WIRTSCHAFTS
UNIVERSITÄT
WIEN VIENNA
UNIVERSITY OF
ECONOMICS
AND BUSINESS

EQUIS ACCREDITED AACSB ACCREDITED ASSOCIATION AMBAS ACCREDITED

Bridging abstraction layers in process mining

Baier, Thomas; Mendling, Jan; Weske, Mathias

Published in:
Information Systems

DOI:
10.1016/j.is.2014.04.004

Published: 01/12/2014

Document Version
Early version, also known as preprint

Link to publication

Citation for published version (APA):
Baier, T., Mendling, J., & Weske, M. (2014). Bridging abstraction layers in process mining. *Information Systems*, 1-40. https://doi.org/10.1016/j.is.2014.04.004

Download date: 09. Aug 2024

Bridging abstraction layers in process mining

Thomas Baier[a,*], Jan Mendling[b], Mathias Weske[a]

[a] *Hasso Plattner Institute at the University of Potsdam*
Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, Germany
[b] *Wirtschaftsuniversität Wien, Augasse 2-6, 1090 Vienna, Austria*

Abstract

While the maturity of process mining algorithms increases and more process mining tools enter the market, process mining projects still face the problem of different levels of abstraction when comparing events with modeled business activities. Current approaches for event log abstraction try to abstract from the events in an automated way that does not capture the required domain knowledge to fit business activities. This can lead to misinterpretation of discovered process models. We developed an approach that aims to abstract an event log to the same abstraction level that is needed by the business. We use domain knowledge extracted from existing process documentation to semi-automatically match events and activities. Our abstraction approach is able to deal with n:m relations between events and activities and also supports concurrency. We evaluated our approach in two case studies with a German IT outsourcing company.

Keywords: Process Mining, Abstraction, Event Mapping

1. Introduction

Process mining finds increasing uptake in practice. Using the event data logged by IT systems, process mining algorithms discover and enhance process models or check whether the execution of a process conforms to specification [1]. Looking at conformance checking and enhancement of process models, it is obvious that the events stemming from IT systems have to be

*Phone: +49 331 5509273
Email address: thomas.baier@hpi.uni-potsdam.de (Thomas Baier)

mapped to activities defined in process models. However, the events are typically more fine-granular than the activities defined by business users. This implies that different levels of abstraction need to be bridged in order to use these process mining techniques. Furthermore, such a mapping is not only necessary for conformance checking and process model enhancement, but also for discovery. The benefit of a discovered process model can only be fully exploited if the presented results are on an abstraction level that is easily understandable for the business user. Nevertheless, most current process mining techniques assume that there is a 1:1 mapping between events and activities. Only a few abstraction approaches address the mapping challenge by clustering events that can be bundled into singular activities (see e.g. [5, 6, 7]). However, these techniques have limited capabilities in dealing with complex mappings between events and activities and most often neglect n:m relationships and concurrency in the execution. Also, they provide no or only limited support for correctly refining these mappings based on domain knowledge.

In this paper, we build on ideas presented in prior work [2, 3] for tackling this mapping problem. Our contribution is a mapping approach that suggests relations between events and activities in an automated manner using existing process documentation as e.g. work instructions. For the set of suggested event-activity relations, we define means to dissolve n:m relations. In contrast to existing approaches, the method introduced in this paper is designed to deal with concurrency and to handle n:m relations between events and activities. We extend our previous work to deal with the complete life cycle of activities and to allow for zooming functionality in process discovery. Moreover, we introduce more sophisticated means to address the challenges of shared functionalities and loops. The capabilities of our approach are evaluated based on two case studies with a service outsourcing provider. The results demonstrate its benefits and emphasize the sensitivity of conformance and performance analysis to the defined mapping problem. Our approach can be used as a preprocessing step for every process mining technique, and therefore adds to the overarching field of business process analysis.

The paper is structured as follows. Section 2 describes the problem of different abstraction levels of event logs and process models. Furthermore, the preliminaries for our approach are introduced. Section 3 introduces the strategies to overcome the gap between abstraction levels of event log and process model. In section 4, we show the results from case studies where we

benchmarked our approach against manually created mappings, and outline the implications on conformance testing and performance analysis. Related work and prior research is reviewed in section 5. Section 6 discusses the implications of our work on research and practice, and elaborates on current limitations. In section 7 we conclude with a short summary of this work and give an outlook to future research.

2. Problem statement and preliminaries

This section provides an example to illustrate the research problem and introduces the preliminaries on which our approach builds.

2.1. Problem description

In order to make the problem more comprehensive, we will use the Incident Management process as defined in the IT Infrastructure Library (ITIL) [4]. Figure 1 shows the process model at a very abstract level. At the bottom of Fig. 1, an excerpt of six cases from a corresponding event log is displayed. The different abstraction levels of process model and execution log spawn multiple challenges that need to be addressed in order to map the events to their corresponding activities.

The first challenge (1) is the effective usage of external knowledge for the abstraction of events to defined activities. While there are different approaches for abstracting event logs to a higher level, like [5, 6, 7], none of these approaches makes systematic use of external knowledge to map events to defined activities. Typically, organizations maintain detailed textual documentation of a process that extends the information provided in the model on a lower abstraction level. This knowledge should be leveraged to bridge the gap between the high-level process model and the low-level event log.

The second challenge (2) is the *unknown relation of events and activities*. In practical settings it is often not known a priori which events refer to which activity. An automated derivation of the relation between events and activities is non-trivial, as simple string matching techniques often do not work between different levels of abstraction. In the given Incident Management example for instance the two events "Group" and "Details" have to be related to the activity "Incident logging". Existing abstraction approaches try to solve this challenge by clustering events that occur in temporal proximity [7]. Often this does not reflect the partitioning of activities as domain

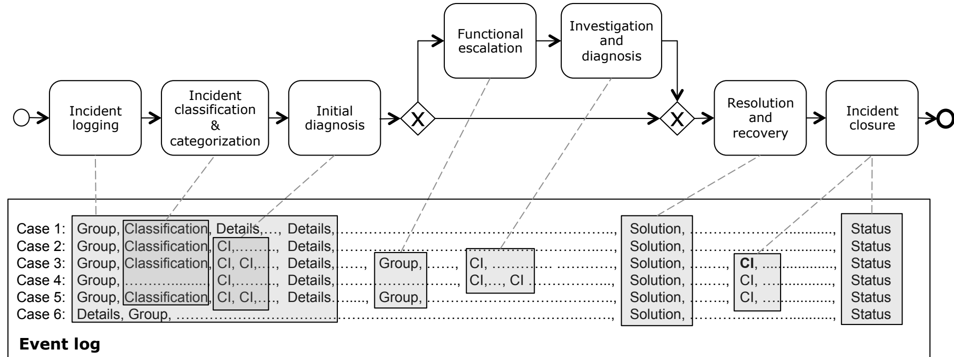

Figure 1: Example of event to activity relations: Incident Management process model and low-level event log with shared functionalities and concurrency

experts would expect it. Furthermore, different events also may have different meanings with respect to the life cycle of an activity. For instance, the event "Group" might always signal the start of activity "Incident logging", while the event "Details" signals the end of the activity. This information is important when it comes to performance analysis and the calculation of activity durations. Last but not least, certain events might not be interesting for the aimed abstraction and should be filtered out in a convenient way.

The third challenge (3) is the *use of shared functionalities*, where different activities access the same functionality of the IT system. In this case, an event of the same type refers to multiple different activities. For example, Fig. 1 depicts the event "CI" that belongs to either one of the activities "Initial diagnosis", "Investigation and diagnosis" or "Incident closure". The abbreviation CI stands for the configuration item, i.e. the affected IT component. Depending on when the CI is changed during the process, the change of

the CI refers to different activities. When happening at the beginning of the process execution, the event reflects the activity "Initial diagnosis". Later it occurs when the configuration item is changed during the "Investigation and diagnosis" or during the quality checks of the incident closure. Moreover, there can be events, such as status updates, that potentially belong to every activity. In practice, we even encountered single instances of events which signal the execution of multiple activities. When looking at shared functionalities, existing approaches either neglect their existence, as in [8, 6], or provide only limited means for disambiguating the relation of a shared functionality to its corresponding activities, as in [7].

The fourth challenge (4) are *loops and concurrency in the process execution*. Regarding *loops* in the execution, we have to distinguish between loops on the activity level and loops on the sub-activity level. Whenever an event related to one specific activity occurs more than once, it has to be decided whether this is a repetition of the activity itself or a repetition of a sub-activity within the same instance of the activity. For example, as to the event "CI", often two occurrences are related to the same activity in Fig. 1. Domain knowledge is needed to decide whether this is a repetition of the complete activity or only a repetition of the sub-activity to which the event relates to. This knowledge is crucial for conformance analysis as well as for process discovery.

Similarly to loops, *concurrency* poses a challenge to the abstraction of event logs. Current event log abstraction approaches, like [7], assume that only events that occur closely to each other in the log belong to the same occurrence of an activity. Concerning the cases 2 – 5, the events "Group" and "Details" would not be related to the same occurrence of the activity "Incident logging", but to two different occurrences of that activity. This leads to a loop in the discovered process model that signals rework, whereas in reality no rework has been done as both sub-activities have only been executed once. But in contrast to the assumed process model, the activities were not executed in sequence but in parallel. Hence, different means have to be found to deal with concurrent execution of sub-activities.

Concurrency and the distinction between loops on activity level vs. loops on sub-activity level is only implicitly tackled in prior research and existing approaches do not allow to infer domain knowledge to clarify this distinction.

Summing up, there are four main challenges for the abstraction of event logs to predefined activities. All four challenges are currently not sufficiently tackled by prior research. Therefore, new means to abstract event logs have

to be defined.

2.2. Processes and process execution

Having defined the problem that we want to solve, we start with the preliminaries on which we ground our work. A process model P contains a non-empty set of activities, which we denote as A. The process model defines a control flow relation $CF \subseteq A \times A$ and the function $subAct : A \rightarrow \mathcal{P}(A)$ defines the hierarchy relation between activities and their subordinated activities, called sub-activities.

At the top of Fig. 2, a process model is depicted with five activities, $A = \{a, b, ba, bb, c\}$. The activities ba and bb are sub-activities of activity b, $subAct(b) = \{ba, bb\}$. A process model might be further described with textual descriptions entailing execution details for each activity. The function

$$desc(a) = \begin{cases} d & \text{if } subAct(a) = \emptyset \\ d \cup \bigcup_{b \in subAct(a)} desc(b) & \text{otherwise} \end{cases}$$

assigns each activity the set of corresponding activity descriptions, including those linked to sub-activities. The function $desc(P) = \bigcup_{a \in A} desc(a)$ returns all activity description of a process P. Looking at the example in Fig. 2, we have $desc(P) = \{d_1, d_2, d_3, d_4, d_5\}$ and, e.g, $desc(b) = \{d_2, d_3, d_4\}$.

When an activity $a \in A$ is executed, we call this an instance of a. An instance of a is denoted as $\hat{a}$ and the set of all activity instances is called $\hat{A}$. The function $\gamma : \hat{A} \rightarrow A$ captures the relationship of an activity instance to its activity class, so we can write $\gamma(\hat{a}) = a$. A process instance $\hat{p}$ is a sequence of activity instances, i.e. $\hat{p} \in \hat{A}^*$, where all activity instances belong to a particular case. As one activity can be executed several times in one process instance, we number the activity instances with an index $i \in \mathcal{N}$. An example process instance is depicted in Fig. 2. This process instance can be written as $\hat{p}_1 = \langle \hat{a}_1, \hat{b}_1, \hat{c}_1, \hat{c}_2 \rangle$. Note that we omit the instances of the sub-activities of b to avoid redundant information. Depending on the abstraction level we are interested in, we could also write $\hat{p}_1 = \langle \hat{a}_1, \hat{ba}_1, \hat{bb}_1, \hat{c}_1, \hat{c}_2 \rangle$. To further reason about activity instances in a process instance, each activity instance has a life cycle, which consists of a set of states LS and a set of transitions LT between these states. In this work we assume the set of transitions to be $LT = \{start, execute, suspend, resume, complete\}$, but we do not depend on the actual specification of transitions. A generic activity life cycle is defined by van der Aalst in [1]. The life cycle of an activity instance

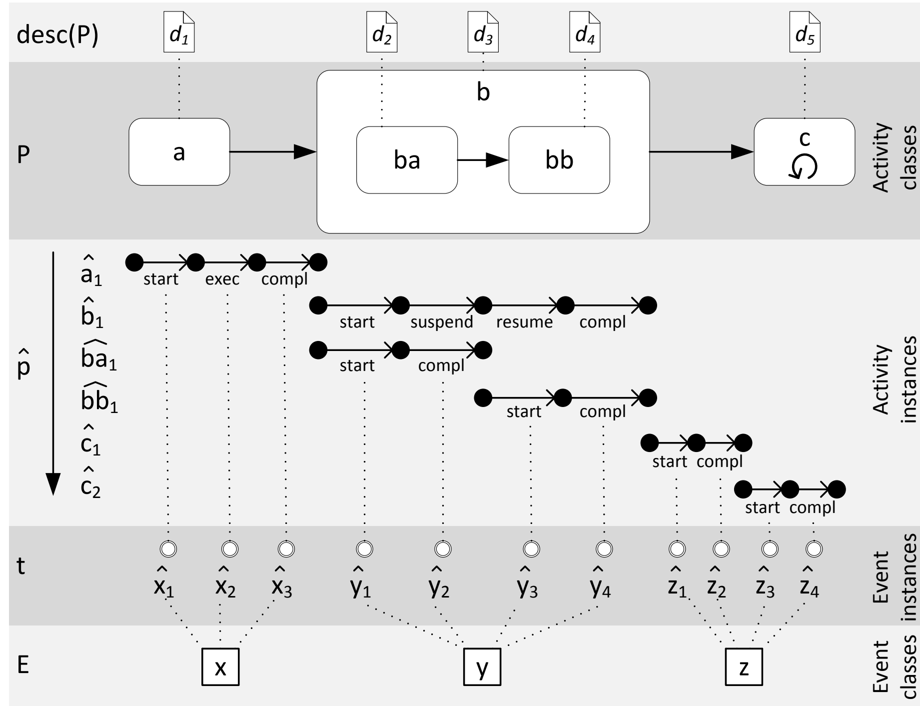

Figure 2: Process model, instances, and relations to events

allows to make detailed analysis of activity durations and idle times. While activity instances in a process instance have a clear ordering, their life cycle transitions can overlap, signaling concurrent execution.

2.3. Event logs

An IT system that supports process executions typically records events for each process instance. An event in our example process refers to the change of a data field in the supporting IT system as for example the setting of the classification. We refer to the type of an event as event class e. The set of all event types is called E. We denote a single instance of an event class e as $\hat{e}$ and we call the set of all event instances $\hat{E}$. Each event instance has different attributes assigned to it. For each attribute $\#_{attr}(\hat{e})$ refers to the value of the attribute $attr \in ATE$ for the event instance $\hat{e}$. ATE is the set of all possible event attributes. We assume that an event instance has at least the attribute $\#_{time}(\hat{e})$, which refers to the time when it occurred, and the attribute $\#_{class}(\hat{e})$, which links to the event class. Other attributes may

contain role information or values of changed database fields. A trace t is a list of the form $\hat{E}^*$, where the contained event instances are ordered by their *time* attribute. Every event instance is assigned to a trace and each trace in an event log is related to a process instance. Note that the relation of event instances to traces might not be trivial in every practical setting. Yet, there is plenty of work on event correlation that tries to relate event instances to traces (see e.g. [9, 10]). In this work we therefore assume that this relation is already given. Although each trace can contain multiple instances of the same event class, each event instance in a trace can be uniquely identified by its attribute values. Traces might be split for analysis of particular parts of a trace. A continuous part of a trace is called a sub-trace. We use the $\|$ operator for the concatenation of traces or event instances. Looking at the example in Fig. 2, we could split the trace after the event instances belonging to the execution of activity a by writing $t = \langle \hat{x}_1, \hat{x}_2, \hat{x}_3 \rangle \| \langle \hat{y}_1, \hat{y}_2, \hat{y}_3, \hat{y}_4, \hat{z}_1, \hat{z}_2, \hat{z}_3, \hat{z}_4 \rangle$.

Like events, traces have attributes. The set of all possible trace attributes is called ATT and $\#_{attr}(t)$ refers to the value of a trace attribute $attr \in ATT$. Regarding trace attributes, we only assume the attribute $\#_{case}(t)$, which uniquely relates a trace to a process instance, as mandatory. Nevertheless, more attributes can be defined and used in the event log abstraction.

The events created by an IT system during the execution of a process instance are typically stored in an event log, which contains a trace of event instances for each process instance. Thus, an event log L is a set of traces $t \in L$. We denote the set of all possible traces as T.

2.4. Relations of processes and event logs

Having a process model P and an event log L, the challenge (2) refers to the derivation of the relation of the activities $a \in A$ and the event classes $e \in E$. As events are supposed to be on a lower abstraction level, multiple event classes may belong to one activity. Due to shared functionalities – as defined in challenge (3) – also one particular event class can refer to several different activities. Thus, we are looking for the relation between event classes and activity types, $EA \subseteq A \times E$, which is essentially a n:m relation. Finally, as we are interested in abstracting event logs, and thus, in relating event instances to activity instances, the ultimate goal is to derive the relation of event instances $\hat{e} \in E$ and activity instances $\hat{a} \in A$. More specifically, we look for the relation of event instances to the particular life cycle transitions of an activity instance. Hence, the objective is to derive the relation $\hat{A}L\hat{E} \subseteq \hat{A} \times LT \times \hat{E}$.

3. Abstracting event logs using semi-automatic matching

In this section the approach for abstracting events to process model activities is introduced. The approach consists of four distinct phases, each addressing one of the four challenges introduced in section 2.1:

1. Annotation of process model activities,
2. Matching of activities and events on type level,
3. Definition of context-sensitive mappings, and
4. Clustering of event instances to activity instances.

Figure 3 shows the four phases with their inputs and outputs. In the first phase the process model activities need to be annotated with more details from the textual descriptions. In the second phase, the potential relations are computed using the annotations. These automatically derived relations have to be refined by a domain expert as input for the next phase. The third phase requires the user to clarify how to dissolve n:m relations that arise from shared functionalities. This phase also provides the option to remove events or complete traces that are not interesting for the aimed abstraction level and results in a preprocessed event log where each event instance is related to its corresponding activity type. In the last phase, the event instances, which are already mapped to activity types, are clustered to activity instances resulting in an abstracted event log.

3.1. Annotation of process model activities

As a major challenge in event-activity matching is the diverging level of abstraction, we utilize annotations. These annotations serve the purpose of enriching the coarse-granular activities of the process model with detailed information that helps to link to events. In modern business process modeling tools, activities can be connected with more detailed textual descriptions, such that the annotation of the activities is readily available. Often, instructions can also be found in tabular form consisting of columns for the activity name and the detailed description. In the following, we assume that such a description is available or can be directly linked to an activity.

In order to effectively use the activity descriptions for the matching of event classes and activity types, we have to pre-process the descriptions. As events represent some kind of change to an object, we are especially interested in the objects contained in the activity descriptions. Therefore, the Stanford Part-of-Speech (POS) tagger [11, 12] is used to filter out these objects. The

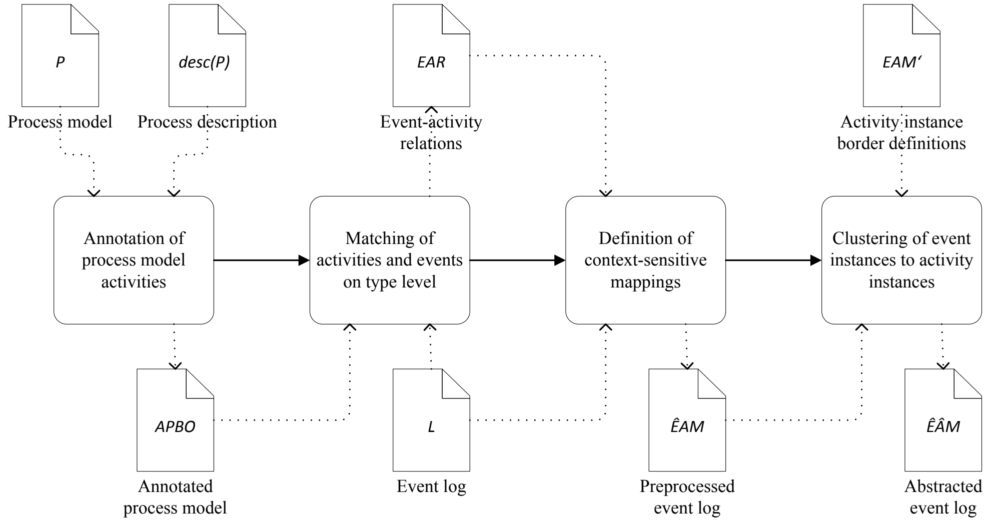

Figure 3: Overview of the 4-phase abstraction approach with inputs and outputs of each phase

POS tagger parses natural text and assigns each word to its part of speech, e.g. verb, noun, article, adjective, etc. From these categories we only take into account words that are nouns or words for which no real category can be found by the POS tagger. The latter are most often abbreviations as e.g. "CI" or foreign words. Furthermore, all numbers are filtered out. The goal is to extract potential business objects. The set of all potential business objects is denoted as PBO. $PBO_a \subset PBO$ is the set of potential business objects $pbo_i \in PBO_a$ that unites all potential business objects for an activity $a \in A$. These objects are extracted from all activity description $d_i \in desc(a)$. Additionally, the labels of the activities are processed in the same way to extract further potential business objects. The activities are annotated with the derived objects for further processing in the next phase of the approach. The result of this phase is an activity annotation relation $APBO \subseteq A \times PBO$. This relation is a n:m relation, i.e. one activity can be linked to multiple potential business objects, and one potential business object can be associated with multiple different activities. Note that the annotation is not mandatory for each activity. Yet, it presumably improves the automated matching result because the textual descriptions are likely to be closer to the abstraction level of the event log than the activities in the process model as we will show in our evaluation in Section 4.

3.2. Semi-automatic matching of events and activities

Having annotated the activities with their potential business objects, the second phase deals with challenge (2), the derivation of the event-activity relation EA. To this end, we inspect each combination of event class and activity name as well as each combination of event class and activity description for potential correspondences.

Algorithm 1: Derive potential event-activity relation

```
1: checkRelation(EventLog L, ProcessModel P, ProcessDescription desc(P))
2: Set EA := ∅
3: Set APBO := ∅
4: Set EPBO := ∅
   {Annotate activities with potential business objects}
5: for all a ∈ A do
6:    Set PBO_a := PBO_a ∪ extractNouns(a)
7:    for all d_i ∈ desc(a) do
8:       Set PBO_a := extractNouns(d_i)
9:    end for
10:   APBO := APBO ∪ {(a, pbo) | pbo ∈ PBO_a}
11: end for
   {Annotate event classes with potential business objects}
12: for all e ∈ E do
13:    Set PBO_e := extractNouns(e)
14:    EPBO := EPBO ∪ {(e, pbo) | pbo ∈ PBO_e}
15: end for
   {Check matches on business object level}
16: for all epbo ∈ EPBO do
17:    for all apbo ∈ APBO do
18:       if checkBusinessObjectMatch(pbo_a, pbo_e) then
19:          EA := EA ∪ {(e, a)}
20:       end if
21:    end for
22: end for
23: return EA
```

Algorithm 1 details the general procedure. The algorithm takes an event log, a process model and a process description. First, we derive the sets of potential business objects for activities and events. The function *extractNouns*

uses the Stanford POS tagger to return all potential business objects as explained above. In order to check for potential correspondences, we also derive the objects from the event classes in the same manner, yielding the relation $EPBO \subseteq E \times PBO$ (line 12–15). Each tuple in $APBO$ is compared to each tuple in $EPBO$ by comparing the business objects (line 16-22). As we aim for a high recall, we do not only make simple string comparisons in order to check the relatedness of two business objects. We employ natural language processing techniques in order to maximize recall as we will explain in the following.

As we evaluate our approach in a context using the German language, we need to pay special attention to this language. Nonetheless, the basic techniques are also available for many other languages. Looking at the German language we face two potential challenges: word form variance and compound words. German is a morphological complex language having a high variance in word forms expressed by many cases and inflections (cf. [13]). Looking at nouns, for example the word "Buch" (book) transforms to "Bücher" in the plural form or to "des Buches" for the genitive case. Regarding compound words, in German these are single words created by concatenating several words to a new word, e.g. "Fach|gruppe" (professional group).

In order to address these two challenges, two techniques from the natural language processing (NLP) area have been proven beneficial: stemming and word decomposition [14]. Stemming refers to the reduction of derived word forms to a common stem, e.g. "Grupp" for "Gruppe" and "Gruppen". In the implementation of our approach we use the stemming functionality of the Apache Lucene project[1]. For the decomposition of compound words, we use a language independent, lexicon-based approach developed by Abels and Hahn [15]. It generates possible splittings of words and checks whether the generated parts are covered in a lexicon. In our approach we use JWordSplitter, an open source implementation of this approach with an integrated German lexicon[2].

Algorithm 2 illustrates the procedure to check for a relation between two business objects. First, we conduct a simple string match (line 2) and second, we decompose the business objects into their smallest semantic components and compare these with one another (line 5-14). The comparison of decom-

[1]See `http://lucene.apache.org`.

[2]See `http://www.danielnaber.de/jwordsplitter/`.

Algorithm 2: Check for business object matches

```
1: checkBusinessObjectMatch(pbo_1, pbo_2)
2: if eventObject = textObject then
3:    return true
4: else
5:    Set wordParts_1 := decompose(pbo_1)
6:    Set wordParts_2 := decompose(pbo_2)
7:    for all wp_1 ∈ wordParts_1 do
8:       for all wp_2 ∈ wordParts_2 do
9:          if stem(wp_1) = stem(wp_2) then
10:            return true
11:         end if
12:      end for
13:   end for
14: end if
15: return false
```

posed word parts is done by comparing the word stems. In this way we are able to relate words like "Fachgruppe" (professional group) and "Skillgruppen" (skill groups).

The result of the described steps is an automatically provided list of potential event-activity relations $EA \subseteq E \times A$ on type level that can be refined by a domain expert. Refining in this case means that the expert has to identify and remove incorrect relations and add relations that are missing. As we do not only aim at mapping events to activities, but also to the specific life cycle transitions of the activities, the domain expert furthermore needs to add the corresponding life cycle transition to the tuples in EA. The *start*, *complete* and *execute* transitions can often be automatically detected during the actual mapping of event instances to activity instances as we will explain later. Hence, we do not require that these transitions are manually related to the event-activity relations. We therefore introduce a placeholder transition $\tau \in LT$ that will be automatically replaced by either *start*, *complete* or *execute*. This results in the relation ALE as defined in Definition 1.

Definition 1. *(Event class to activity life cycle relation) The relation $ALE \subseteq EA \times LT$ relates event classes to the life cycle transitions of activities for which they signal their execution. The set LT is extended by the life cycle*

transition τ*, which acts as a place holder for the set of automatically derivable life cycle transitions* $\{start, complete, execute\}$*. Thus, the set of life cycle transitions is* $LT = LT \cup \{\tau\} \setminus \{start, complete, execute\}$*.*

3.3. Building context-sensitive event-to-activity mappings

Based on the approach reported above, we find the event-activity relations. Yet, as formulated in challenge (3), there are often events representing shared functionality used by multiple activities. Hence, the event-activity relations can not be directly used for the abstraction as we have to disambiguate the event instances for which there are multiple relations of their corresponding event class in ALE. This section describes the necessary steps to get from the relations to a concrete event-to-activity mapping that can be used to abstract the event log.

The challenge in this context is to identify the conditions that help to decide when one event class matches one of alternative activities. To this end, we consider the context of an event instance, either as defined over the event or trace attributes or over the surrounding event instances. First, we take *attributes* into account. For instance, the role that is related to an event might be important to distinguish different activities in the process model. In Figure 1, the selection of a configuration item (CI) belongs to the activity "Initial diagnosis" when executed by a first level agent while the same event class refers to the activity "Investigation and diagnosis" when executed by a second level supporter. Second, the relation of an event instance to an activity might also depend on the *context* in terms of preceding or succeeding event instances. While the selection of a CI normally happens during the activities "Initial diagnosis" or "Investigation and diagnosis", depending on the executing role, it can also be performed as a quality improvement step during the closure of the incident ticket. As shown in Figure 1, this is always the case if the solution has been documented before. Also, an event might be interpreted differently if it occurs for the first time or if it has been preceded by earlier executions. In the example in Figure 1, the working group is always set in the beginning where it refers to the logging of the incident while every other change of the working group refers to the functional escalation.

In order to use such domain knowledge, we have to encode it in a formal way. Definitions 2 and 3 introduce the formalization of attribute conditions and event context conditions.

Definition 2. *(Attribute condition) Let* $AT = ATE \cup ATE$ *be the set that unites all event and trace attributes. Let* O *be the set of comparison operators and let* V *be the set of values that an attribute* $attr \in AT$ *should be tested against. Then, an attribute condition is a tuple* $ac \in AT \times O \times V$*.* AC *is the set of all attribute conditions. An attribute condition* ac *is evaluated for an event instance by the function* $EV_{ac}(ac, \hat{e}) = o(\#_{attr}(\hat{e}), v)$*, where* $o \in O$ *is a boolean function that compares two given input values, and* $v \in V$ *is the value given by the attribute condition.*

Definition 3. *(Event context condition) The event context* EC *for an event instance* $\hat{e}$ *is defined as* $EC(\hat{e}) = (t_{before}, t_{after})$ *such that* $\exists t \in T : t = t_{before} \| \hat{e} \| t_{after}$ *where* t_{before} *and* t_{after} *are sub-traces of trace* t*. The sub-traces can be accessed by the function* $EC(\hat{e}, r) \rightarrow T$*, where* $r \in \{before, after\}$ *refers to the part of an event context* EC*.* $EC(\hat{e}, before)$ *returns the sub-trace* t_{before} *and* $EC(\hat{e}, after)$ *returns the sub-trace* t_{after}*.*

A condition over a trace is defined by a function $f(t) \rightarrow \{true, false\}$*, which evaluates a linear temporal logic (LTL) formula [16]. The set of all LTL formula functions is referred to as* F*.*

An event context condition is a tuple $ecc \in F \times \{before, after\}$*.* ECC *is the set of all event context conditions. An event context condition* ecc *is evaluated for an event instance by the function* $EV_{ecc}(ecc, \hat{e}) = f(EC(\hat{e}, r))$*.*

When shared functionalities are discovered in the first matching phase, the user needs to define the necessary attribute and event context conditions in order to dissolve the assignment problem. Assume that all event instances have the attribute *role* and that the set of available comparison operators $O = \{equals, contains, startswith\}$. A context conditions to identify when an event instance of the event class “CI” belongs to the activity “Initial diagnosis” could be written as (’role’, ’equals’, ’first level’). To identify the cases when an instance of the event class “CI” belongs to the activity “Incident closure” we can create the event context condition ($\Diamond$(’Document solution’), ’before’). The $\Diamond$(’Document solution’) is an LTL formula that stands for “eventually activity ’Document solution’ occurs”. LTL has been chosen as it gives good flexibility for defining even more advanced conditions. In order to check the LTL statements we use the functionality of the ProM LTL Checker plug-in as introduced by Aalst et al. [17].

It is also possible that multiple conditions need to match in order to map an event instance to an activity class. Thus, we define a context condition

c as the conjunction of attribute conditions and event context conditions as presented in definition 4.

Definition 4. *(Context condition) A context condition c is a tuple $c = (AC', ECC')$ with $AC' \subseteq AC$ and $ECC' \subseteq ECC$. The set of all context conditions is denoted as C. The evaluation function checks whether all defined conditions hold for an event instance. It is defined as*

$$EV(c, \hat{e}) = \begin{cases} \mathit{true} & \forall ac \in AC'\colon EV_{ac}(ac, \hat{e}) = \mathit{true} \wedge \\ & \forall ecc \in ECC'\colon EV_{ecc}(ecc, \hat{e}) = \mathit{true} \\ \mathit{false} & \mathit{otherwise} \end{cases}$$

While the conditions introduced in Def. 4 allow to mix attribute and context conditions, practical settings often require to check whether a certain event has happened before and fulfills a certain attribute condition. For example if the mapping of an event depends on the current priority of the case and the priority can dynamically change. We therefore introduce global event attributes that are added to each event instance. The values of these attributes are updated by the attribute values of occurrences of event instances of specific classes that are defined in the global attribute relation $GATR \subseteq E \times ATE$. Assuming that the event instances of the class "Priority" have an attribute called "value" that stores the priority that has been set, the tuple for this example would be specified as ("Priority", "value").

Having defined the context conditions, we introduce an event class to activity mapping EAM based on event classes and conditions that have to be fulfilled for a corresponding event instance in order to be mapped to a specific life cycle transition of an activity.

Definition 5. *(Event class to activity mapping) An event to activity mapping is a relation $EAM \subseteq ALE \times C$, which relates an event class to a life cycle transition of an activity type based on a set of attribute conditions and a set of event context conditions.*

Definition 5 gives the mapping between activities from a process model and event classes found in an event log. For our examples the event class to activity mapping could be defined by a domain expert as
{('CI', 'Incident closure', τ, ({}, {($\Diamond$('Document solution'), 'before')})),
('CI', 'Initial diagnosis', τ, ({('role', 'equals', 'second level')},

$\{(!\Diamond($'Document solution'$)$, 'before'$)\}))$,
('CI', 'Investigation & diagnosis', τ, $(\{($'role', 'equals', 'second level'$)\}$,
$\{(!\Diamond($'Document solution'$)$, 'before'$)\}))\}$.

A special case of an *EAM* is when an event class potentially belongs to every activity in the process model. One example for this are protocol events, which signal that somebody documented what they did. Another example we encountered in practice are status events, which show if somebody is currently working on a case or if it is on hold. These events show the life cycle of the individual activities and can for instance be used to calculate idle times within the execution of an activity. Such events typically belong to the activity that is currently executed or about to start. In order to achieve such a dynamic mapping, we introduce a special activity place holder that can be used in the *EAM* mapping definition. This place holder is called *CLOSEST_ACTIVITY* and will be processed after all other mappings. It signals the mapping algorithm to assign an event instance to the activity to which its closest neighbor is assigned to. The distance for the determination of the closest event is measured over the time stamps of the event instances.

Another special case for abstracting events to activities, is the omitting of insignificant events. While some of the existing abstraction approaches, like [5], implement this type of abstraction by automatically hiding events from infrequent event classes, we observed that it is often better to let a domain expert control what to omit and what to keep. Sometimes, events that occur very often are not interesting from a business point of view and infrequent events can be highly important once they occur. In some cases one might also not want to omit all events belonging to a specific event class, but only those event instances fulfilling certain conditions. An example for this are message events where only messages with a certain priority or type should be kept. We therefore introduce another activity place holder for *EAM* called *REMOVE_EVENT*. In the same fashion, it can be helpful to abstract from complete traces that contain certain behavior identified by event instances in a certain context. The activity place holder for removing complete traces is called *REMOVE_TRACE*. For example one might be interested in abstracting away from cases that have been reported by monitoring systems and can be identified by the fact that the “Details” event occurs before the first “Group” event as in case 6 in Fig. 1.

Having established the relations between event classes and activities on the type level, we can turn to the instance level. Therefore, we specify a func-

tion $\hat{E}AM$ that maps event instances to the corresponding activity classes for which all defined conditions hold. Note that normally, this set consists of only one activity. Only in the case where one event instance signals the execution of multiple activities, the size of the set will be larger than one.

Definition 6. *(Event instance to activity mapping) The relation* $\hat{E}AM \subseteq \hat{E} \times A \times LT$ *is the mapping of event instances to life cycle transitions of activity types, for which* $(\hat{e}, a, lt) \in \hat{E}AM \implies \exists (e, a, lt, c) \in EAM : e = \#_{class}(\hat{e}),\ Ev(c, \hat{e}) = true$.

Definition 6 covers 1:1, 1:n and n:m relations on the instance level. A 1:1 mapping on instance level only occurs for events and activities that are on the same abstraction level. Looking at different abstraction levels, it is most likely that an activity instance on a higher level subsumes multiple event instances representing different sub-activities. Thus, in most cases we face a 1:n mapping on instance level and event instances will be clustered to activity instances. Nevertheless, it can be the case that one event instance reflects multiple activities. For example, when an incident needs to be resolved with a change in the IT infrastructure, one has to document the necessary steps as well as a back-out plan for the case something goes wrong during implementation. When the supporting IT system only reserves one field for these two texts, it will also only save one event for the change of this field. If the process model distinguishes the writing of the plan and the writing of the back-out plan in two activities, we have one event representing two activities. By defining conditions over the content of the protocol attached to the event instance as attribute, one can find out whether both activities have been executed, e.g. by searching for keywords like “back-out”.

Using the defined mappings in EAM, we iterate over the traces in a log and assign each event instance the name of its corresponding activity and the defined life cycle transition. In case multiple mappings match for an event instance $\hat{e}$ – as in the protocol example above – we duplicate $\hat{e}$ as many times as needed. In order to keep the relation to the original event class, we introduce a new event attribute $\#source(\hat{e})$ that contains the original event class. We refer to the attribute $\#source(\hat{e})$ as source event class. The result of this phase is a preprocessed event log where all event instances are assigned to their corresponding activity.

3.4. Clustering events to activity instances

Having mapped all event instances to the life cycle transitions of their corresponding activity type, the next step is to define how to assign event instances belonging to the same activity type to activity instances. As there might be multiple activity instances for one activity in a process instance, i.e. in a loop, challenge (4) raises the question which criteria are used to map an event instance $\hat{e}$ to an activity instance $\hat{a}_i$. In this case, we need to define the border between events belonging to two or more instances of the same activity. We therefore introduce the notion of instance border conditions.

Definition 7. *(Instance border condition) An instance border condition defines whether an event instances belongs to the set of event instances mapped to an activity instance $\hat{a}_i$ or if it belongs to another activity instance $\hat{a}_{i+1}$. It is defined as a boolean function $bc : \mathcal{P}\left(\hat{E}\right) \times \hat{E} \rightarrow \{true, false\}$. The set of all border conditions is BC. Each tuple in EAM is extended by a border condition function such that $EAM' \subseteq EAM \cup BC$.*

Instance border definitions relate to two levels: intra and inter activity structure. Concerning the *intra activity* structure, we have to decide whether there are loops in activities on the sub-activity level or not. While the assumed process model might not contain loops, this does not imply that there are no loops on the *inter activity* level in the execution. These have to be lifted to activity level if we assume there should not be any loops on sub-activity level. In line with this assumption, an activity instance border is marked by the repetition of source events from the same event class, i.e. the repetition of a source event class signals that a new activity instance has started. Thus, for example two protocol events could indicate rework and therefore two instances of the corresponding activity.

Using recurring source event classes as instance border definition works only if there are no loops in the assumed sub-activity model. If there are *loops* on the intra and inter activity level, multiple event instances from the same event class might belong to one activity instance. A typical example for this is a loop over the order items in an order list where different activities like "Choose supplier" have to be performed for each order item and are modeled on activity level. The activity "Choose supplier" might contain different sub-activities that have to be executed for each supplier, like e.g. "Check prices". Thus, we have a loop on activity level and a loop on sub-activity level. In order to find the correct instance borders, we need to extend the instance

border definition to also use different business objects, e.g. the order line, as instance border markings. Thus, instance borders can be defined over any attributes attached to an event.

If necessary attributes are not available or if it is not possible to make statements about the assumed sub-activity model, we need to use *heuristics* in order to be able to identify different activity instances. Similar to the approaches of Li et al. [7] and Günther et al. [8, 6], a first heuristic for an instance border can be defined based on a threshold for the maximum distance between two events that belong to one activity instance. While previous works only focus on a very narrow and short distance based on the number of events in between two events, we extend this definition using also the time perspective, i.e. defining how long the time frame between two event instances of the same activity instance can be. For example one might limit the time distance between two events of the same activity instance, e.g. two edit events for a protocol belong to different activity instances if there are more than 24 hours between them. Using a maximal number of events that are allowed to occur between two events of the same activity instance still makes sense as a heuristic from our practical experience. Yet, it needs to be specified by the user and should also allow larger distances when there are many events assigned to single activities and concurrency cannot be precluded. In a similar manner, another heuristic can be build on the assumption of a maximum number of event instances that belong to one activity instance. Here, a domain expert has to estimate how many sub-activities are approximately executed per activity instance. This is simple, if we can exclude loops on sub-activity level, but might be difficult otherwise. In the former case, one would limit the the maximum number of event instances to the number of assigned event classes for an activity.

The defined instance border conditions are used to establish the relation $\hat{E}\hat{A}M$, which maps event instances to life cycle transitions of activity instances as specified in Definition 8.

Definition 8. *(Event instance to activity instance mapping)* $\hat{E}\hat{A}M \subseteq \hat{E} \times \hat{A} \times LT$ *is the relation that assigns an event instance to the life cycle transition of its corresponding activity instance* $\hat{a}$*. The function* $\lambda : \hat{A} \rightarrow \mathcal{P}(\hat{E})$ *returns all event instance related to an activity instance in* $\hat{E}\hat{A}M$*. It holds that* $(\hat{e}, \hat{a}, lt) \in \hat{E}\hat{A}M \Rightarrow \exists\, (\hat{e}, a, lt) \in \hat{E}AM : (a = \gamma\,(\hat{a})) \wedge bc\,(\hat{e}, \lambda\,(\hat{a}) \setminus \{\hat{e}\}) = \mathit{false}$ $\wedge\ \exists\, (e, a, lt, bc) \in EAM : (e = \#_{class}(\hat{e})\,,\ a = \gamma\,(\hat{a}))$*. An event instance* $\hat{e}$ *that is mapped to an activity instance* $\hat{a}$ *is referred to as source event of* $\hat{a}$*.*

Based on these pieces of information, we can approach the clustering task. In order to transform a given event log to a higher abstraction level, we iterate over the traces of the preprocessed event log where all event instances are mapped to their corresponding activity using the relation EAM as explained in Section 3.3. The events assigned to the same activity class need to be clustered to activity instances adhering the instance border definitions as defined by EAM'. We use a tree-based incremental clustering algorithm known from classical data mining [18]. For every activity class the clustering forms a tree with the event instances as leaves and the activity instances on the next higher level. The clustering starts with an empty tree and event instances are incrementally inserted. Updating the tree is done by finding the right place to put the new leaf, potentially triggering a restructuring if an instance border is found. The best host for a new event is the event with the minimal distance that can be expressed by a distance function using e.g. the time stamps of the events or other attributes. Having found the optimal host, we have to check the border conditions for all events belonging to the activity instance. If no instance border is found, the event is added to the activity instance cluster of the determined host event. Once an instance border is found, we need to determine where the new activity instance starts. This is done using a goodness function based on the summation of distances within a cluster. The goal is to find the optimal clustering with the minimal sum of distances between events belonging to the same activity instance cluster. For further explanations of the instance clustering algorithm, we refer the reader to our previous work [2].

Having all event instances assigned to their corresponding activity instances, the next step is to identify the life cycle transitions of the activity instances that are mapped to the τ transition. Therefore, the first and last event instance of a cluster are assigned the *start* and *complete* transition, respectively. All other event instances mapped to the τ transition are assigned to the *execute* transition. As the *execute* transition does not give any additional information for analysis on activity level, we remove all event instances assigned to this transition from the abstracted log. While we abstract from event instances assigned to the transition *execute*, these event instances are still of interest when it comes to the analysis of individual activities, i.e. the sub-activity level. The clustering algorithm therefore returns two results. The first result is the abstracted event log, which only contains event instances assigned to transitions other than *execute*. The second result is a new event log for each individual activity, which contains the corresponding

	Complete month	Extract Raw	Extract Mapped	Extract Abstracted
Event classes	39	33	25	25
Cases	16,922	401	401	401
Variants	16,655	309	125	103
Event instances	545,996	11,875	11,375	9,364

Table 1: Event log statistics for the incident process

activity instance clusters as traces. These logs can be used to analyze the individual activities, e.g. by mining the sub-activity model for a specific activity. Using special mining plugins as e.g. introduced in [19] one can utilize these activity logs to interactively zoom into the different abstraction levels.

4. Evaluation

In order to validate how well our approach works with real life data, we conducted two case studies using data from the incident and change management process of a large German IT outsourcing project. Within the case studies we evaluated all four phases of our approach with respect to the four challenges described in Section 2.1. Based on these challenges we formulate evaluation hypotheses. In line with the first challenge (1), we posit that (a) the usage of external knowledge improves the retrieval of event-activity relations on type level. Referring to the second challenge (2), we assert that (b) our matching approach is able to identify most of the event-activity relations automatically. Moreover, we claim (c) that events of shared functionalities can be successfully disambiguated as asked by challenge (3). Tackling challenge (4), we propose that (d) loops and concurrency can be handled well by the clustering mechanism of our approach. Apart from these four hypotheses that arise from the main challenges of event log abstraction, another very important claim that has to be evaluated is that (e) accuracy of the abstraction result significantly impacts process analysis tasks such as conformance and performance analysis. This point is especially important, as it asks for the actual practical impact of a correct abstraction of an event log.

Before we start with the evaluation of the five hypotheses, we provide some background information for the two processes that have been used. Both processes are well documented with process models and work instructions and both processes are supported by the ticketing software IBM Tivoli Service Desk. The process model for the incident management process has 41 activities and the corresponding event log contains about 17,000 cases,

	Complete month	Extract Raw	Extract Mapped	Extract Abstracted
Event classes	55	42	37	37
Cases	2,005	17	17	17
Variants	1,947	15	16	13
Event instances	125,337	1,151	859	748

Table 2: Event log statistics for the change process

39 event classes and a total of about 550,000 event instances for a selected month. For the change process, the model contains 63 activities and the event log about 2,000 cases, 55 event classes and about 125,000 event instances. For cases our approach has been applied and manually checked by the responsible process managers. This resulted in an abstraction gold standard for a part of the logs. Table 1 and 2 show the size of the extracted part on which the gold standard abstractions have been built. Furthermore, it also shows the numbers for the two main phases in the transformation of the event log. Looking at these numbers, it can be seen how complexity is reduced during the abstraction phases. Event classes are eliminated, event instances clustered and thereby the overall variance is significantly reduced.

For the purpose of evaluation, our approach has been implemented as a set of two plugins in the process mining workbench ProM[3]. In order to process the events stemming from the ticketing system, we extracted them from the underlying database into a CSV file[4] using standard SQL. Each line in the CSV file represents one event instance with all its attributes. From the CSV file we converted the event data to the XES event log format in order to import it into the ProM framework (See [20]). This conversion can be easily done with existing tools as e.g. Disco[5].

The extracted event log is the starting point for our abstraction and the following sections will guide through the phases of our approach and evaluate the formulated hypotheses in the light of the conducted case studies.

4.1. Evaluation of automated event-to-activity matching

In the first phase, we had to link the process model activities with the corresponding work instructions for both processes. The work instructions have been provided in word format in tabular form as it is illustrated in

[3]See `http://processmining.org` for more information on ProM.
[4]Comma separated values
[5]See `http://www.fluxicon.com/disco/`

	Available activity descriptions	Model activities	Annotated activities	Used descriptions
Incident	238	41	31	64
Change	89	63	60	60

Table 3: Process model annotations with activity descriptions

Fig. 4. Therefore, the tables have been converted to CSV files and imported into the ProM framework using a dedicated import plug-in. Our main ProM plug-in automatically matches the activities in the process model with the work instructions over the given IDs that are also exemplified in Fig. 4. The plug-in uses the part-of-speech tagging facilities provided by the Stanford POS tagger to automatically extract the potential business objects from the activity descriptions. An overview of the number of annotated activities and used activity descriptions is given in Table 3. Although the process model for the incident process is smaller, a larger amount of textual description with a total of 238 activity descriptions is available. This is due to the documentation of sub-activities that are not part of the process model. The matching algorithm annotated 31 process model activities with 64 descriptions for the incident process. Yet, not all process model activities could be annotated due to missing descriptions. This already provides valuable information for the process manager. For the change process a smaller set of documenting descriptions is available containing at most one description per activity.

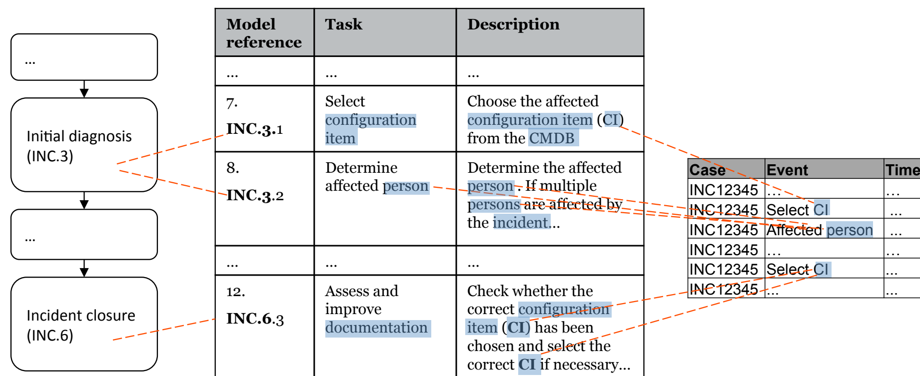

Figure 4: Link between process model activities, work instructions and events

Having annotated the process model activities with the potential business object stemming from the work instructions, we turn to phase two of our approach. In this phase, the ProM plug-in also extracts the potential business

objects from the event classes and automatically matches events and activities on type level as described in section 3.2. In order to assess hypothesis (a), Fig. 5 shows the number of correctly identified event-activity relations distinguishing whether the match has been found over the activity name or using the description. For both processes, it can be seen that the external knowledge accounts for a significantly higher share of correctly identified event-activity relations. Looking at hypothesis (b), we measure the *precision* (number of correctly matched event-activity pairs divided by all matched pairs) and the *recall* (number of correctly matched event-activity pairs divided by all manually matched event-activity pairs) [21]. Figure 6 presents the overall results for these two measures. It can be seen that a high recall of 70% and 86% is achieved, while precision is in a lower range. For the incident process, the precision of 28.62 % is mainly caused by matches that are based on the additional activity descriptions. Here, we achieve a precision of 26.48 % while the precision of matches on the activity names is high with 64.29 %. The precision for the change process is with 42.58 % substantially higher than the precision for the incident process. Here, the difference between the precision for matches on activity names and matches on the annotated description texts is small. However, description matches account for most of the overall recall, which yields with 70 % and 86.09 % a good result from a practical perspective. Hence, our claim (b) to be able to derive most of the event-activity relations is supported.

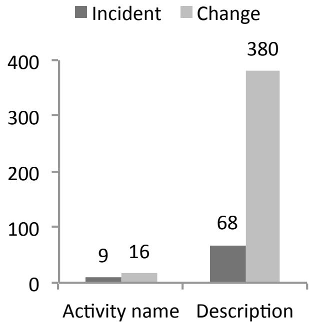

Figure 5: Correct matches by provenance

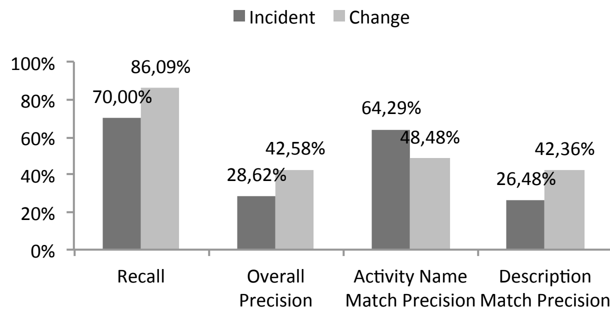

Figure 6: Recall and precision for automated matching

We also investigated why certain event-activity relations were not found. It turned out that the main reason is that the sub-activities represented by

the events where not documented in the work instructions. Some of these are simply missed out and need to be updated in the description. Here, the approach helped in identifying gaps in the documentation. The other fraction of the undocumented relations are steps that are automatically executed by the system and are therefore missing in the documentation. Future research might investigate in how far such relations can be retrieved from existing software documentations. Beside the undocumented relations, there are two other minor reasons why relations could not be found. First, some relations can only be found by interpreting event attributes. For example there is an event class "Kommunikationsprotokoll" (communication protocol), which contains all events for sent e-mail messages. Looking at the subjects of these messages, which are most often standardized, one could derive further relations. We also encountered one case where the relation could have been established using a verb instead of a business object. However, we did not include verbs in our approach as their inclusion leads to a drastic increase in false positives. Finally, we encountered some mappings that could have been found using synonym relations. However, these synonyms are of a domain-specific nature and not covered in general-purpose tools like WordNet.

4.2. Evaluation of disambiguation of shared functionalities

In order to evaluate (c) the disambiguation of shared functionalities, the event-activity relations have been manually finalized (end of second phase) and the mapping has been provided with context-sensitive mapping conditions by the domain experts (third phase). During the second phase, 22 shared functionalities, i.e. event classes that belong to several different activities, have been identified for the incident management process. The change process contained 40 event classes that reflect shared functionalities. Some event classes belong to more than ten different activities. One example for such a shared functionality used by many activities is the communication protocol. The correspondence for these e-mails can be distinguished using attribute conditions over their headers. For the incident management process 62 attribute conditions and 11 context conditions have been defined. As the change process contains more shared functionalities, the number of constraints is also higher. The domain expert defined 101 attribute conditions and 43 context conditions.

Figure 10 shows a screenshot of an excerpt of a mapping in our ProM plug-in. The plug-in allows for adding, editing and removing mapping definitions containing attribute and context conditions. The mappings can be stored

into and loaded from XML files. An example of the XML representation is shown in Fig. 11. Once all conditions have been specified, the plug-in takes the event log and the mapping definitions and produces a new event log where all event instances are mapped to their corresponding activity types. This intermediate event log has been manually checked by the domain experts and did not reveal any errors at this stage. Thus, our hypotheses (c), which states that shared functionalities can be successfully handled, holds true.

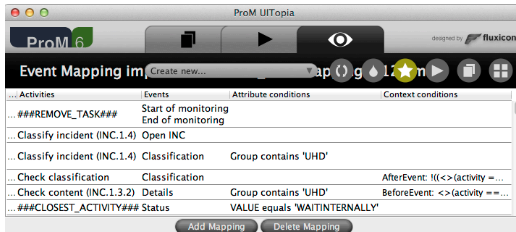

Figure 7: Screenshot of mapping in ProM

```
<EventMapping>
    <activity><name>Check content (INC.1.3.2)</name></activity>
    <eventName>Details</eventName>
    <InstanceBorder>
        <className>SourceEventLoopDifferentResource</className>
    </InstanceBorder>
    <metaDataCondition>
        <name>Group</name>
        <value>UHD</value>
        <type>contains</type>
    </metaDataCondition>
    <eventCondition location="BeforeEvent"><![CDATA[
        <>(activity == "Change of Group");
    ]]></eventCondition>
</EventMapping>
```

Figure 8: Excerpt of the mapping definitions in XML

4.3. Evaluation of activity instance clustering

Having the successfully evaluated result of phase three, we turn to the final phase that automatically clusters the assigned event instances to activity instances. For the evaluation of hypothesis (d), the handling of loops and concurrency, we compare the abstraction results for different activity instance border definitions. Therefore, the domain experts first had to specify the correct instance borders and afterwards, assess the resulting event log and check whether all instances have been mapped correctly. The result of this manual checking has two purposes. First, it shows how well the clustering approach works, and second, it is used to benchmark different instance borders used to mimic the handling of activity instances by other approaches.

Looking at the definition of the instance borders, we have to identify potential loops on inter and intra activity level. Both processes are supported by a form based web interface that allows for saving most of the fields edited by the user at any time. That means the user can enter text in a field, click the save button, enter more text, press save again, and so forth. For each

saving, events are created. This means that for most event classes a sequence of events of the same event class is not truly a repetition on activity level. However, in case two different users edit a field, this must be considered as repetition establishing a new activity instance border. As this is the case for most of the event classes in the two processes, this instance border is defined on a global level. Nevertheless, there are event classes where event repetition by different resources does not signal an activity repetition. Examples for this are status or progress events. Within one activity instance the status of the process instance might be set several times to waiting and back to working. Here, loops on the sub-activity level should not be lifted to activity level and thus, we declare that for these event classes no activity instance border exists. This means that the global activity instance border is overwritten on event class level in each tuple in EAM' that defines a mapping for such an event class.

Figure 9 shows the correct assignment of events to activity instances according to different definitions of activity instance borders. The very left bars show the results for the instance borders defined by the domain experts that were manually checked and found to be overall correct. The next best alternative to these manually defined instance borders is to define no activity instance borders at all, which means that event instances assigned to the same activity type are always clustered into one activity instance. This configuration yields around 90 % correctly assigned event instances for both cases. The explanation for this is on the one hand that the activities that are seen to be non-repeatable account for a large share of the event instances. On the other hand, it can be seen that there is not much repetition on activity level. Figure 9 also shows the results for the heuristic instance border that defines a maximal distance between two event instances belonging to the same activity instance. We have chosen the two configurations of a maximal distance of 1 and 2 as these are used by other abstraction approaches like in [6] and [7]. A distance of 1 means that there must not be any event of another activity in between two event instances belonging to the same activity instance, i.e. no concurrency. This configuration yields a fraction of around 3/4 of correct event instance for the incident process, but only half of the event instances could be correctly assigned for the event log of the change process. The distance of 2 allows for one event instance in between and yields slightly better results. These results show that the handling of concurrency and loops needs attention and it shows that our approach is able to handle both, as claimed in hypothesis (d).

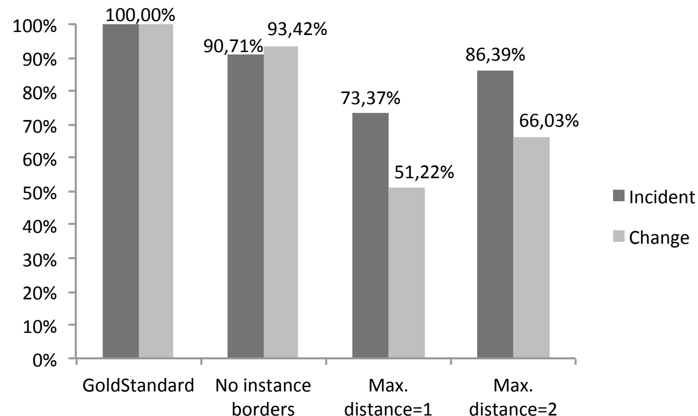

Figure 9: Fraction of correctly clustered event instances for different instance borders

4.4. Evaluation of impact on process analysis

To assess (e) the impact of different instance border definitions, we investigated their influence on conformance and performance analysis. First, the conformance to the designed process model is measured using the constraint-relative behavioral profile conformance metric as defined in [22]. We analyzed the conformance of the abstracted event log to the corresponding process model for the same definitions of activity instance borders as in the previous section and present the results in Figure 10. While the results for the incident process show only little variance, the conformance metric gives a maximal difference of around 8 % points between the lowest and highest result for the change process. Thus, a considerable influence of the correct activity instance clustering on conformance checking can be observed.

Looking at performance analysis, the duration for each activity instance has been measured as the difference between its start and end event. Figure 11 shows the difference of the average activity instance duration of the abstractions with the selected instance border definitions with respect to the gold standard. Here, the impact is even more striking with a maximum of almost 2 days of difference to the gold standard in the average duration. Moreover, a clear difference between the two processes can be seen. Again, differences are less high in the incident process. More interesting is that it can be clearly seen that for performance analysis there is a different ranking in the goodness of the different instance border definitions when comparing the results of the two processes. For the incident process the maximum

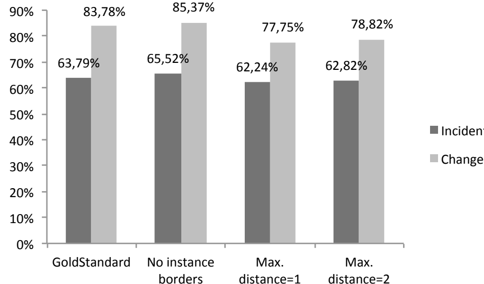

Figure 10: Conformance results for different instance border definitions

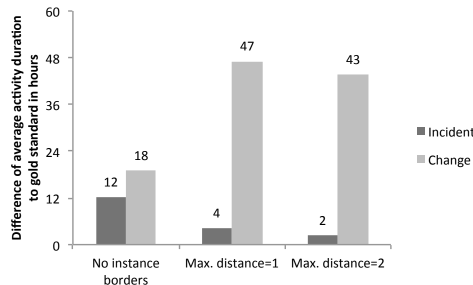

Figure 11: Differences in activity performance results for different instance borders in comparison to the gold standard

distance between events yields the best result. In the performance analysis of the change activities, no instance border turns out to be closer to the gold standard results in average. This shows how sensible these analyses are towards the right clustering.

5. Related Work

Research related to this paper can be generally subdivided into approaches working on event logs and approaches working on process models. The work that focuses on event logs can be mainly subdivided into event log abstraction and event correlation. The related techniques that work on process models fall into one of three categories: process model abstraction, process model matching and process model similarity. In both main categories – work on event logs and on process models – there are a few hybrid approaches that take both an event log and a process model as input. Yet, they always focus on either log or model when it comes to the objectives and the output of those techniques. Table A.4 in the appendix provides an in-depth classification of the most relevant related work. Besides the input of event logs and process models, we assessed two criteria: The usage of external knowledge and the possible types of relations on class level.

Relating to challenge (1), we examined whether external knowledge is used and if this is the case, we specify what kind of knowledge is used. Looking at the event log focussed techniques, almost no approach makes use of external knowledge. Only [23, 9] use sources code as external knowledge that is also manipulated by the technique in order to generate events that

can be correlated to process instances and used for process mining. Yet, these techniques do not address the other challenges. Process model techniques are more advanced when it comes to the usage of external knowledge. Especially in recent years, more sophisticated techniques that use linguistic information have evolved [24, 25]. Other model-based techniques – as e.g. in [26] – leverage different semantic information as e.g. roles, resources or data objects, which can be seen as external knowledge in this area. The approach presented in this paper tries to leverage different types of external knowledge. While we use linguistic information in the matching of events and activities, we furthermore make use of process descriptions to extend the available information about activities in the process model.

Turning to challenge (2), the possible relations between matched objects on type level, we found that most works in the area of event log abstraction support 1:n relations. Günther et al. introduce in [8] an approach that clusters events to activities using a distance function based on time or sequence position. Due to performance issues with this approach, a new means of abstraction on the level of event classes is introduced in [6]. These event classes are clustered globally based on co-occurrence of related terms, yielding better performance but lower accuracy. Still, both approaches are only able the relate a particular event class to one specific activity. Thus, they partially help to address challenge (2), but are not able to identify shared functionalities and therefore do not address challenge (3). The approach introduced by Li et al. [7] can handle n:m relations by defining context-dependent patterns. Yet, context in that work is limited to event classes of surrounding event instances, which only partly addresses challenge (3) as not all shared functionalities can be distinguished. Moreover, two events belonging to one activity instance cannot be separated by more than one event of another activity instance in between. This leads to problems when dealing with concurrency as formulated in challenge (4). Another approach that uses pattern recognition and machine learning techniques for abstraction is introduced in [27]. The authors aim at recognizing activities in streams of sensor data and are able to address challenge (2) and potentially challenge (3) with 60-80 % accuracy. Unfortunately, this approach does not allow to refine mappings to obtain full accuracy and also does not tackle challenge (4).

A different means of abstracting event logs is to simply remove insignificant behavior. This requires that a relation between events and activities has already been established. Together with the fuzzy miner, an approach is defined to abstract a mined process model by removing and clustering less

frequent behavior [5]. While the clusters are not directly related to activities, they can be interpreted as such. Nevertheless, the approach only allows for 1:n relations between event classes and clusters. The techniques used in event correlation by definition only aim at 1:1 relations, as the main objective of event correlation is to assign each event instance to one process instance.

In the works on process model abstraction we also find only 1:n relations between the matched objects. Here, these objects are activities only. Model similarity tries to make 1:1 relations between the sets of activities of two different process models. In the area of process model matching there are several works that establish n:m relations [28, 29, 30]. The work on ICoP defines a generic framework for process model matching [29]. This framework is extended with semantic concepts and probabilistic optimization in [24], adapting general concepts from ontology matching [31]. The implications of different abstraction levels for finding correspondences is covered in [25, 30, 32]. However, all these works focus on finding matches between two process models, not between events and activities.

The approach reported in [33] clusters process instances with similar behavior in order to abstract behavior that is different between these clusters. While this is an interesting approach for exploratory analysis, it is not able to abstract events that always occur together.

Furthermore, there are different approaches that apply behaviour abstraction in process discovery and trace alignment [34, 35, 36]. The technique proposed in this paper provides preprocessing for these approaches.

6. Discussion

In this section we emphasize implications for research and for practice and shed light on limitations of our work.

The research presented in this paper has several implications for research and practice. The aggregation of events in process mining has attracted some research [8, 6, 5]; however, this problem has been hardly approached from a perspective of descriptive semantics. It is the contribution of this paper to leverage insights from matching for specific problems to the processing of event logs. Our work also adds an important perspective to the discussion of how quantitative results from conformance checking have to be interpreted. It has been emphasized in prior research that different definitions of conformance measurement yields significantly different values for the same process

and log [37]. The fact that conformance measurement is also strongly influenced by the way how events are mapped and clustered to activity instances has received less attention so far. Our evaluation shows that different strategies of clustering events can yield strikingly different results (85% versus 77% for the change process). This finding emphasizes the importance of providing accurate techniques for matching events and activities.

Moreover, a correct abstraction will also benefit performance analysis on activity level, as the correct clustering of event instances is important to identify start and end of an activity to calculate the duration. Last but not least, process discovery also profits from a correctly abstracted event log. First, mined process models can be better understand by domain experts if they are on the abstraction level that is typically used in a company or department. Second, the introduced abstraction approach also allows for more advanced process discovery with the possibility to zoom into the different abstraction levels.

There are also limitations of our work. The accurate mapping of events to activity instances requires manual work. Our approach provides systematic support for automating a considerable share of this task. However, there is still a significant share of manual work required in order to encode missing domain knowledge into the required mapping definitions. This involves the definition of context conditions, event-activity relations, and instance border definitions. Still, these provide more accurate conformance and performance results as compared to techniques that cluster events without taking external knowledge into account as we have shown in our evaluation. Concerning the amount of manual work there is still room for improvement in future works. While the evaluation shows that our approach works good in detecting the relations and gives a high recall for the found event-activity relations, the low precision still leaves the user with a certain amount of wrong relations that need to be sorted out. Here further research is needed to increase precision and to develop methods to make the sorting out more efficient, e.g. by guiding the user in some way.

Looking at the generalizability of our approach, we are confident that it can be used in any application scenario. Although our evaluation only looks at two specific cases, the approach itself is generic. Yet, the mappings are always domain specific, which means that you cannot simply transfer the mappings from one process to the next, but always need to create mappings that fit to the process and supporting application at hand.

7. Conclusion

In this paper we presented a novel approach to tackle the abstraction of event logs. Our approach distinguishes from current works by explicitly targeting a specific abstraction level and by allowing for n:m relations and concurrency. We therefore explicitly encode domain knowledge into the mapping function in order to get the same level of abstraction as used in the defined business activities. We do this in a semi-automated manner by automatically matching events and activities using existing process descriptions and by allowing for the specification of activity instance borders. Our approach can be used as preprocessing of event logs to lift the results of process mining techniques to a business level. We have successfully evaluated our approach and could thereby show the influence of incorrect abstractions on conformance and performance analysis results.

Future work should seek for possibilities to automatically propose context conditions and should investigate possibilities to guide the user in the steps that have to be performed manually, as e.g. the selection of correct event-activity relations.

Appendix A. Overview of related work

Author (Year)	Title	Approach	Category	Model input	Event log input	External knowledge	Class level relation
Cook et al. (2013)	Activity Discovery and Activity Recognition: A New Partnership	Mapping sensor data to activities using machine learning techniques	pattern discovery	no	yes	no	n:m
Günther et al. (2006) [8]	Mining Activity Clusters From Low-level Event Logs	Clustering of correlated event instances based on event proximity on trace level	pattern discovery	no	yes	no	1:n
Günther et al. (2007) [5]	Fuzzy mining: adaptive process simplification based on multi-perspective metrics	Clustering of correlated event classes / omitting infrequent event classes	pattern discovery	no	yes	no	1:n
Günther et al. (2009) [6]	Activity mining by global trace segmentation	Hierarchical clustering of correlated event classes	pattern discovery	no	yes	no	1:n
Li et al. (2011) [7]	Mining context-dependent and interactive business process maps using execution patterns	Hierarchical clustering of correlated event classes; omitting of insignificant event classes	pattern discovery	no	yes	no	n:m
Beheshti et al. (2011) [38]	A Query Language for Analyzing Business Process Execution	Related events are grouped into folders (process instances) and paths (models) by correlation conditions expressed in an extension of SPARQL	event correlation	no	yes	no	1:1
Bose et al. (2013) [39]	Enhancing Declare Maps Based on Event Correlations	Pruning relations between events not sharing the same data objects and disambiguation of event relations using conditions on their attributes	event correlation	yes	yes	no	1:1
Motahari-Nezhad et al. (2010) [40]	Event correlation for process discovery from web service interaction logs	Correlating event instances to process instances using attribute values	event correlation	no	yes	no	1:1
Pérez-Castillo (2012) [9]	Assessing event correlation in non-process-aware information systems	Discovering correlation sets over attributes from events generated by inserting source code statements	event correlation	no	yes	source code	1:1
Rozsnyai (2011) [10]	Discovering Event Correlation Rules for Semi-Structured Business Processes	Discovery of event correlation rules based on statistics of event attributes	event correlation	no	yes	no	1:1
Steinle et al. (2006) [41]	Mapping Moving Landscapes by Mining Mountains of Logs : Novel Techniques for Dependency Model Generation	Correlating event classes (here systems) by time proximity and user sessions	event correlation	no	yes	no	n:m
Pérez-Castillo (2011) [23]	Generating event logs from non-process-aware systems enabling business process mining	Generating events that reflect business activities by inserting source code statements	event generation	no	no	source code	1:n
Greco et al. (2008) [33]	Mining taxonomies of process models	Abstraction of activities from hierarchical clustered process models mined using trace clustering	model abstraction	yes	yes	no	1:n

Polyvyanyy et al. (2008) [34]	Process Model Abstraction: A Slider Approach.	Aggregation and elimination of insignificant model elements	model abstraction	yes	no	probabilities	1:n
Polyvyanyy et al. (2009) [42]	On Application of Structural Decomposition for Process Model Abstraction	Hierarchical structural decomposition of process models	model abstraction	yes	no	no	1:n
Smirnov et al. (2012) [26]	From fine-grained to abstract process models : A semantic approach	Activity aggregation by clustering with a distance measure over the properties of activities	model abstraction	yes	no	roles, data	1:n
Smirnov et al. (2013) [43]	Business Process Model Abstraction Based on Synthesis from Well-Structured Behavioral Profiles.	Leveraging behavioral profiles to derive control-flow structure for abstracted models	model abstraction	yes	no	no	1:n
Branco et al. (2010) [28]	Matching business process workflows across abstraction levels	Matching of model fragments based on attributes and control flow structure using process structure trees.	model matching	yes	no	no	m:m
Klinkemüller et al. (2013) [25]	Increasing Recall of Process Model Matching by Improved Activity Label Matching	Matching based on bag-of-words and label pruning	model matching	yes	no	semantic relations (WordNet, Lin metric)	1:1
Leopold et al. (2012) [24]	Probabilistic Optimization of Semantic Process Model Matching	Matching based on annotation, semantic relations and behavioural constraints	model matching	yes	no	semantic relations (WordNet, Lin metric)	1:1
Weidlich et al. (2010) [29]	The ICoP Framework : Identification of Correspondences between Process Models	Matching based on syntactic label similarity	model matching	yes	no	no	n:m
Weidlich et al. (2012) [30]	Behaviour Equivalence and Compatibility of Business Process Models with Complex Correspondences	Determining behavioral equivalence between sets of activities	model matching & similarity	yes	no	no	n:m
Dijkman et al. (2011) [44]	Similarity of business process models: Metrics and evaluation	Matching of models using semantic label matching, structural and behavioral matching	model similarity	yes	no	synonym relations	1:1
Dongen et al. (2008) [45]	Measuring Similarity between Business Process Models	Matching of models based on labels and input/output context	model similarity	yes	no	synonym relations	1:1
Kunze et al. (2011) [46]	Behavioral similarity - a proper metric	Similarity measure of two models based on the Jaccard coefficient using behavioral profiles	model similarity	yes	no	no	1:1
Polyvyanyy et al. (2012) [47]	Isotactics as a Foundation for Alignment and Abstraction of Behavioral Models	Equivalence of aligned models with complex correspondences using concurrency semantics	model similarity	yes	no	no	n:m

Table A.4: Overview of related work

References

[1] W. M. P. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, 1st Edition, Springer, 2011.

[2] T. Baier, J. Mendling, Bridging abstraction layers in process mining: Event to activity mapping, in: BPMDS'13, Vol. 147 of LNBIP, Springer, 2013, pp. 109–123.

[3] T. Baier, J. Mendling, Bridging abstraction layers in process mining by automated matching of events and activities, in: BPM, 2013, pp. 17–32.

[4] D. Cannon, D. Wheeldon, ITIL – Service Operation, TSO, 2007.

[5] C. W. Günther, W. M. P. van der Aalst, Fuzzy mining: adaptive process simplification based on multi-perspective metrics, in: BPM'2007, Springer, 2007, pp. 328–343.

[6] C. W. Günther, A. Rozinat, W. M. P. van der Aalst, Activity mining by global trace segmentation, in: BPM Workshops, 2009, pp. 128–139.

[7] J. Li, R. Bose, 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, Springer, 2011, pp. 109–121.

[8] C. W. Günther, W. M. P. van der Aalst, Mining activity clusters from low-level event logs, in: BETA Working Paper Series, Vol. WP 165, Eindhoven University of Technology, 2006.

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

[10] S. Rozsnyai, A. Slominski, G. T. Lakshmanan, Discovering event correlation rules for semi-structured business processes, in: DEBS, 2011, pp. 75–86.

[11] D. Jurafsky, J. Martin, Speech and language processing, Prentice Hall, 2008.

[12] K. Toutanova, C. D. Manning, Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger, EMNLP (2000) 63–70.

[13] K. Kettunen, M. Sadeniemi, T. Lindh-Knuutila, T. Honkela, Analysis of eu languages through text compression., in: FinTAL, Vol. 4139 of LNCS, Springer, 2006, pp. 99–109.

[14] M. Braschler, B. Ripplinger, How Effective is Stemming and Decompounding for German Text Retrieval?, IR 7 (3/4) (2004) 291–316.

[15] S. Abels, A. Hahn, Pre-processing Text for Web Information Retrieval Purposes by Splitting Compounds into their Morphemes, in: OSWIR, 2005.

[16] A. Pnueli, The Temporal Logic of Programs, in: Foundations of Computer Science, 1977, pp. 46–57.

[17] W. M. P. van der Aalst, H. T. de Beer, B. F. van Dongen, Process mining and verification of properties: An approach based on temporal logic, Vol. 3760 of LNCS, Springer, 2005, pp. 130–147.

[18] I. H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, 2nd Edition, Morgan Kaufmann, 2005.

[19] R. P. J. C. Bose, H. M. W. E. Verbeek, W. M. P. van der Aalst, Discovering hierarchical process models using prom., Vol. 107 of LNBIP, Springer, 2011, pp. 33–48.

[20] C. W. Günther, XES Standard Definition, 1.0, Draft. available at: http://www.xes-standard.org/ (November 2009).

[21] R. A. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, ACM Press / Addison-Wesley, 1999.

[22] M. Weidlich, A. Polyvyanyy, N. Desai, J. Mendling, M. Weske, Process compliance analysis based on behavioural profiles, Information Systems 36 (7) (2011) 1009 – 1025.

[23] R. Pérez-Castillo, B. Weber, J. Pinggera, S. Zugal, I. G. R. de Guzmán, M. Piattini, Generating event logs from non-process-aware systems enabling business process mining, Enterprise IS 5 (3) (2011) 301–335.

[24] H. Leopold, M. Niepert, M. Weidlich, J. Mendling, R. Dijkman, H. Stuckenschmidt, Probabilistic optimization of semantic process model matching, in: BPM'2012, 2012, pp. 319–334.

[25] C. Klinkmüller, I. Weber, J. Mendling, H. Leopold, A. Ludwig, Increasing recall of process model matching by improved activity label matching, in: BPM, 2013, pp. 211–218.

[26] S. Smirnov, H. A. Reijers, M. Weske, From fine-grained to abstract process models: A semantic approach, Inf. Syst. 37 (8) (2012) 784–797.

[27] D. J. Cook, N. C. Krishnan, P. Rashidi, Activity discovery and activity recognition: A new partnership, IEEE T. Cybernetics 43 (3) (2013) 820–828.

[28] M. C. Branco, J. Troya, K. Czarnecki, J. M. Küster, H. Völzer, Matching business process workflows across abstraction levels, in: MoDELS, 2012, pp. 626–641.

[29] M. Weidlich, R. M. Dijkman, J. Mendling, The ICoP Framework: Identification of Correspondences between Process Models, in: CAiSE 2010, Vol. 6051 of LNCS, Springer, 2010, pp. 483–498.

[30] M. Weidlich, R. Dijkman, M. Weske, Behaviour Equivalence and Compatibility of Business Process Models with Complex Correspondences, ComJnl.

[31] J. Euzenat, P. Shvaiko, Ontology Matching, Springer-Verlag, 2007.

[32] M. Weidlich, T. Sagi, H. Leopold, A. Gal, J. Mendling, Making process model matching work, in: Business Process Management - 11th International Conference, BPM 2013, Proceedings, LNCS, Springer, 2013.

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

[34] A. Polyvyanyy, S. Smirnov, M. Weske, Process Model Abstraction: A Slider Approach, in: EDOC, IEEE, 2008, pp. 325–331.

[35] D. Fahland, W. M. P. van der Aalst, Simplifying discovered process models in a controlled manner, Inf. Syst. 38 (4) (2013) 585–605.

[36] R. P. J. C. Bose, W. M. P. van der Aalst, Process diagnostics using trace alignment: Opportunities, issues, and challenges, Inf. Syst. 37 (2) (2012) 117–141.

[37] K. Gerke, J. Cardoso, A. Claus, Measuring the compliance of processes with reference models, Vol. 5870 of LNCS, Springer, 2009, pp. 76–93.

[38] S.-M.-R. Beheshti, B. Benatallah, H. R. M. Nezhad, S. Sakr, A query language for analyzing business processes execution, in: BPM, 2011, pp. 281–297.

[39] R. P. J. C. Bose, F. M. Maggi, W. M. P. van der Aalst, Enhancing declare maps based on event correlations, in: BPM, 2013, pp. 97–112.

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

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

[42] A. Polyvyanyy, S. Smirnov, M. Weske, On application of structural decomposition for process model abstraction, in: BPSC, 2009, pp. 110–122.

[43] S. Smirnov, M. Weidlich, J. Mendling, Business process model abstraction based on synthesis from well-structured behavioral profiles, Int. J. Cooperative Inf. Syst. 21 (1) (2012) 55–83.

[44] R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, J. Mendling, Similarity of Business Process Models: Metrics and Evaluation, Information Systems 36 (2) (2011) 498–516.

[45] B. F. van Dongen, R. M. Dijkman, J. Mendling, Measuring similarity between business process models., in: J. A. B. Jr., J. Krogstie, O. Pastor, B. Pernici, C. Rolland, A. Sølvberg (Eds.), Seminal Contributions to Information Systems Engineering, Springer, 2013, pp. 405–419.

[46] M. Kunze, M. Weidlich, M. Weske, Behavioral similarity - a proper metric, in: BPM, 2011, pp. 166–181.

[47] A. Polyvyanyy, M. Weidlich, M. Weske, Isotactics as a foundation for alignment and abstraction of behavioral models, in: BPM, 2012, pp. 335–351.