Mining Processes with Multi-Instantiation

Ingo Weber[1,2] Mostafa Farshchi[1,3] Jan Mendling[4] Jean-Guy Schneider[3]
[1]Software Systems Research Group, NICTA, Sydney, Australia
firstname.lastname@nicta.com.au
[2]School of Computer Science & Engineering, University of New South Wales
[3]Swinburne University of Technology, Hawthorn, Australia
{mfarshchi,jschneider}@swin.edu.au
[4]Wirtschaftsuniversität Wien, Vienna, Austria
jan.mendling@wu.ac.at

ABSTRACT

Process mining, in particular discovering process models by mining event traces, is becoming a widely adopted practice. However, when the underlying process contains sub-processes which are instantiated multiple times in parallel, 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 paper, 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 a real-world data set.

Categories and Subject Descriptors

H.1.0 [**Information Systems**]: Models and Principles

General Terms

Documentation, Verification, Design

Keywords

Process Mining, Multi-Instantiation, Discovery, Conformance

1. INTRODUCTION

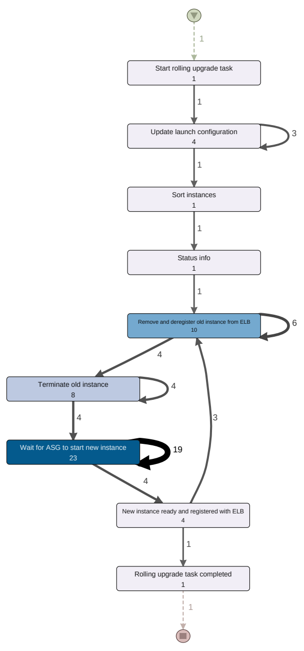
Process mining has two main directions: mining event logs and deriving process artifacts from them, termed *discovery*; and comparing event logs with process artifacts, *conformance checking*. Process mining is a very active area of research and is becoming a widely adopted practice [24]. However, when 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, and hence producing events that are inter-tangled in the resulting sequential event logs. The well-known workflow patterns [25] 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, 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 [29], 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 was recently discussed in the context of artifact-centric process mining [5, 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 high rates of either false positives or false negatives, respectively.

In this paper, we propose an extension to well-known process mining techniques, adding the capability of handling multi-instantiated sub-processes to both discovery and conformance checking. To this end, we define strategies for extracting sub-process identifiers (IDs) and making those explicit in a pre-processing step. 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. We have evaluated the approach with two independent data sets taken from real-world applications, namely cloud resource management and student examination logs. For space reasons, we only present the evaluation based on the first data set in this paper; the full work can be found in a Technical Report [26]. This evaluation reveals the benefits in terms of precision and fitness against the baseline of standard process mining algorithms.

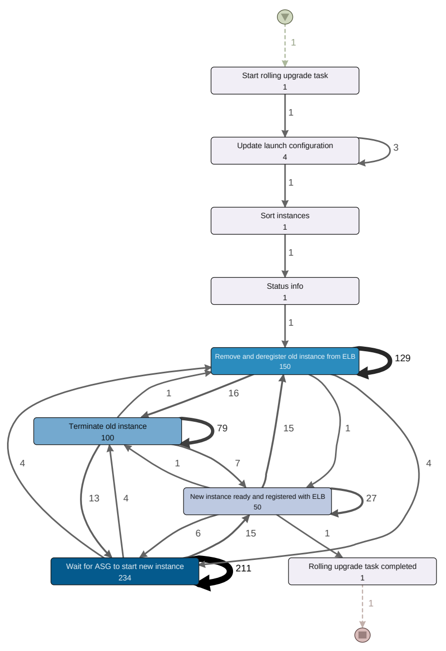
This paper is organized as follows. The problem is showcased in Section 2. An approach and a formal model are given in Section 3. The implementation and evaluation are discussed in Section 4. Section 5 contrasts the work against related work, and Section 6 concludes.

SAC'15 April 13-17, 2015, Salamanca, Spain.
Copyright 2015 ACM 978-1-4503-3196-8/15/04.
http://dx.doi.org/10.1145/2695664.2699493 ...$15.00.



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

Figure 1: Mined model from rolling upgrade (RU): 4 VMs, 1 at a time, referred to as *RU Simple* model from here on.



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: Mined model from RU: 50 VMs, 5 at a time, without treatment of multi-instantiated sub-processes, called *RU Chaos* model from here on.

2. MOTIVATING EXAMPLE

In separate work [28, 29], 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 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, to maintain full availability of services, 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, the procedure is repeated. This is done until all n VMs have been replaced. In previous work [28, 29], we examined the logs produced by Asgard's rolling upgrade procedure, (i) to understand 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. 1 shows a mined process model using the fuzzy miner [11] of 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 ($x = 1$). After the 4th iteration, the process completes. The individual steps to pre-process the log of Asgard so that it can be used in Disco are described in [28]. The right-hand side of Fig. 2 shows an analogously mined model, however, in this case from logs where 50 VMs have been upgraded, 5 at a time ($n = 50, x = 5$). As can be seen, the mined process hardly captures the sequencing at the level of individual instances, but creates a highly-connected model instead.

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

Our work on cloud resource management includes online conformance checking [29]. The model on the left-hand side of Fig. 1 can be used to that end, in cases where $x = 1$. If $x > 1$, 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 paper is to find a better balance in case of multiple instances.

3. HANDLING MULTI-INSTANTIATION

This section presents an approach for handling MI in process mining. We start with an overview and formal definitions, followed by explanations of the inidividual steps.

3.1 Approach Overview

Fig. 3 shows the steps for discovering hierarchical processes with multi-instantiation. For simplicity, we first describe how to handle the case of one sub-process and one high-level process, and then generalize.

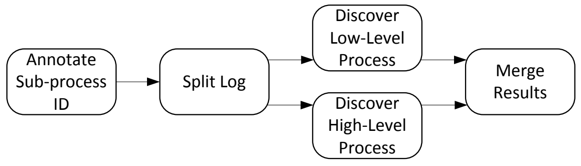


Figure 3: Overview of the MI approach

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, needs to reflect the specifics of the input format, and requires expert knowledge; and deriving an SPI-ID extraction function cannot be fully automated in general. While some automatic support is certainly possible, this is not the focus of this paper. 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 an SPI-ID gets replaced with an event whose activity field is set to a generic term, e.g., "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 [27] or the fuzzy miner [11]. 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 partly 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 has no 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 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 line 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.

```
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 4: Sample log lines from rolling VM upgrade.

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 event universe $\mathcal{E}$ is denoted by $\Sigma_{\mathcal{E}}$.*

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

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 extraction function in this paper, automation techniques such as described in [2, 4] may help in the subprocess identification. The result is 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:

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

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 4. 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 4, 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`. Line 4 tracks the status of VM `i-4ed2a247` while 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, $\phi_1, \phi_2, \ldots, \phi_n$ with mutual exclusion:

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

The results of these extraction 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 mentioned above, 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. 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].

For the running example, Fig. 5 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

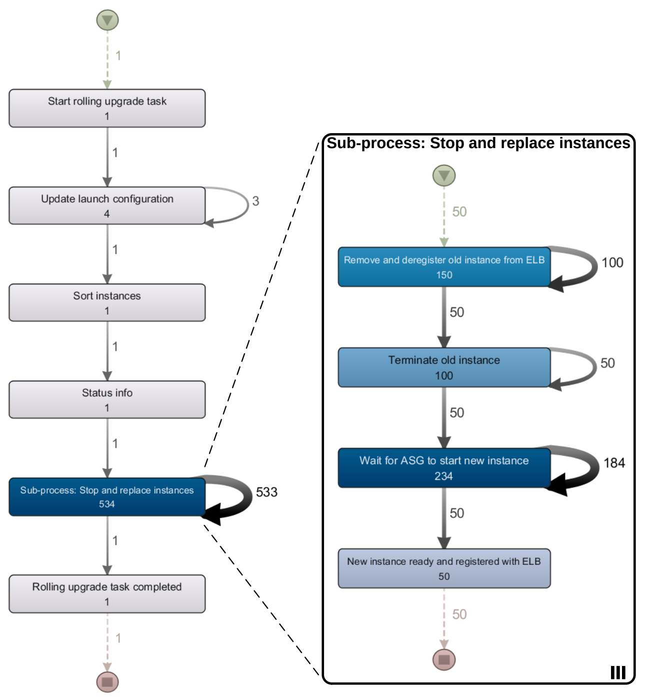

1
Start rolling upgrade task
1
1
Update launch configuration
4
3
1
Sort instances
1
1
Status info
1
1
Sub-process: Stop and replace instances
534
533
1
Rolling upgrade task completed
1
1
Sub-process: Stop and replace instances
50
Remove and deregister old instance from ELB
150
100
50
Terminate old instance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

Figure 5: 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.

checking function $\gamma_{pm} : \mathcal{E} \times \Sigma_{\mathcal{E}} \mapsto$ `{fit,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. The assumption is only that the instance of the respective next-level sub-process can be derived from the combination of trace σ and event e_i.

4. IMPLEMENTATION & EVALUATION

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

4.1 Data Set

The data set used for the evaluation is Asgard's rolling upgrade, which has been used as a running example throughout the paper. Fig. 4 shows some sample log lines. We used logs from 10 runs of rolling upgrade, each upgrading 50 VMs, 5 at a time ($n = 50, x = 5$). 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]` (see Line 1 in Fig. 4). Details about the abstraction from individual log lines to the process activities as shown in Fig. 1 is described in [28].

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 pipelines to prepare data for process mining, e.g., as described in [28].

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`.

Conformance checking is implemented as a RESTful service, based on the Restlet framework,[2] and forms part of a set of online error detection and diagnosis services as described in [29]. The current implementation has approx. 2,800 lines of Java code and is based on the existing conformance checking implementation [29]. The extension incorporates the sub-process conformance checking described in Section 3.6. A call to the service from localhost takes on average about 9ms, and hence does not introduce a significant performance overhead.

4.3 Process Discovery Evaluation

4.3.1 *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 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).

4.3.2 *Discussion of discovery results*

The *RU MI* model as given in Fig. 5 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

[2]`http://restlet.org/`

Discovery Algorithm	Outcome
α-algorithm in 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 in ProM 6.3	Straight sequence of the 9 activities, 5 with self-loops
Fuzzy Miner in Disco 1.6.5	*RU Chaos* model, Fig. 2
Simplification with Uma [6] in 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 1: Results of classical approaches for discovering models from rolling upgrade logs.

Fig 2. Unlike the *RU MI* model, the resulting models were not 100% identical, but differed only marginally. Table 2 quantifies the discovery results in terms of numbers of nodes and edges.

	Rolling Upgrade		
Metric	Simple	Chaos	MI
Number of nodes	11	11	14
Number of edges	15	24	17

Table 2: Graph metrics for the different models.

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.

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.

Tables 3 and 4 provide an overview of the results. Table 3 serves as an explanation of how the results in Table 4 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). Table 4 uses rates of these measures, e.g., TPR is the true-positives rate (number of TP events

		Actual	
		c	n
Classified as	c	TP	FP
	n	FN	TN

Table 3: Result classification (c: conforming; n: non-conforming).

		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: Conformance checking results for two logs and three models (false classification in bold, rows containing only "0%" values omitted).

divided by total number of events in the log).

First, we compared conformance of a correct MI log (Table 4, "Correct log") when checked against the three rolling upgrade models. As to be expected, the 6033 events from the 10 runs of rolling upgrade conform 100% to the *RU MI* model (Fig. 5). 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). In contrast, when checking the conformance of this log against the *RU Simple* model (Fig. 1), an average fitness of 44.8% resulted. The log only contains conforming events, the remaining 55.2% thus are false negatives.

To test cases where the *RU Chaos* model is overly general, we modified the rolling upgrade data set with some degree of randomness (Table 4, "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 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.

5. RELATED WORK

This work relates to research 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 [9, 10, 11]. 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 [3]. 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 [8]. 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. [7]. It has been tackled more generally in database research in [2] and adapted to event logs in [4]. 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 [5, 19]. As our logs do not stem from autonomous artifacts, we can build on a simpler conceptual model as compared to the Proclets approach [5]. Also, our evaluations with real-world data emphasize 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. One specific technique to post-process overly complex mined models makes use of unfoldings and filtering [6]. We tested if this or other approaches could provide meaningful results on our data – see Table 1.

In summary, 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 checking towards an appropriate identification of MI subprocesses.

6. CONCLUSIONS AND FUTURE WORK

In this paper, 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; an evaluation of the approach using an independent real-world data set containing student examination events is omitted due to space restrictions, but can be found in [26]. The results from our evaluation demonstrate that our approach of treating multiple instantiation of sub-processes effectively overcomes weaknesses of classical approaches with regards to both process discovery and conformance checking.

In future research, we plan to incorporate timing information from events in discovery and conformance checking. When combining advanced duration profiles with multi-instantiation, we aim to achieve an unprecedented, fine-grained analysis of execution correctness in near-realtime. This is particularly critical for cloud management operations, which often have significant failure rates and can result in large-scale outages. We further plan to semi-automate the specification of SPI-ID extraction functions.

7. ACKNOWLEDGMENTS

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

8. REFERENCES

[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 ICDE*, pages 1448–1450, 2007.

[3] R. Bose, F. Maggi, and W. van der Aalst. Enhancing declare maps based on event correlations. In *BPM*, 2013.

[4] 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 *BPM*, 2014.

[5] 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 *BPM*, pages 345–361, 2011.

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

[7] K. Gerke, A. Claus, and J. Mendling. Process mining of RFID-based supply chains. In B. Hofreiter and H. Werthner, editors, *IEEE Conference on Commerce and Enterprise Computing, CEC*, 2009.

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

[9] 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.

[10] 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. TUE, 2006.

[11] C. W. Günther and W. M. P. van der Aalst. Fuzzy mining: adaptive process simplification based on multi-perspective metrics. In *BPM*, 2007.

[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. van der Aalst. Mining context-dependent and interactive business process maps using execution patterns. In *BPM Workshops*, 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, 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 *IEEE EDOC*, pages 325–331, 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. *Intl. J. 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] I. Weber, M. Farshchi, J. Mendling, and J.-G. Schneider. Mining processes with multi-instantiation: Discovery and conformance checking. Technical Report 201420, School of CSE, University of New South Wales, Sydney, Australia, Sept. 2014.

[27] 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. TUE, 2006.

[28] X. Xu, I. Weber, H. Wada, L. Bass, L. Zhu, and S. Teng. Detecting cloud provisioning errors using an annotated process model. In *Workshop on Middleware for Next Generation Internet Computing*, 2013.

[29] X. Xu, L. Zhu, I. Weber, L. Bass, and W. Sun. POD-diagnosis: Error diagnosis of sporadic operations on cloud applications. In *IEEE/IFIP DSN*, 2014.