workflow management through conversational case-based reasoning. In *Proc. Eurpean Conf. on Case–based Reasoning (ECCBR'04)*, pages 434–448, Madrid.

Weber, B., Wild, W., Lauer, M., and Reichert, M. (2006). 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.

Weske, M. (2001). Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In *Proc. Hawaii International Conference on System Sciences (HICSS-34).*