Mining Processes with Multi-Instantiation: Discovery and Conformance Checking

Ingo Weber[1,2] Mostafa Farshchi[1,3]
Jan Mendling[4] Jean-Guy Schneider[3]

[1] Software Systems Research Group, NICTA, Sydney, Australia
Ingo.Weber@nicta.com.au
[2] University of New South Wales, Sydney, Australia
[3] Swinburne University of Technology, Hawthorn, Australia
{mfarshchi,jschneider}@swin.edu.au
[4] Wirtschaftsuniversität Wien, Vienna, Austria
jan.mendling@wu.ac.at

Technical Report
UNSW-CSE-TR-201420
September 2014

Abstract

Process mining is becoming a widely adopted practice. However, when the underlying process contains multi-instantiation of sub-processes, classical process mining techniques that assume a flat process are not directly applicable. Their application can cause one of two problems: either the mined model is overly general, allowing arbitrary order and execution frequency of activities in the sub-process, or it lacks fitness by capturing only single instantiation of sub-processes. For conformance checking, this results in a too high rate of either false positives or false negatives, respectively. In this report, we propose an extension to well-known process mining techniques, adding the capability of handling multi-instantiated sub-processes to discovery and conformance checking. We evaluate the approach with two independent data sets taken from real-world applications.

1 Introduction

Process mining, the act of mining event logs and deriving process artifacts from them or comparing them with process artifacts, is a very active area of research and is becoming a widely adopted practice [24]. However, when the underlying process contains multi-instantiation (MI) in sub-processes, it is not possible to directly apply classical process mining techniques. Multi-instantiation of sub-processes refers to several instances of a sub-process being executed concurrently. The well-known workflow patterns for control flow distinguish several types of MI, depending on whether the number of concurrent sub-processes is known at design time, at runtime when entering the MI sub-process part, or completely unknown a priori [25], emphasizing their importance for process models.

Multi-instantiation can be present for various reasons. For instance, a request for quotations (RFQ) may be sent to a list of potential suppliers concurrently. Alternatively, as observed in our own work [28], cloud management processes may touch on a number of cloud resources in parallel – for example, when upgrading application software on a large number of virtual machines. Conceptually, this problem relates to n:m relationships between resources, which is recently discussed in the context of artifact-centric process mining [4, 19]. In general, mining such processes entails one of two problems for MI: either the mined model is overly general, allowing arbitrary order and execution frequency of activities in the sub-process, or it lacks fitness by capturing only a single instantiation of the sub-process. For conformance checking, this results in a too high rate of either false positives or false negatives, respectively.

In this report, we propose an extension to well-known process mining techniques, adding the capability of handling multi-instantiated sub-processes to discovery and conformance checking, respectively. To this end, we define strategies for extracting sub-process identifiers (IDs) and making those explicit in a preprocessing step to the actual process mining techniques. Having sub-process IDs allows us to hierarchically discover process models with multi-instantiation of sub-processes. Similarly, conformance checking can be done on this basis. However, the extraction of sub-process IDs requires domain expertise, and so does the encoding of extraction rules; thus, at present, they remain semi-automated steps. We have evaluated the approach with two independent data sets taken from real-world applications, namely cloud resource management and student examination logs. The evaluation reveals the benefits in terms of precision and fitness against the baseline of standard process mining algorithms.

The remainder of this report is organized as follows. The problem is showcased in Section 2. An approach and a formal model are given in Section 3. The implementation and evaluation are discussed in Section 4. Section 5 contrasts the work against related work, and Section 6 summarizes the main contributions and provides an outlook on future work.

2 Motivating Example

In separate work [27, 28], we have analyzed cloud management operations. Here, we considered the "rolling upgrade" procedure as implemented by the open-source tool Netflix Asgard, a management tool for cloud resources on Amazon Web Services (AWS). A rolling upgrade replaces virtual machines (VMs) on

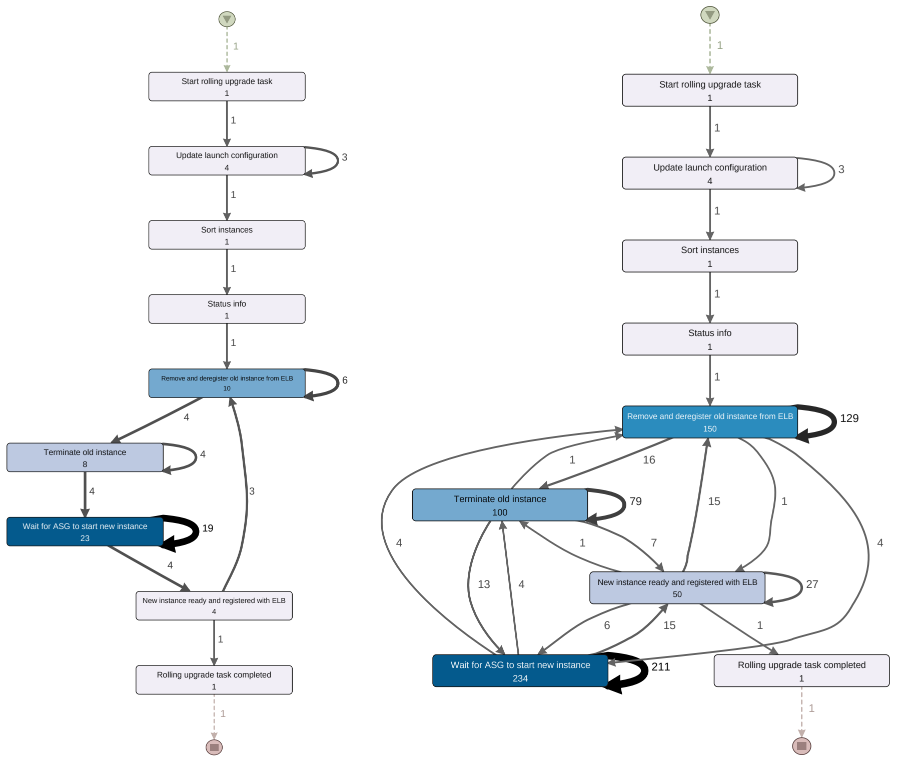

Start rolling upgrade task
Update launch configuration
Sort instances
Status info
Remove and deregister old instance from ELB
Terminate old instance
Wait for ASG to start new instance
New instance ready and registered with ELB
Rolling upgrade task completed
Start rolling upgrade task
Update launch configuration
Sort instances
Status info
Remove and deregister old instance from ELB
Terminate old instance
New instance ready and registered with ELB
Wait for ASG to start new instance
Rolling upgrade task completed

Figure 2.1: Mined models from rolling upgrade (RU). Left: 4 VMs, 1 at a time, referred to as *RU Simple* model from here on. Right: 50 VMs, 5 at a time, without treatment of multi-instantiated sub-processes, called *RU Chaos* model from here on.

AWS, x at a time. Let us assume that an application is running on a total of $n = 20$ VMs, and that these VMs were created from an Amazon Machine Image (AMI), a template that contains all application logic at version k. Adding a new VM to the application's cluster can then be done by starting a new VM from said AMI. This approach is referred to as using *heavily-baked images*, that is, for any change – no matter how small – a new AMI is "baked". Finally this new AMI, referred to as version $k + 1$, is rolled out by replacing all running VMs.

However, in order to maintain full availability of services to users, not all VMs are replaced at once. Instead, the n VMs running version k are replaced gradually. This is done by taking x of the n VMs running version k out of service, replacing them with x VMs running version $k+1$. Once the x new VMs with version $k+1$ are ready to accept requests, the procedure is repeated. This is done until all n VMs have been replaced.

In our previous work [27, 28], we examined the logs produced by Asgard's rolling upgrade procedure, (i) to understand the how the process is implemented, and (ii) to detect deviations from the desired process at runtime, for the purpose of early error detection.

The left-hand side of Fig. 2.1 shows a mined process model using the Fuzzy

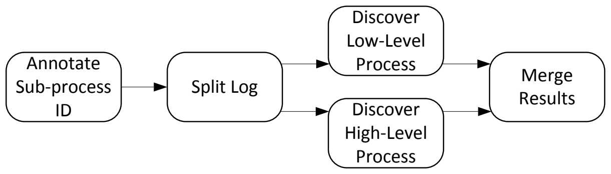


Figure 3.1: Overview of the multi-instantiation approach

Miner as implemented in Fluxicon Disco.[1] The first four activities prepare the upgrade. The next four activities form a loop, upgrading 4 VMs ($n = 4$), one at a time ($k = 1$). After the 4th iteration, the process completes. The individual steps to pre-process the log of Asgard so that it can be fed to Disco are described in detail in [27]. The right-hand side of Fig. 2.1 shows an analogously mined model, however, in this case from logs where 50 VMs have been upgraded, 5 at a time. As can be seen, the mined process on the right hardly captures the sequencing at the level of individual instances, but creates a highly-connected model instead.

Our work on cloud resource management includes online conformance checking [28]. The model on the left-hand side of Fig. 2.1 can be used to that end, in cases where $k = 1$. If $k > 1$, the false positive rate is high and many events will be deemed unfit. Thus, the *fitness* of this model is low. In contrast, the model on the right-hand side will cover almost arbitrary execution traces. It is thus underfitting the log, and lacks *precision*. Such trade-offs have been acknowledged before for process mining [24]. The aim of this report is to find a better balance in case of multiple instances.

3 Handling Multi-Instantiation in Process Mining

In this section we define an approach for handling multiple instances in process mining. Section 3.1 gives an overview and Section 3.2 provides formal definitions. Sections 3.3 to 3.6 define the individual steps of our approach, i.e., sub-process ID extraction, trace splitting, model merging, and conformance checking.

3.1 Approach Overview

Fig. 3.1 shows the steps taken to discover hierarchical processes, where low-level processes are subject to multi-instantiation. For simplicity, we first describe how to handle the case of one sub-process and one high-level process. Then we discuss how to generalize from that solution.

The approach assumes as input one or more event traces with annotated process instance IDs (PI-IDs). The first step is the *extraction and annotation* of sub-process instance IDs (SPI-IDs). This is domain-specific, requires expert

[1]`https://fluxicon.com/disco/`. Note that all figures from Disco in this report we obtained by setting the levers for "activities" and "paths" to 100%.

knowledge, and can generally not be fully automated. The transformation from the input log or event trace to one that contains SPI-IDs needs to be encoded, such that it can be applied to all traces with the same input format.

Once the SPI-IDs have been annotated, the event trace can be *split* into one trace for the high-level process and traces for the sub-process. Each sub-process trace contains all events with one SPI-ID. In the high-level trace, each event with a subprocess instance ID gets replaced with an event whose activity field is set to a generic term, for example, "sub-process", while all other attributes like time stamp and the process instance ID are retained.

Then, standard process mining *discovery* algorithms can be applied to these traces, such as the heuristic miner [26] or the fuzzy miner [10]. The results are two process models: one for the high-level process and one for the sub-process. The high-level process contains an activity labeled "sub-process". In the final *merge* step of the approach, the respective sub-process model replaces said activity, as an expanded sub-process with multi-instantiation.

This approach can be generalized into two directions: multiple sub-processes on the same level, and multiple levels of nesting. In the latter case, it suffices to recursively apply the procedure to the split sub-process trace instead of standard discovery. In the former case, multiple kinds of sub-processes can be distinguished, for example, SPI-ID$_1$, SPI-ID$_2$, etc. One assumption is that the membership in a particular kind of sub-process is exclusive, that is, no event belongs to multiple sub-processes. The activity name replacement for the high-level trace during the split step needs to reflect the different kinds, that is, the activity label becomes "sub-process-1", "sub-process-2", etc. Discovery is then done once per kind of sub-process and once for the high-level process, and the merge step merges the results by including all sub-process models at the appropriate point in the high-level model. In the following, we formally define the required concepts.

3.2 Formal Definition of Event Traces

We provide compact formal definitions of events and traces, based to a degree on the definitions in [24, Chapter 4].

Definition 1 *Let $\mathcal{E}$ be the* event universe*, that is, the set of all possible event identifiers, $\mathcal{PI}$ be the* process instance universe*, that is, the set of all possible process instance identifiers, and $\mathcal{A}$ be the set of activity names. We require equality and inequality to be defined for these sets.*

Events are characterized by various attributes*. Let $\mathcal{AN}$ be a set of attribute names for events. For an event $e \in \mathcal{E}$ and attribute name $n \in \mathcal{AN}$, $\#_n(e)$ denotes the value of attribute name n for event e. If the event e does not have an attribute named n, then $\#_n(e) = \perp$ (where $\perp$ denotes the null value).*

For the purpose of our analysis, an event e must have the following attributes: a *timestamp*, denoted by $\#_{time}(e)$, correspondence to an *activity*, denoted by $\#_{activity}(e) \in \mathcal{A}$, and a pointer to a *process instance*, denoted by $\#_{pi-id}(e) \in \mathcal{PI}$. We use the operator '$<$' to compare timestamps, that is, for events $e_1, e_2 \in \mathcal{E}$, $\#_{time}(e_1) < \#_{time}(e_2)$ indicates that event e_1 happened *before* event e_2.

In contrast to the original definition that groups events into cases [24], we consider the case or *process instance* as a mandatory attribute. This allows us

```
1. [2014-03-22 16:01:34,557] [Task:Pushing ami-d8b429e8 into group
   bpm--ASG for app bpm] com.netflix.asgard.Task 2014-03-22_16:01:34 22:
   {Ticket: null} {User: null} {Client: localhost 127.0.0.1}
   {Region: us-west-2} [Pushing ami-d8b429e8 into group bpm--ASG for
   app bpm] Disabling bpm / i-58ed9d51 in 1 ELBs.
2. [2014-03-22 ... app bpm] Terminate 1 instance [i-58ed9d51]
3. [2014-03-22 ... app bpm] It took 2m 38s for instance i-58ed9d51 to
   terminate and be replaced by i-4ed2a247
4. [2014-03-22 ... app bpm] It took 35s for instance i-4ed2a247 to go
   from Pending to InService
```

Figure 3.2: Sample log lines from rolling VM upgrade.

to flexibly re-assign events to cases. Further, the original log entry attribute $\#_{log-entry}(e)$ refers to whatever constitutes the originally logged event. Examples are a log lines in case of systems producing text-based log files, a database entry if the system stores its event logs there, or a message sent over a network.

Definition 2 *A* trace*, denoted by $\sigma = [e_1, e_2, \ldots, e_n]$, is a finite, non-empty sequence of events from the event universe $\mathcal{E}$ such that σ does not contain an event $e \in \mathcal{E}$ more than once, that is $\forall\, e_i, e_j \in \mathcal{E}, i, j \in [1, n], i \neq j : e_i \neq e_j$. The set of all traces for an event universe $\mathcal{E}$ is denoted by $\Sigma_{\mathcal{E}}$.*

In the following, we use the abbreviation $e_i \in \sigma$ to indicate that the event e_i is part of the trace σ, that is, $\exists\, e_j, j \in [1, n] : e_i = e_j$.

3.3 Sub-Process ID Extraction

As mentioned above, the extraction of sub-process IDs is highly application--specific. While we rely on a manual definition of the SPI-ID in this report, automatic techniques such as described in [2, 3] can be used for the subprocess identification. The result it either way an SPI-ID extraction function $\phi : \mathcal{E} \times \Sigma_{\mathcal{E}} \mapsto \mathcal{SPI} \cup \{\perp\}$, where $\mathcal{SPI}$ is the *sub-process instance universe*, that is, the set of all possible sub-process instance identifiers. ϕ requires that the event e is an event of the trace σ (i.e. $e \in \sigma$) and is defined as follows:

$$\phi(e, \sigma) = \begin{cases} \perp & \text{if } e \text{ is not part of the sub-process} \\ \mathit{spi\_id} & \text{otherwise} \end{cases}$$

Please note that if for two events $e_1, e_2 \in \sigma$ the corresponding sub-process IDs are equal and not null, that is $\phi(e_1, \sigma) = \phi(e_2, \sigma) \neq \perp$, then the corresponding process IDs need to be equal as well – i.e. $\#_{pi-id}(e_1) = \#_{pi-id}(e_2)$.

Given a sub-process extraction function ϕ, we can iterate through an event trace σ and annotate the SPI-ID for each event $e_i \in \sigma$ as $\#_{spi-id}(e_i) := \phi(e_i, \sigma)$. In the running example, rolling upgrade, each sub-process instance deregisters and terminates exactly one virtual machine. Some sample log lines for one sub-process instance are given in Fig 3.2. These cover a timespan of around 4 minutes, and many lines in between have been omitted. Also, except for the timestamps, the first 269 characters are identical for each log line. In order to improve readability of Fig 2.1, we have only included the full information for the first line.

In Line 1, Asgard disables VM `i-58ed9d51` in the load balancer. In Line 2, this VM is terminated (i.e., switched off and removed from the resource pool). Line 3 informs us that VM `i-58ed9d51` has been replaced by VM `i-4ed2a247`. Finally, Line 4 tracks the status of VM `i-4ed2a247` as it is booting up.

As can be seen, the SPI-ID can initially be set to the name of the VM it concerns, that is, `i-58ed9d51`. However, once it has been replaced by `i-4ed2a247`, each line mentioning VM `i-4ed2a247` actually belongs to the sub-process instance of `i-58ed9d51` – but `i-58ed9d51` is not listed any further. Note that sub-process instance identification can be quite application-specific and requires expert knowledge. It also shows why the sub-process instance ID extraction function ϕ requires the trace σ as argument: from the event derived from Line 4 alone, the right SPI-ID cannot be derived without that context.

In order to generalize from the case of a single sub-process, it suffices to have multiple extraction functions, $\phi_1, \phi_2, \dots, \phi_n$ with mutual exclusion:

$$\forall\, e \in \mathcal{E}, i \in [1,n] : [\phi_i(e,\sigma) \neq \bot] \Rightarrow [\forall\, j \in [1,n], i \neq j : \phi_j(e,\sigma) = \bot].$$

The results of these functions are annotated as separate attributes, that is, $\#_{spi-id_i}(e) := \phi_i(e,\sigma)$. SPI-ID extraction over multiple nested levels of processes can be isolated, and hence does not pose additional complexity.

3.4 Trace Splitting

As indicated in Section 3.1, we need to split an event trace into two traces: one for the high-level process, denoted by σ_H, and one for the sub-process, denoted by σ_S. Furthermore, for each event that belongs to the sub-process, we replace this event with one labeled "sub-process" in the event trace for the high-level process, and redefine the set of activity names as $\mathcal{A} := \mathcal{A} \cup \{$`'sub-process'`$\}$. This procedure is defined as follows:

For each event $e_i \in \sigma$:

- if $\#_{spi-id}(e_i) = \bot$, add e_i to σ_H;
- if $\#_{spi-id}(e_i) \neq \bot$:
  - generate a new event e'_i;
  - for all $n \in \mathcal{AN} : \#_n(e'_i) := \#_n(e_i)$.
  - set $\#_{pi-id}(e'_i) := \#_{spi-id}(e_i)$;
  - set $\#_{spi-id}(e'_i) := \bot$;
  - set $\#_{activity}(e_i) :=$`'sub-process'`;
  - add e_i to σ_H and e'_i to σ_S.

The trace σ_H then contains the events pertaining to the high-level process, and σ_S those for the sub-process. The generalization to n different sub-processes is achieved by generating additional traces $\sigma_{S_1}, ..., \sigma_{S_n}$, similar to σ_S.

3.5 Discovery and Merging

On the split traces, we perform standard process discovery separately for each trace. The usual trade-offs in discovery apply here, for example, between precision, fitness, and generalization. We do not discuss these trade-offs separately here, but rather point to standard ways of dealing with them in process mining [24].

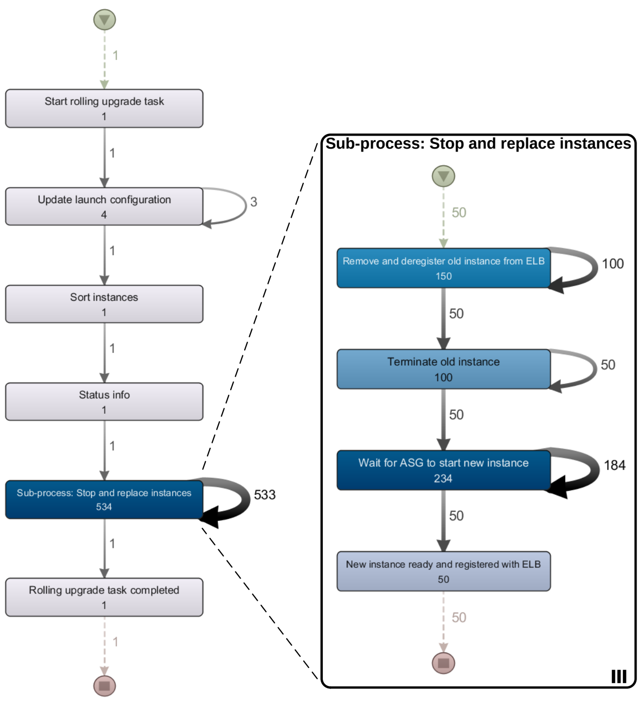

1
Start rolling upgrade task
1
1
Update launch configuration
4
3
1
Sort instances
1
1
Status info
1
1
Sub-process: Stop and replace instances
534
533
1
Rolling upgrade task completed
1
1
Sub-process: Stop and replace instances
50
Remove and deregister old instance from ELB
150
100
50
Terminate old instance
100
50
50
Wait for ASG to start new instance
234
184
50
New instance ready and registered with ELB
50
50

Figure 3.3: Discovered process: high-level process (left), sub-process (right), with association shown by dotted lines. We refer to the combined model as *RU MI* model.

For the running example, Fig. 3.3 shows the results of process discovery, for the high-level process (left) and for the sub-process (right). In the high-level process, the sub-process shows up as a regular activity. We added the box around the sub-process, along with the symbol for parallel multi-instantiation and the lines showing the connection between the two discovered processes. Merging the process models (not shown) is done by replacing the collapsed "sub-process" activity in the high-level process with an expanded sub-process, containing the process model for the sub-process.

3.6 Conformance Checking

Conformance checking tests if a set of event traces fit a given process model, or vice versa. Say, a conformance checking function $\gamma_{pm} : \mathcal{E} \times \Sigma_{\mathcal{E}} \mapsto \{\texttt{fit},\texttt{unfit}\}$ is available, where pm refers to a process model without multi-instantiation. Then conformance checking for process models with sub-processes with multi-instantiation can be achieved by viewing a sub-process model spm as a regular activity a_{spm} in the higher-level process model pm.

For each event e_i of a trace σ, we do the following. Say e_i corresponds to an activity a. If a belongs to pm directly, conformance is determined as $\gamma_{pm}(e_i, \sigma)$. If, in contrast, a belongs to the sub-process model spm, then we first construct e_i'

from e_i by setting $\forall n \in \mathcal{AN} \setminus \{activity\} : \#_n(e'_i) := \#_n(e_i)$ and $\#_{activity}(e'_i) := a_{spm}$. Then we check $\gamma_{pm}(e'_i, \sigma)$. If that returns `fit`, we construct e''_i from e_i as $\forall\, n \in \mathcal{AN} \setminus \{pi - id\} : \#_n(e''_i) := \#_n(e_i)$ and $\#_{pi-id}(e''_i) := \phi(e_i, \sigma)$ (i.e., replace the PI-ID with the SPI-ID). We then check $\gamma_{spm}(e''_i, \sigma)$, that is, the conformance on the sub-process level. The multi-instantiation in the sub-process is thus handled by conformance checking in the same way as multiple instances of a high-level process are handled: the right instance is identified by the PI-ID, or in the case of sub-processes by the SPI-ID which has been copied to $\#_{pi-id}(e''_i)$.

Using this method, nesting of sub-processes can be of arbitrary depth, and recursive conformance checking on sub-processes can cope with it. However, the core assumption is that the instance of the respective next-level sub-process can be derived from the combination of the trace σ and the event e_i.

4 Implementation & Evaluation

For our evaluation, we first describe the data sets and the implementation. Then, we discuss results of discovery and conformance checking.

4.1 Data Sets

The two data sets used for the evaluation are (i) Asgard's rolling upgrade logs, and (ii) examination data from a university. Rolling upgrade has been used as a running example throughout the report, and Fig. 3.2 shows some sample log lines. What is noteworthy here is the fact that each event corresponds to one log line. Some log lines may be considered noise for our purposes, and can be easily filtered out. In this sense, the quality of the logs produced by Asgard is relatively high: process type and PI-ID can be derived from parts of each log line, for example, `[Task:Pushing ami-d8b429e8 into group bpm--ASG for app bpm]` (corresponding to Line 1 of Fig. 3.2). Details about the abstraction from individual log lines to the process activities shown in Figs. 2.1 and 3.3 is described in [27].

The second data set corresponds to anonymized event logs from a university, and were given in XES (eXtensible Event Stream) format, an XML-based open standard for storing and tracing event logs.[1] The university events record when students take exams for specific courses. In the second half of the studies, students need to select two specializations (SBWLs). Each of the SBWLs consists of five courses. Exams for these courses can be taken individually ("Kurs" I-V) or for multiple of the five at once ("Fachprüfung"). Examination rules dictate that the exam of first course of each SBWL has to be passed before a student can take any further exams in that SBWL. In contrast to the rolling upgrade data set, neither noise filtration nor pattern matching were required as XES provides the traces in a structured format.

4.2 Implementation

We implemented both, SPI-ID extraction and trace splitting, as stand-alone Java applications. As such, they can be easily integrated into pre-processing

[1] `http://www.xes-standard.org/xesstandarddefinition/`

pipelines to prepare data for process mining, e.g., as described in [27]. For the university data set, we used OpenXES,[2] the reference implementation of the XES standard.

The SPI-ID extraction is domain-specific, and thus required domain expert knowledge. This domain knowledge is encoded in Java functions for the respective log types. For the rolling upgrade data, activities belonging to the sub-process are identified using regular expressions. We also use regular expressions to identify the IDs of the virtual machines, for example, `i-58ed9d51`. Similarly, we extracted the sub-process identifier in the university data set from the XES field `concept:name`.

Conformance checking is implemented as a RESTful service, based on the Restlet framework,[3] and forms part of a set of online error detection and diagnosis services as described in [28]. The current implementation requires approx. 2,800 lines of Java code and is based on the existing conformance checking implementation [28]. The extension incorporates the sub-process conformance checking described in Section 3.6, and adds the capability to accept XES as input. Making a call to the service from localhost takes on average about 9ms, hence does not introduce a significant performance overhead.

4.3 Process Discovery Evaluation

Comparison with classical approaches.

In order to assess the suitability of other mining approaches for multi-instantiated sub-processes, we tried to discover a meaningful model from the rolling upgrade data using several classical discovery algorithms. Table 4.1 shows that neither of the classical approaches was able to discover a satisfactory model on the original event traces. However, on the traces split by our approach, all of these algorithms provided perfectly acceptable results (with the exception of Uma, which was not tested for this case since no simplification of the split models was needed).

Discussion of discovery results.

The *RU MI* model as given in Fig. 3.3 was extracted from the first of the 10 rolling upgrade traces. We subsequently ran the sub-process model extraction on the remaining traces and got the identical model every single time (modulo some changes in transition frequencies). We performed the same extraction process without considering multi-instantiated sub-processes in order to get the *RU Chaos* model as given in Fig 2.1 (right). Unlike the *RU MI* model, the resulting models were not 100% identical, but differed only marginally (e.g., edges with only one occurrence were missing in some of the models).

We then ran the model discovery on the SBWL data set. For presentation purposes, we omit the overall process map in which the sub-process is embedded, and only show the corresponding SBWL-related models in Fig. 4.1. First, we filtered the data to obtain SBWL events only. This resulted in 10,590 events from 1,326 cases (i.e., students). When applying the fuzzy mining in Disco to this data set, we obtained a very large model – see Fig. 4.1 (a), referred to as

[2]`http://www.xes-standard.org/openxes/start/`
[3]`http://restlet.org/`

Discovery algorithm	Tool	Outcome
α-algorithm	ProM 6.3	8 unconnected Petri net parts due to self-loops. After removing repetition in events, a Petri Net similar to *RU Chaos* resulted (9 transitions, 16 places, 35 edges), called α model below.
Heuristics Miner	ProM 6.3	Straight sequence of the 9 activities, 5 with self-loops
Fuzzy Miner	Disco 1.6.5	*RU Chaos* model, Fig. 2.1, right
Simplification with Uma [5]	ProM 6.3	(Based on the α model, as it requires a Petri Net as input.) Results vary with parameter settings for Uma, and range from removing a few places and edges, to splitting off a loop of 2 transitions, to removing everything but the start place.

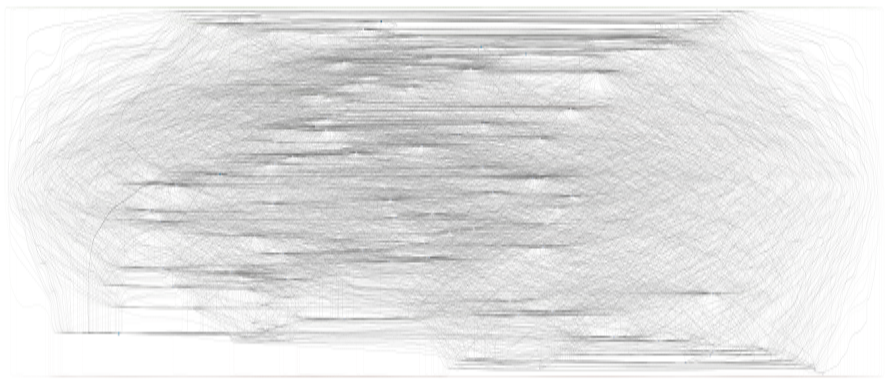
Table 4.1: Results of classical approaches for discovering models from rolling upgrade logs.

SBWL full– with 107 different activities. In order to obtain a fairer comparison with our MI approach, we then abstracted from the specialization of each exam. Furthermore, for purposes of conformance checking (see next section), we split the events for SBWL course I into passed and failed ones. Discovery on this transformed log resulted in a much more compact model, with only 7 activities (SBWL course I passed or failed, SBWL courses II - V and SBWL combined exam) – refer to Fig. 4.1 (b), called *SBWL abstracted* from here on. Finally, we took the filtered data set, split SBWL course I into pass/fail again, and applied our MI approach of SPI-ID extraction and trace splitting, which resulted in Fig. 4.1 (c), *SBWL MI*.

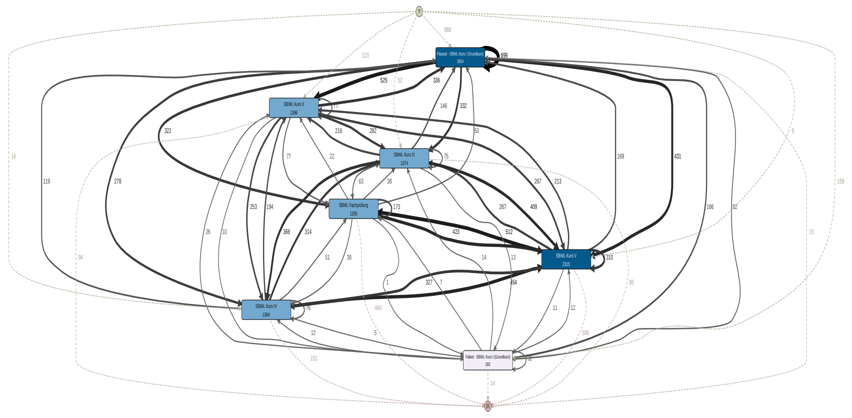
	Rolling Upgrade			SBWL		
Metric	Simple	Chaos	MI	Full	Abstracted	MI
Number of nodes	11	11	14	109	9	9
Number of edges	15	24	17	2818	62	50

Table 4.2: Graph metrics for the different models.

Table 4.2 quantifies the discovery results in terms of numbers of nodes and edges. In comparison with *SBWL abstracted*, the *SBWL MI* model has less edges. This is due to the fact that only transitions for the same specialization are shown, whereas *SBWL abstracted* also contains transitions across different specializations. Further, close inspection of the numbers reveal that in *SBWL MI* transitions from the start node to SBWL course I (passed or failed) are more common than in *SBWL abstracted* – which means that adherence to the rule of passing that course before proceeding to other exams in an SBWL is likely easier to observe in the *SBWL MI* model. Conformance checking will

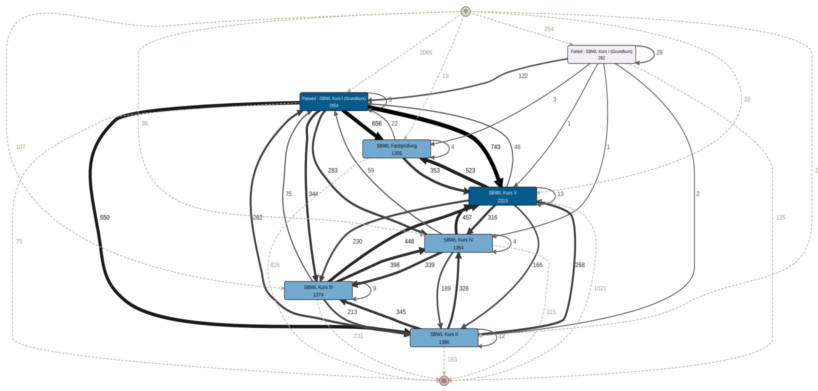


(a)



Passed - SBWL Kurs I (Grundkurs)
SBWL Kurs II
SBWL Kurs III
SBWL Fachprüfung
SBWL Kurs V
SBWL Kurs IV
Failed - SBWL Kurs I (Grundkurs)

(b)



Failed - SBWL Kurs I (Grundkurs)
Passed - SBWL Kurs I (Grundkurs)
SBWL Fachprüfung
SBWL Kurs V
SBWL Kurs IV
SBWL Kurs III
SBWL Kurs II

(c)

Figure 4.1: Discovered process from the SBWL data set: (a) only SBWL events, without further treatment; (b) SBWL events abstracted, without MI treatment; (c) SBWL events with MI treatment.

shed more light on this distinction.

4.4 Conformance Checking Evaluation

Analyzing the conformance of logs against discovered models has two purposes: to validate if the approach to conformance checking in MI models works, and to assess the quality of the discovered models.

Rolling upgrade.

In order to validate the conformance checking with MI treatment, we conducted two experiments on the rolling upgrade data set. Our hypothesis is that, by not considering MI, either the mined model is overly general, allowing arbitrary order and execution frequency of activities in the sub-process, or it lacks fitness by capturing only single instantiation of sub-processes.

		Actual	
		c	n
Classified as	c	TP	FP
	n	FN	TN

		Model		
Log	Metric	Simple	Chaos	MI
Correct log	TPR	44.8%	100%	100%
	FNR	**55.2%**	0%	0%
Erroneous log	TPR	7.57%%	24.8%	24.8%
	TNR	75.2%	0%	75.2%
	FPR	0%	**75.2%**	0%
	FNR	**17.2**%	0%	0%

Table 4.3: Top: result classification (c: conforming; n: non-conforming). Bottom: conformance checking results for two logs and three models (false classification in bold, rows containing only "0%" values omitted).

Table 4.3 provides an overview of the results. The left part serves as an explanation of how the results on the right were calculated – e.g., if an event is actually conforming (left column) and classified as conforming (top row) it is considered to be a true positive (TP). Similarly, if an event is actually non-conforming but classified as conforming, this is a false positive (FP). The table on the right uses rates of these measures, e.g., TPR is the true-positives rate (number of TP events divided by total number of events in the log).

First, we compared conformance of a correct MI log (Table 4.3 right, "Correct log") when checked against the three rolling upgrade models. The data used was the rolling upgrade of 50 VMs, 5 at a time, from 10 runs. As to be expected, the 6033 events in this data set conform 100% to the *RU MI* model (Fig. 3.3). This demonstrates that the approach to conformance checking in MI models works in principle. The log also conforms perfectly with the *RU Chaos* model (Fig. 2.1, right). In contrast, when checking the conformance of this log against the *RU Simple* model (Fig. 2.1, left), an average fitness of 44.8%

resulted. The log only contains conforming events, the remaining 55.2% thus are false negatives.

To test the case where the model is overly general, we modified the rolling upgrade data set with some degree of randomness (Table 4.3 right, "Erroneous log"), so that it still perfectly conforms with the *RU Chaos* model. In particular, in the *RU Chaos* model the four activities of the sub-process ("Remove and deregister...", "Terminate old instance", "Wait for ASG...", and "New instance ready...") are fully connected. With respect to these four activities, this model has only two constraints on a trace of corresponding events: the trace has to start the sub-process with "Remove and deregister...", and has to end it with "New instance ready...". We modified the 10 traces to adhere to these constraints, but reordered all events in between arbitrarily (using Microsoft Excel's random number generator). Note that the timestamps of the reordered events were kept in the original order. By construction, the conformance of this modified log with *RU Chaos* model is 100%. We then checked the conformance of the modified log against the *RU Simple* model and the *RU MI* model. The average fitness of the erroneous log against the *RU MI* model is 24.8%, that is, the introduced random reshuffling resulted in 75.2% of the events being non-conforming. Since the *RU Chaos* model classifies these 75.2% as conforming, they are actually FP. The *RU Simple* model detected the non-conforming events correctly, but classified an additional 17.2% as non-conforming – these are false negatives.

This demonstrates that our approach to MI treatment in both discovery and conformance checking can overcome the weaknesses of both baseline alternatives, that is, assuming single instantiation or assuming arbitrary order. As hypothesized, single instantiation can result in many false negatives, whereas arbitrary order can result in many false positives.

University data.

The process of studying to obtain a degree at most universities is highly flexible, human-driven, and many of its constraints cannot be expressed as flow constraints in the models. Examples for such constraints involve combinations of grades, times, and groups of exams passed (not taken). These constraints are strictly enforced by the information system of the university, but do not lend themselves naturally to conformance checking.

One exception for the SBWLs in the university data set is the aforementioned rule that Course I must be successfully passed before other exams in that SBWL can be taken. As can be seen from Fig. 4.1 (c), this rule is not always followed. The reasons are two-fold: in some cases the rule is simply not enforced; in others, students obtain an exemption from that rule from the responsible lecturer.

In order to perform conformance checking, we needed to have a *normative model*, i.e., a process model that shows the permitted process. Conformance of each event trace can then be checked against that model. For simplicity, we designed the model shown in Fig. 4.2. This model implements the mentioned rule, but is overly general otherwise, such that e.g., the same exam can be taken infinitely often.

In contrast to the rolling upgrade data, where we compared the suitability of three different models through conformance checking, here we compare conformance checking with two different input filters: *abstraction-only* (as in *SBWL abstracted*, Fig. 4.1 (b)) vs. splitting the traces based on our *MI* approach (as

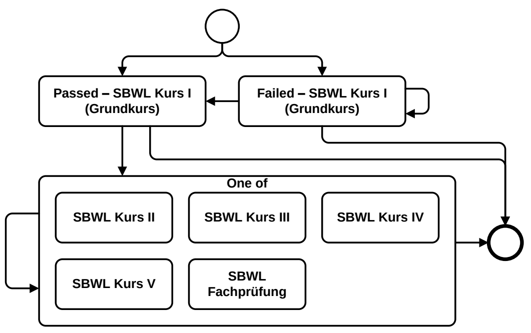


Figure 4.2: Normative process model for SBWL.

in *SBWL MI*, Fig. 4.1 (c)). Table 4.4 shows the results, where it is important to note that the percentages are aggregated into traces (not events). On an individual event basis, the average fitness rates are 92.6% (abstraction-only) vs. 96.7% (MI). As can be observed from the table, the false negative rate of abstraction-only is 37.6%, which is prohibitively high for many use cases. In contrast, the MI approach can separate the different specializations, and clearly identify where students did not adhere to the rule.

		Input filter	
Log	Metric	Abstraction-only	MI
University data	TPR	45.6%	83.1%
	TNR	16.8%	16.8%
	FNR	**37.6%**	0%

Table 4.4: Conformance checking results for the University data set.

5 Related Work

This work relates to the research area that aims to improve the understanding of process mining results. In general, it can be subdivided into abstraction techniques that work on the logs, techniques that work with dependencies between artifacts, and techniques that abstract resulting process models.

One of the first approaches towards clustering sets of events has been defined in [8, 9, 10]. This work builds on the observation that events are often more fine-granular than activities on a business level, and therefore clusters should be treated as a whole. This idea is closely related to the concept of a subprocess. Further approaches for clustering correlated events extend this line of research in various directions. In this context, various techniques are used for deleting insignificant behaviour, e.g. in terms of event classes [13] and event relationships [11]. Clustering criteria are extended based on statistics [14], temporal proximity [23], attribute values [20], and text content [1]. Domain-specific refinements for source code are discussed in [15]. Model hierarchies are generated based on clustering traces of similar behavior by the approach of [7]. However, non of these approaches explicitly deals with multiple instantiation tasks.

The research stream on artifact-centric process mining deals with the appropriate handling of the underlying relationships of artifacts that relate to a business process. On the other hand, this entails the automatic identification of potentially n:m relationships between artifacts and related events. This problem is known in process mining for a while, e.g. [6]. It has been tackled more generally in database research in [2] and adapted to event logs in [3]. Our work shares the idea with artifact-centric mining that a decomposition of log data helps to apply classical techniques on subsets of the log [4, 19]. As our logs do not stem from autonomous artifacts, we can build on a simpler conceptual model as compared to the Proclets approach in [4]. Also, we report evaluations with real-world data, emphasizing the importance of MI decomposition for accurate conformance checking.

A complementary stream of research investigates opportunities for abstracting process models, which might have resulted from process mining. One prominent approach here is to use structural decomposition and to abstract, for instance, using a slider technique [17, 18]. Also abstractions based on textual content [12, 21] and behavioral abstractions have been investigated [22]. A different approach in contrast to abstraction is to rework the process model to be more structured [16]. All these approaches work on process models in general. A specific technique to post-process overly complex mined models is presented in [5]. It makes use of unfoldings and filtering. We tested if this or other approaches could provide meaningful results on our data – see Table 4.1.

Altogether, our research complements prior work on mining processes with n:m artifact relationships with a specific approach to handle multiple instantiation, and highlights the sensitivity of conformance towards an appropriate identification of MI subprocesses.

6 Conclusions and Future Work

In this report, we have addressed the problem of dealing with multiple instantiation of sub-process in process mining. Our contribution is an approach for making use of sub-process instance IDs to hierarchically mine event logs. To this end, we have defined procedures for annotating and splitting traces, applying mining on them separately, and integrating the results. Furthermore, we have described how conformance checking can be adapted to multi-instantiated sub-processes. The approach has been implemented and evaluated using log data from cloud management and student examinations. The results from our evaluation demonstrate that our approach of extracting multiple instantiation of sub-processes effectively overcomes weaknesses of classical approaches with regards to process discovery and conformance checking, respectively.

In future research, we aim to investigate opportunities for automating those steps that are currently semi-automated in our approach. In particular, the automatic extraction – or at least suggestion – of potential sub-process instance IDs might benefit from techniques from data mining. Furthermore, we plan to integrate our prototype into ProM in order to enable its dynamic combination with different techniques for mining the separated processes.

Acknowledgments

We thank Fluxicon for providing academic licenses for Disco. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

Bibliography

[1] T. Baier and J. Mendling. Bridging abstraction layers in process mining by automated matching of events and activities. In *BPM*, pages 17–32, 2013.

[2] J. Bauckmann, U. Leser, F. Naumann, and V. Tietz. Efficiently detecting inclusion dependencies. In *IEEE 23rd International Conference on Data Engineering, 2007. ICDE 2007.*, pages 1448–1450. IEEE, 2007.

[3] R. Conforti, M. Dumas, L. García-Bañuelos, and M. L. Rosa. Beyond tasks and gateways: Discovering BPMN models with subprocesses, boundary events and multi-instance markers. In S. Sadiq, P. Soffer, and H. Voelzer, editors, *Business Process Management 2014*, Lecture Notes in Computer Science. Springer, 2014.

[4] D. Fahland, M. de Leoni, B. F. van Dongen, and W. M. P. van der Aalst. Conformance checking of interacting processes with overlapping instances. In S. Rinderle-Ma, F. Toumani, and K. Wolf, editors, *Business Process Management - 9th International Conference, BPM 2011, Clermont-Ferrand, France, August 30 - September 2, 2011. Proceedings*, volume 6896 of *Lecture Notes in Computer Science*, pages 345–361. Springer, 2011.

[5] D. Fahland and W. M. P. van der Aalst. Simplifying discovered process models in a controlled manner. *Information Systems*, 38(4):585–605, 2013.

[6] K. Gerke, A. Claus, and J. Mendling. Process mining of rfid-based supply chains. In B. Hofreiter and H. Werthner, editors, *2009 IEEE Conference on Commerce and Enterprise Computing, CEC 2009, Vienna, Austria, July 20-23, 2009*, pages 285–292. IEEE Computer Society, 2009.

[7] G. Greco, A. Guzzo, and L. Pontieri. Mining taxonomies of process models. *Data & Knowledge Engineering*, 67(1):74–102, Oct. 2008.

[8] C. W. Günther, A. Rozinat, and W. M. P. van der Aalst. Activity mining by global trace segmentation. In *BPM Workshops*, pages 128–139, 2009.

[9] C. W. Günther and W. M. P. van der Aalst. Mining activity clusters from low-level event logs,. In *BETA Working Paper Series*, volume 165. Eindhoven University of Technology, 2006.

[10] C. W. Günther and W. M. P. van der Aalst. Fuzzy mining: adaptive process simplification based on multi-perspective metrics. In *Proceedings of BPM 2007*, pages 328–343. Springer, 2007.

[11] R. P. Jagadeesh Chandra Bose, F. M. Maggi, and W. M. P. van der Aalst. Enhancing declare maps based on event correlations. In *Proceedings of BPM 2013*, pages 97–112, Beijing, China, Aug. 2013.

[12] H. Leopold, J. Mendling, H. A. Reijers, and M. L. Rosa. Simplifying process model abstraction: Techniques for generating model names. *Information Systems*, 39:134–151, Jan. 2014.

[13] J. Li, R. Bose, and W. M. P. van der Aalst. Mining context-dependent and interactive business process maps using execution patterns. In *BPM'2010 Workshops, volume 66 of LNBIP*, pages 109–121. Springer, 2011.

[14] H. R. M. Nezhad, R. Saint-Paul, F. Casati, and B. Benatallah. Event correlation for process discovery from web service interaction logs. *VLDB J.*, 20(3):417–444, 2011.

[15] R. Pérez-Castillo, B. Weber, I. G.-R. de Guzmán, M. Piattini, and J. Pinggera. Assessing event correlation in non-process-aware information systems. *Software and Systems Modeling*, pages 1–23, Sept. 2012.

[16] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas. Structuring acyclic process models. *Information Systems*, 37(6):518–538, 2012.

[17] A. Polyvyanyy, S. Smirnov, and M. Weske. Process Model Abstraction: A Slider Approach. In *EDOC*, pages 325–331. IEEE, 2008.

[18] A. Polyvyanyy, S. Smirnov, and M. Weske. On application of structural decomposition for process model abstraction. In *BPSC*, pages 110–122, 2009.

[19] V. Popova, D. Fahland, and M. Dumas. Artifact lifecycle discovery. *CoRR*, abs/1303.2554, 2013.

[20] S. Rozsnyai, A. Slominski, and G. T. Lakshmanan. Discovering event correlation rules for semi-structured business processes. In *DEBS*, pages 75–86, 2011.

[21] S. Smirnov, H. A. Reijers, and M. Weske. From fine-grained to abstract process models: A semantic approach. *Information Systems*, 37(8):784–797, 2012.

[22] S. Smirnov, M. Weidlich, and J. Mendling. Business process model abstraction based on synthesis from well-structured behavioral profiles. *International Journal of Cooperative Information Systems*, 21(1):55–83, 2012.

[23] M. Steinle, K. Aberer, S. Girdzijauskas, and C. Lovis. Mapping moving landscapes by mining mountains of logs: Novel techniques for dependency model generation. In *VLDB*, pages 1093–1102, 2006.

[24] W. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes.* Springer, 2011.

[25] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed Parallel Databases*, 14(1):5–51, 2003.

[26] A. J. M. M. Weijters, W. M. P. van der Aalst, and A. K. Alves de Medeiros. Process mining with the heuristics miner-algorithm. In *BETA Working Paper Series*, volume 166. Eindhoven University of Technology, 2006.

[27] X. S. Xu, I. Weber, H. Wada, L. Bass, L. Zhu, and S. Teng. Detecting cloud provisioning errors using an annotated process model. In *MW4NextGen'13: Proceedings of the 8th Workshop on Middleware for Next Generation Internet Computing*, Beijing, China, December 2013.

[28] X. S. Xu, L. Zhu, I. Weber, L. Bass, and W. Sun. Pod-diagnosis: Error diagnosis of sporadic operations on cloud applications. In *DSN'14: IEEE/IFIP International Conference on Dependable Systems and Networks*, Atlanta, GA, USA, June 2014.