From Low-Level Events to Activities - A Pattern-Based Approach

Felix Mannhardt[1(✉)], Massimiliano de Leoni[1], Hajo A. Reijers[1,2],
Wil M.P. van der Aalst[1], and Pieter J. Toussaint[3]

[1] Eindhoven University of Technology, Eindhoven, The Netherlands
{f.mannhardt,m.d.leoni,h.a.reijers,w.m.p.v.d.aalst}@tue.nl
[2] Vrije Universiteit Amsterdam, Amsterdam, The Netherlands
[3] Norwegian University of Science and Technology, Trondheim, Norway
pieter@idi.ntnu.no

Abstract. Process mining techniques analyze processes based on event data. A crucial assumption for process analysis is that events correspond to occurrences of meaningful activities. Often, low-level events recorded by information systems do not directly correspond to these. Abstraction methods, which provide a mapping from the recorded events to activities recognizable by process workers, are needed. Existing supervised abstraction methods require a full model of the entire process as input and cannot handle noise. This paper proposes a supervised abstraction method based on behavioral activity patterns that capture domain knowledge on the relation between activities and events. Through an alignment between the activity patterns and the low-level event logs an abstracted event log is obtained. Events in the abstracted event log correspond to instantiations of recognizable activities. The method is evaluated with domain experts of a Norwegian hospital using an event log from their digital whiteboard system. The evaluation shows that state-of-the art process mining methods provide valuable insights on the usage of the system when using the abstracted event log, but fail when using the original lower level event log.

Keywords: Process mining · Supervised abstraction · Event log · Alignment

1 Introduction

Organizations use information systems to support their work. Often, information about the usage of those systems by workers is recorded in event logs [1]. Process mining techniques use such event data to analyze processes of organizations. It is assumed that recorded events correspond to meaningful activities in instances of a process (i.e., cases). This information about recorded executions of activities can then be used, e.g., to *discover* models describing the observed behavior or to check *conformance* with existing process documentation. The ability to identify executions of activities based on events is crucial for any process mining technique. Events that do not directly correspond to activities recognizable for process workers are unsuitable for process analytics since their semantics are not clear to domain experts. However, events recorded

by information systems often do not directly correspond to recognizable executions of activities [2]. Generally, there can be an n:m-relation between recorded events and activities [2,3], i.e., one higher level activity may create multiple low level events and one such event possibly relates to multiple activities. There are proposals for unsupervised abstraction methods that try to determine this relation based on identifying subsequences and machine learning methods [2,4–7], as well as proposals for supervised methods based on existing process documentation and constraint satisfaction [3,8–11]. Unsupervised abstraction methods, clearly, do not take existing knowledge into account and may fail to provide meaningful labels for discovered event clusters. Existing supervised abstraction methods [3,8–11] assume knowledge about a single model for the overall process. They resolve to clustering methods and heuristics when challenged with event logs from processes that feature *n:m event-activity relations*, *concurrent* activities, and *noise* (i.e., erroneous or missing events).

This paper proposes a supervised event abstraction method. We use behavioral *activity patterns* to capture domain knowledge about the conjectured relation between high-level activities and recorded low-level events. We *align* the behavior defined by these activity patterns with the observed behavior in the event log. Our technique uses alignment techniques, and, hence, is able to find an optimal mapping between low-level events and activity patterns even for event logs that contain noise. In this way, we obtain a reliable *abstraction mapping* from low-level events to activity patterns. This mapping is used to create an abstracted event log. This log contains only high-level events at the desired level of abstraction, which relate directly to executions of high-level activities. We applied the proposed method together with domain experts from a Norwegian hospital to an event log retrieved from a digital whiteboard system at the observation ward of the hospital. Through observation and interviews with people working at the hospital we were able to identify activity patterns for 18 recognizable activities, which, together, explained 91 % of the recorded behavior. Using the abstracted event log, we were able to analyze how nurses use the digital whiteboard system in their daily work. We obtained process models that relate to their actual work in a meaningful way.

The remainder of this paper is structured as follows. First, we describe the event abstraction problem in more detail (Sect. 2). Then, we present the five main steps of our abstraction method (Sect. 3). We evaluate the proposed method using the results obtained for the digital whiteboard event log (Sect. 4), and conclude with a summary and a sketch of future work (Sect. 5).

2   Problem Description

We start with a definition of event logs. An event log stores information about activities that were recorded by one or more information systems while supporting the execution of a process. Each execution of a *process instance* results in a sequence of events.

Definition 1 (Event Log). *Given universes of attributes A and values U, we define an event log as $(E, \Sigma, \#, \mathscr{E})$ with:*

– *E is a set of unique event identifiers;*
– *$\Sigma \subseteq U$ is a set of activities;*

Table 1. Excerpt of a trace $\sigma_L \in \mathscr{E}_L$ from a low-level event log with identifiers **Id** and attributes **Activity**, **Time**, **Instance**, and **Nurse**. Symbol $\perp$ denotes that the attribute was not recorded. The last columns show those high-level activities, which caused the event.

Id	Activity	Time	Instance	Nurse	High-level activity	High-level instance
e_{12}	NurseChanged	122	12	NurseA	Shift	1
e_{13}	CallSignal1	122	13	$\perp$	Shift	1
e_{14}	CallSignal0	124	14	$\perp$	Shift	1
...	...	...	...	...	...	...
e_{20}	CallSignal4	185	20	$\perp$	Alarm	2
e_{21}	CallSignal1	197	21	$\perp$	Alarm	2
...	...	...	...	...	...	...
e_{29}	NurseChanged	250	29	NurseB	Handover	3
e_{30}	CallSignal4	310	30	$\perp$	Alarm	4
e_{31}	CallSignal1	311	31	$\perp$	Alarm	4
e_{32}	NurseChanged	312	32	NurseC	Handover	5
e_{33}	CallSignal0	315	33	$\perp$	Alarm	4

- $\# : E \rightarrow (A \nrightarrow U)$ *is a function that obtains attribute values recorded for an event;*
- $\mathscr{E} \subseteq E^*$ *is the set of traces over* E. *A trace* $\sigma \in \mathscr{E}$ *records the sequence of events for one process instance. Each event identifier occurs only in a single trace.*

Given an event $e \in E$ in the event log $\mathscr{E}$, we write $\#_a(e) \in A \nrightarrow U$ to obtain the value $u \in U$ recorded for attribute $a \in A$. Three mandatory attributes are recorded by each event: $\#_{act}(e) \in \Sigma$, the **name of the activity** that caused the event; $\#_{time}(e) \in U$, the **time** when the event occurred; $\#_{ai}(e) \in U$, the **activity instance**, i.e., an identifier linking multiple events, which are related to the same execution of a single activity.

Example 1. Table 1 shows an excerpt of a trace $\sigma_L \in \mathscr{E}_L$ obtained from a low-level event log $(E_L, \Sigma_L, \#^L, \mathscr{E}_L)$ that is recorded by a digital whiteboard, which supports the work of nurses in a hospital. Each row represents an unique event $e \in E_L$ together with the produced data (i.e., attributes) created by a change in the system. For confidentiality reasons, we show only some events of an artificial trace that resembles the real data. The initial events are omitted. After 122 min low-level activity NurseChanged (*NC*) occurs resulting in event e_{12}. Attribute Nurse is recorded as $\#^L_{\texttt{Nurse}}(e_{12}) = NurseA$. Next, two low-level activities CallSignal1 (*CS1*) and CallSignal0 (*CS0*) are registered as events e_{13} and e_{14} by a call signal system, which is integrated with the whiteboard. An hour later the call signal system records the activity CallSignal4 (*CS4*) as event e_{20} and, again, activity *CS1* as event e_{21}. Some further low-level events follow.

Often, not all events $e \in E_L$ represent work at the same level of abstraction [1,4]. The execution of some high-level activities might result in multiple low-level events being recorded during their execution. Those events only store the names of low-level activities Σ_L, i.e., $\#^L_{act}(e) \in \Sigma_L$ instead of names of recognizable high-level activities. *Event abstraction* can be seen as the problem of transforming such an event
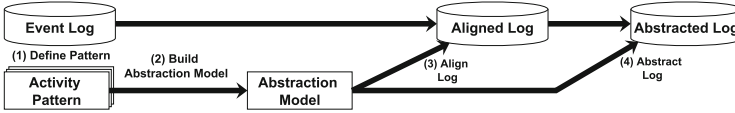
Fig. 1. Overview of the proposed event abstraction method

log $(E_L, \Sigma_L, \#^L, \mathscr{E}_L)$ at a lower or mixed level of abstraction, into a new event log $(E_H, \Sigma_H, \#^H, \mathscr{E}_H)$ with events E_H that record executions of activities Σ_H at the desired, higher level of abstraction. We need to determine how low-level events E_L are related to **high-level events** E_H, i.e., we need to find an **abstraction mapping** $\pi \subseteq E_L \times E_H$.

Please note that determining a good abstraction mapping π (i.e., one that reflects what really happened) is difficult for several reasons. Low-level events mapped to more than one high-level activity, i.e., *shared functionality* [3] need to be disambiguated. It is difficult to differentiate between *reoccurring* and *concurrent* activities [3]. Also, the low-level event log might contain *noise*: Erroneous events that should not have been recorded or missing events that should have been recorded.

Example 2. Event log $\mathscr{E}_L$ shown in Table 1 contains low-level events. The various `CallSignal` events do not directly correspond to high-level activities. Moreover, depending on the context, those events correspond to different high-level activities. The last two columns in Table 1 list the corresponding names of high-level activities that caused the low-level events as well as an identifier uniquely identifying the execution of the activity, i.e., the activity instance. For example, we know that in the context of a shift change events *CS1* and *CS0* are recorded when the patient is visited in the normal routine, i.e., events e_{12}, e_{13}, e_{14} correspond to one execution (i.e., instance 1) of the high-level activity `Shift`. This mapping between low-level events and high-level activity instances cannot be solely done on the activity names. For example, when *CS1* and *CS0* are preceded by event *CS4* they correspond to an alarm triggered by the patient, i.e., events e_{30}, e_{31}, e_{33} were caused by instance 4 of high-level activity `Alarm`.

3   Pattern-Based Abstraction of Event Logs

We present a method that takes an event log $(E_L, \Sigma_L, \#^L, \mathscr{E}_L)$ at a lower level of abstraction and transforms it to an event log $(E_H, \Sigma_H, \#^H, \mathscr{E}_H)$ at the desired level of abstraction. We establish an abstraction mapping π from events E_L to the events E_H. Our method can deal with noise, reoccurring and concurrent behavior, and shared functionality. The proposed method consists of four steps (Fig. 1):

1. We encode the low-level behavior of activities in *activity patterns* (Sect. 3.1).
2. We compose activity patterns in an *abstraction model* (Sect. 3.2).
3. We align the behavior of the *abstraction model* and the low-level event log (Sect. 3.3).
4. We create an *abstracted* event log using the alignment information (Sect. 3.4).

We describe these steps in the following sections. Note that activity patterns represent domain knowledge on the behavior of high level activities in terms of low-level events.
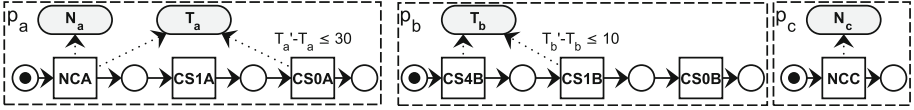
Fig. 2. Three activity patterns $p_a, p_b, p_c \in P$ for the example in DPN notation

3.1 Define Activity Patterns

In the reminder of this paper, we use process models to capture behavior. Generally, our abstraction method is independent of the particular formalism (e.g., Petri nets, UML, Declare, BPMN) used to model processes. We represent knowledge about the relation between low-level events and given high-level activities Σ_H in *activity patterns*. Each activity pattern is a process model describing those events that are expected to be seen in the event log for one instance of the corresponding high-level activity.

Definition 2 (Activity Pattern). *Given a set of low-level activity names Σ_L, process moves Σ_M, process attributes A and values U. Let $S = (\Sigma_M \times (A \nrightarrow U))$ be the set of all possible process steps. Let $\#_{name} : \Sigma_M \to \Sigma_L$ be a labeling function that returns the low-level activity name of a process move. An activity pattern $p \subseteq S^*$ captures sequences corresponding to an execution of one instance of a high-level activity. Steps $s \in \sigma$ in process traces $\sigma \in p$ correspond to low-level activities executed as part of the high-level activity. We denote with $P = \{p \subseteq S^*\}$ the set of all activity patterns.*

In the remainder, we require that process moves are not shared between activity patterns, i.e., given two different patterns $p_1, p_2 \in P$ and sequences $\sigma_1 \in p_1, \sigma_2 \in p_2$ we require for any steps $(m_1, w_1) \in \sigma_1, (m_2, w_2) \in \sigma_2$ that $m_1 \neq m_2$. Given a step, we can uniquely identify to which pattern it belongs. However, process moves from different patterns may be associated with the same activity name, i.e., $\#_{name}(m_1) = \#_{name}(m_2)$.

Example 3. Fig. 2 shows three activity patterns p_a, p_b and p_c defined for the event log in Table 1, implemented as Data Petri Nets (DPNs) [12]. We implement activity patterns by using DPN as notation with well-defined semantics, which can express the control-flow as well as the data-, resource- and time-perspective of a pattern. We refer to [12] for an introduction to DPNs. We use transitions of the DPN to model process moves. We name transitions uniquely by using the abbreviated low-level activity name concatenated with the pattern name, e.g., transition *CS1A* models activity CallSignal1 in pattern p_a. Therefore, we can easily obtain the activity name (i.e., $\#_{name}(x)$) for each transition x. The *first pattern p_a* describes a shift change. First, the nurse responsible for the patient changes (*NCA*) and the name of the nurse is recorded (N_a). Within 30 min ($T_a' - T_a \leq 30$), the responsible nurse visits the patient and the call signal system records a button press (*CS1A*). Finally, the nurse leaves the room and another button press is registered (*CS0A*) resetting the status. The *second pattern p_b* describes a similar sequence (i.e., transitions *CS1B* and *CS0B*), but represents a different high-level activity: The patient is attended outside of the normal routine. Transition *CS4B* has to be executed at most 10 min beforehand (i.e., $T_b' - T_b \leq 10$). The low-level activity corresponding to *CS4B* is an *alarm* triggered by the patient. The *third pattern* describes a

simple handover between nurses: Only the responsible nurse changes (*NCC*) without any consultation of the patient. The corresponding low-level activity `NurseChanged` is shared with a transition *NCA* of pattern p_a. This is an example of shared functionality.

Using domain knowledge about the high-level activities of the process at hand we define such an *activity pattern* for every activity of interest. *Activity patterns* represent the knowledge about how high-level activities are reflected by low-level events in the event log. Please note that we do not expect an *activity pattern* to be an exact representations of every possible way a high-level activity manifests itself in the event log. Later, in Sect. 3.3 we show that our method is able to deal with approximate matches.

3.2 Build an Composed Abstraction Model

With a set of activity patterns for the process under analysis at hand, we can compose their behavior into an integrated *abstraction model*.

Definition 3 (Composition Function). *A composition function $f : 2^P \rightarrow P$ combines the behavior activity patterns $p_1, \ldots, p_n$ into an (composite) activity pattern $cp \in P$, i.e., $f(p_1, \ldots, p_n) = cp$. We denote with $F \subseteq 2^P \rightarrow P$ the set of all composition functions.*

We provide the semantics for five basic composition functions: *sequence*, *choice*, *parallel*, *interleaving* and *cardinality*. Our abstraction method is not restricted to these functions. Further composition functions can be added. We introduce some necessary notations for sequences. Given a sequence $\sigma \in S^*$ and a subset $X \subseteq S$, $\sigma|_X$ is the *projection* of σ on X. For example, $\langle w,o,r,d \rangle|_{\{o,r\}} = \langle o,r \rangle$. $\sigma_1 \cdot \sigma_2 \in S^*$ *concatenates* two sequences, e.g., $\langle w,o \rangle \cdot \langle r,d \rangle = \langle w,o,r,d \rangle$. Given activity patterns $p_i \in P$ with $p_i \subseteq S_i^*$ and $i \in \mathbb{N}$, we define the following functions:

– **Sequence** composition $\odot \in F$:

$$p_1 \odot p_2 = \{\sigma \in S^* \mid \sigma_1 \in p_1 \wedge \sigma_2 \in p_2 \wedge \sigma = \sigma_1 \cdot \sigma_2\}.$$

Binary operation $\odot$ is associative. We write $\bigodot_{1 \leq i \leq n} p_i = p_1 \odot p_2 \odot \ldots \odot p_n$ to compose ordered collections of patterns in sequence. We define $\bigodot_{1 \leq i \leq 0} p_i = \{\langle \rangle\}$.

– **Choice** composition $\otimes \in F$:

$$p_1 \otimes p_2 = p_1 \cup p_2.$$

Binary operation $\otimes$ is commutative and associative. We write $\bigotimes_{1 \leq i \leq n} p_i = p_1 \otimes p_2 \otimes \ldots \otimes p_n$ to compose sets of patterns in choice.

– **Parallel** composition $\diamond \in F$:

$$p_1 \diamond p_2 = \{\sigma \in (S_1 \cup S_2)^* : \sigma|_{S_1} \in p_1 \wedge \sigma|_{S_2} \in p_2\}.$$

Binary operation $\diamond$ is commutative and associative. We write $\diamond_{1 \leq i \leq n} p_i = p_1 \diamond p_2 \diamond \ldots \diamond p_n$ to compose sets of patterns in parallel.
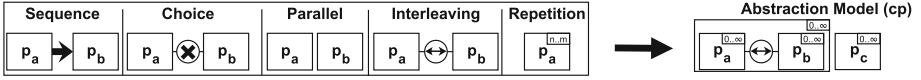
Fig. 3. Overview of the graphical notation of the supported composition functions and an example of their usage in an composed pattern. Patterns are depicted as plain boxes for better legibility.

- **Interleaving** composition $\leftrightarrow \in F$ with $p(n)$ denoting the set of all permutations of numbers $\{1, \ldots, n\}$:

$$\leftrightarrow (p_1, \ldots, p_n) = \bigotimes_{(i_1, \ldots, i_n) \in p(n)} \bigodot_{1 \leq k \leq n} p_{i_k}.$$

- **Repetition** composition $[n, m] \in F$ with $n \in \mathbb{N}_0, m \in \mathbb{N} \cup \{\infty\}$, and $n \leq m$:

$$p_1^{[n,m]} = \bigotimes_{n \leq i \leq m} \bigodot_{1 \leq k \leq i} p_1.$$

We build an overall abstraction model with a formula that contains all patterns of interest. The resulting composed pattern $cp \in S^*$ should include the overall behavior that we expect to observe for the execution of *all* high-level activities.

Example 4. Given the activity patterns p_a, p_b and p_c shown in Fig. 2, we can compose their behavior to $cp = (\leftrightarrow (p_a^{[0,\infty]}, p_b^{[0,\infty]}))^{[0,\infty]} \diamond p_c^{[0,\infty]}$. We allow indefinite repetition of all activity patterns using the repetition composition. We allow the absence of patterns using the repetition composition as the corresponding high-level activities might not have been executed in every process instance. We restrict cp to only contain the interleaving of patterns p_a and p_b as there is only one responsible nurse per patient. Therefore, the activities expressed by p_a and p_b can occur in any order but should not happen in parallel. We add p_c using the parallel composition as handovers can take place in parallel to p_a and p_b. In the remainder of this example, we omit the attribute assignments w from steps $(t, w) \in S$ for improved legibility. The result of this composition is the abstraction model cp. Model cp corresponds to all behavior that could be observed for executions of the three high-level activities. For example, $\langle NCA, CS1A, NCC, CS0A \rangle \in cp$ is expected, whereas $\langle NCA, CS1A, CS4B, CS0A \rangle \notin cp$ is not expected.

We designed a graphical representation for each composition function, which can be used to design abstraction models in the implementation of our approach. Figure 3 shows the graphical notation for each of the composition functions. Moreover, the graphical representation of the composition of activity patterns p_a, p_b, p_c as defined in Example 4 is shown. Because the repetition composition is unary, we attach its graphical representation directly to patterns. Parallel composition is the least restrictive composition. Unless otherwise specified, we assume that patterns are composed in parallel. We draw a box around composed patterns if necessary to clarify the precedence of operations. For example, patterns p_a and p_b are first interleaved and then composed in parallel with p_c. We implemented the composition of activity patterns using
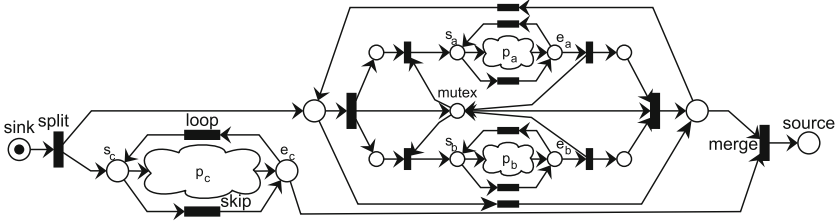
Fig. 4. DPN created by our implementation for the abstraction model cp. Activity patterns p_a, p_b, p_c are depicted as clouds with source places s_a, s_b, s_c and sink places e_a, e_b, e_c. Black transitions are invisible routing transitions, which are not recorded in any event log.

Table 2. The top three rows show an alignment of the running example log trace and abstraction model. Low-level events (L. Event) e are aligned to process moves (P. Move) m that relate to the same low-level activity (L. Activity). Write operations are omitted for better legibility. One process move could not be aligned to an event, symbol $\gg$ is used in this case. The bottom five rows show the high-level event returned by the abstraction method described in this paper.

L. Event (e)	e_{12}	e_{13}	e_{14}	...	e_{20}	e_{21}	$\gg$	...	e_{29}	e_{30}	e_{31}	e_{32}	e_{33}
L. Activity	NC	CS1	CS0	...	CS4	CS1		...	NC	CS4	CS1	NC	CS0
P. Move (m)	NCA	CS1A	CS0A	...	CS4B	CS1B	CS0B	...	NCC	CS4B	CS1B	NCC	CS0B
H. Activity	Shift		Shift	...	Alarm		Alarm	...	Handover	Alarm		Handover	Alarm
Life-cycle	Start		Complete	...	Start		Complete	...	Complete	Start		Complete	Complete
Instance	3		3	...	6		6	...	10	11		12	11
Time	122		124	...	185		197	...	250	310		312	315
H. Event	$\hat{e}_5$		$\hat{e}_6$	...	$\hat{e}_{11}$		$\hat{e}_{12}$	...	$\hat{e}_{20}$	$\hat{e}_{21}$		$\hat{e}_{22}$	$\hat{e}_{23}$

the DPN notation. To simplify the composition, we assume that the DPNs of activity patterns have a single source place and a single sink place. Figure 3 shows the DPN encoding of $cp = (\leftrightarrow (p_a^{[0,\infty]}, p_b^{[0,\infty]}))^{[0,\infty]} \diamond p_c^{[0,\infty]}$. The implementation of all compositions using DPN is available in a technical report [13].

Example 5. Figure 4 depicts the DPN implementation of abstraction model cp. The abstraction model starts with a single sink place `sink` and ends with a single source place `source`. We model the parallel composition of $p_c^{[0,\infty]}$ with $\leftrightarrow (p_a^{[0,\infty]}, p_b^{[0,\infty]})^{[0,\infty]}$ by adding invisible transitions `split` and `merge`, which realize a parallel split and join. Invisible transitions cannot be observed; they are only added for routing purposes. We use place `mutex` to model the mutual exclusion constraint of the interleaving composition of patterns $p_a^{[0,\infty]}$ and $p_b^{[0,\infty]}$. Place `mutex` guarantees that only either p_a or p_b can be executed at the same time, yielding the interleaving of p_a and p_b. Each repetition composition is implemented by adding two invisible transitions `loop` and `skip`, which allow to repeat the pattern indefinitely or to skip its execution, respectively.

3.3   Alignment of Patterns Behavior and the Event Log

With an abstraction model at hand, we need to relate the behavior in the low-level event log to process traces defined by the abstraction model. More specifically, we need to

determine the mapping between low-level events in the event log and process steps of the abstraction model. We use existing **alignment** techniques [12] that establish a mapping between log traces and process traces. The top three rows of Table 2 show such an alignment between the example log trace (Table 1) and a process trace of the example abstraction model cp (Fig. 3). The alignment in Table 2 consists of *moves* $(e, s) \in (E_L \cup \{\gg\}) \times (S \cup \{\gg\})$ that relate low-level events e to process steps s in the abstraction model. Events e can only be mapped to process steps $s = (m, w)$ that refer to the same low-level activity, i.e., $\#_{act}(e) = \#_{name}(m)$. It may not be possible to align all events and process steps. These deviating events and process steps are mapped to $\gg$ (e.g., $(\gg, (CS0B, w))$ in Table 2). Alignments find an optimal mapping, which minimizes the number of such deviations. They return the most likely mapping between events and process steps. Moreover, an alignment guarantees that its sequence of model steps without $\gg$-steps is a process trace defined by the model. For example, the third row in Table 2 is a process trace of abstraction model cp. Pattern p_a is executed once, i.e., $\langle NCA, CS1A, CS0A \rangle$ is a sub-sequence. Patterns p_b and p_c are both repeated twice, i.e., there are two sub-sequences $\langle CS4B, CS1B, CS0B \rangle$ and two sub-sequences $\langle NCC \rangle$. We can uniquely identify sub-sequences of initial activity pattern since we required that process moves are unique among activity patterns.

3.4   Build the Abstracted Event Log Using the Alignment

We describe how to build the high-level event log $(E_H, \Sigma_H, \#^H, \mathscr{E}_H)$ and the abstraction mapping π using an alignment of the low-level event log with the abstraction model.

The bottom four rows of Table 2 show how we obtain the high-level event log from the information provided by the alignment. We align each trace of the low-level event log with the abstraction model. Doing so, we obtain an alignment as shown in the first three rows for each trace in the low-level log. Given the alignment, we use two mappings to build the high-level log:

– $\mu : \Sigma_M \to \Sigma_H$, a mapping between process moves and high-level activities.
– $\lambda : \Sigma_M \nrightarrow L$, a mapping between process moves and life-cycle transitions.

Mapping function μ can be obtained from the initially defined activity patterns. Each activity patterns models exactly one high-level activity and each process move belongs to exactly one activity patterns, thus, the corresponding high-level activity can be uniquely determined for each process move. For example, we use $\mu(NCA) = Shift$ and $\mu(NCC) = Handover$. Mapping function λ defines which process moves correspond to transitions in the life-cycle of activities. Mapping λ is motivated by the observation that activities rarely happen instantaneously. Activities have *life-cycles* [1]. The set of life-cycle transitions L and mapping function λ is specified by the user. In the case-study we use $L = \{start, complete\}$ and define λ such that the first process move of an activity pattern is mapped to the *start* transition and the last process move is mapped to the *complete* transition. The other process moves are not mapped, i.e., they are not in the domain of λ. For example, we use $\lambda(NCA) = start$ and $\lambda(CS0A) = complete$.

We add new high-level events e_H to E_H (i.e., $E_H = E_H \cup \{e_H\}$) for those alignment moves (e, s) for which process steps $s = (m, w)$ are not mapped to $\gg$ (i.e., $s \neq \gg$) and

process move m is mapped to a life-cycle transition $\lambda(m)$ (i.e., $m \in dom(\lambda))$[1]. In this manner, we create a high-level trace in $\mathscr{E}_H$ for each low-level trace in $\mathscr{E}_L$. We obtain the high-level log $(E_H, \Sigma_H, \#^H, \mathscr{E}_H)$ and a mapping between low-level events e and the new high-level events e_H. We include (e, e_H) in the abstraction mapping π when event e is not mapped to $\gg$ (i.e., $e \neq \gg$). For example, events $\hat{e}_5$ and $\hat{e}_6$ in Table 2 are created based on the alignment of low-level events e_{12} and e_{14} to process moves NCA and $CS0A$, i.e., $(e_{12}, \hat{e}_5) \in \pi$ and $(e_{14}, \hat{e}_6) \in \pi$. We assign event $\hat{e}_5$ the activity name $Shift$ (i.e., $\#^H_{act}(\hat{e}_5) = Shift$) and the life-cycle transition $start$ (i.e., $\#^H_{cycle}(\hat{e}_5) = start$). Event $\hat{e}_6$ is assigned the same activity name $Shift$, but a different life-cycle transition: $complete$.

Then, the high-level events E_H are enriched with additional information: *activity instance* and *timestamp*. A unique instance identifier is added for each execution of an activity pattern. For example, event $\hat{e}_{21}$ and event $\hat{e}_{23}$ are both assigned instance identifier 11 (i.e., $\#^H_{ai}(\hat{e}_{21}) = \#^H_{ai}(\hat{e}_{23}) = 11$). Both are aligned to process steps in the same execution of activity pattern p_b (*Alarm*). Instance 11 of the activity *Alarm* was started by event $\hat{e}_{21}$ and completed by event $\hat{e}_{23}$. Regarding the timestamp, there are two cases depending on the alignment move (e, s): (1) The process step was aligned to a low-level event e and (2) the process step was mapped to $e = \gg$. In the first case, we use the timestamp of the aligned low-level event (e.g., $\#^H_{time}(\hat{e}_{11}) = \#^H_{time}(e_{20}) = 185$). In the second case, we cannot directly obtain a timestamp. For example, event $\hat{e}_{12}$ in Table 2 is missing a low-level event: $e = \gg$. There are multiple methods to determine the most likely timestamp for $\hat{e}_{12}$. For the case study (Sect. 4), we use timestamps of neighboring low-level events that are mapped to the same activity instance, e.g., we use the timestamp from event e_{21} for the high-level event $\hat{e}_{12}$ (i.e., $\#^H_{time}(e_{21}) = 197$).

In general, there might be scenarios where one event could be mapped to several activity instances. We simplified the discussion by assuming that events are only mapped to single activity instances. This is not a limitation, as described by Baier et al. [3]: Those events can be duplicated in a pre-processing step beforehand.

Finally, we define two quality measures for the abstraction mapping. First, we use **fitness** as a measure for how well the entire event log matches the behavior imposed by the abstraction model. In this context, a fitness measure such the one defined in [12] for alignments of DPNs can be seen as measure for the quality of the used abstraction model. A low fitness indicates that there are many events that cannot be correctly matched, thus, the abstraction model does not capture the whole process correctly. Second, we define a **matching error** $\varepsilon : \Sigma_H \to [0, 1]$ on the level of each recognized high-level activity. Some process steps in the alignment are not matched to an event in the log, i.e., the event is missing. For example, in Table 2 one execution of process activity $CS0B$ is mapped to $\gg$. Given a high-level activity $h \in \Sigma_H$ (e.g., *Alarm*) and the subset of process activities $M \subset \Sigma_M$ that are mapped to the activity pattern defined for the high-level activity (e.g., $CS4B$, $CS1B$ and $CS0B$). We determine the number of those alignment moves $(e, (m, w))$ with process activities $m \in M$, for which the event is missing, i.e., $e = \gg$. The matching error $\varepsilon(h)$ is the fraction of such erroneous alignment moves over the total number of alignment moves with process move m. For example, $\varepsilon(Alarm) = \frac{5}{6}$ for the alignment in Table 2. The matching error can be used to exclude unreliable matches, which exceed a certain ε-threshold.

---

[1] $dom(f)$ denotes the domain of a function f.

4   Evaluation

We evaluate the proposed abstraction method by conducting a case study using event data that was obtained from a digital whiteboard system[2]. The whiteboard supports the daily work of nurses in the observation unit of a Norwegian hospital. Our method is implemented as plug-in of the open-source process mining framework ProM[3].

4.1   Case and Dataset

Digital whiteboard systems are used to improve health care processes by raising situation awareness among nurses and to support coordination of care [14]. In our case, the whiteboard is used to manage information about admitted patients. The information is displayed in a tabular manner, where each row shows information about a single patient. The cells are used for various purposes, such as displaying logistical and medical information about the patient. A call signal system, which allows patients to trigger an alarm, is integrated with the whiteboard. Alarms are shown on the whiteboard. Generally, there are few constraints on how the whiteboard is actually used.

We obtained an event log with *8,487 cases* and *286,000 events* recorded by the whiteboard of the observation unit between 04/2014 and 12/2015. Each case records events for the visit of a single patient. On average, traces contain 34 events. Events are recorded for changes of single cells of the whiteboard. This very fine grained logging leads to a low-level event log. Events in the log do not directly represent recognizable activities. In total, there are 42 distinct low-level activity names in the log. Moreover, varying work practices among nurses lead to different events being recorded for the same high-level activity. The event log is unsuitable for any kind of process analytics as the semantics of results are not clear to process workers.
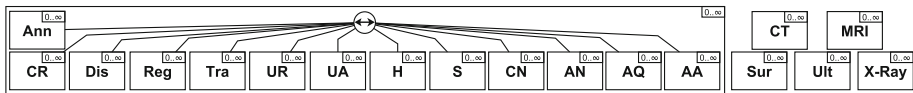

Fig. 5. Abstraction model used in the case study. Most activities can only interleave as there is only one nurse assigned to a patient.

We created an abstraction model with 18 activity patterns as shown in Fig. 5. The activity patterns are based on information on the whiteboard system and interviews with a domain expert from the hospital, who observed the actual work of nurses. In this case study, we do not use all composition functions that our framework provides, as some do not apply to the whiteboard system. However, we believe that the unused functions are useful in many different settings. All 18 activity patterns are listed in Table 3 together with the number of process activities and the name of the modeled high-level activity. The examples introduced in Fig. 2 correspond to the activities *Shift* (p_a), *Alarm Normal* (p_b) and *Handover* (p_c) in the case study.

---

[2] The used whiteboard system is *Imatis Visi*: http://www.imatis.com.

[3] Plug-in *Log Abstraction* of the ProM package *LogEnhancement*: http://promtools.org.

4.2   Results and Discussion

We applied the proposed abstraction method to the event log and successfully obtained a smaller abstracted event log with 206,054 high-level events for 103,027 activity instances (i.e., each instance has a *start* and a *complete* event). The computation of the abstracted event log took one hour and used 6 GB of memory. We decomposed the DPN of the abstraction model into two smaller DPNs that did not share labels. The overall fitness with regard to the log was 0.91, which indicates that most of the observations could be explained. Even though 9 % of the events did not match, this is a good result for further analysis as we can expect the event log to contain noise, i.e., events unrelated to any modeled high-level activity. The abstracted event log contains 25 high-level activities: 18 activities were obtained through abstraction and 7 further activities were already at the appropriate level of abstraction. Table 3 shows the resulting number of activity instances that were matched, as well as the corresponding matching error. It should be noted that the relatively high error for the activity *Surgery* stems from the fact that this activity is sometimes recorded in a different manner, i.e., one event is missing. Regarding the error for activity *Alarm Assist* we found that the *assist* button can be pressed without a prior alarm by the patient, which is different from our initial assumption.

Table 3. Activity patterns used in the digital whiteboard case study. For each pattern we list the number of process activities, the number of low-level activity names shared with other patterns, and the results of our method: the number of recognized activity instances and the matching error.

Activity name	Transitions (shared)	Matches	Matching error (ε)
Announcement (Ann)	8 (6)	29	0.02
Change Room (CR)	5 (4)	662	0.09
Discharge (Dis)	7 (4)	8,054	0.0
Registration (Reg)	6 (6)	9,855	0.01
Transfer (Tra)	6 (6)	575	0.09
Update Report (UR)	4 (0)	6,912	0.0
Update Arrival (UA)	5 (1)	4,626	0.0
Handover (H)	1 (1)	24,228	0.0
Shift (S)	3 (3)	405	0.04
Call Nurse (CN)	2 (2)	12,416	0.08
Alarm Normal (AN)	3 (3)	8,842	0.02
Alarm Quick (AQ)	2 (2)	12,730	0.0
Alarm assist (AA)	5 (3)	32	0.17
CT	4 (2)	1,443	0.0
MRI	4 (2)	124	0.0
Surgery (Sur)	3 (3)	297	0.17
Ultrasound (Ult)	5 (3)	1,164	0.0
X-Ray	4 (2)	1,117	0.0

(a) Events *NC* (blue), *CS1* (green) and *CS0* (yellow) in the original event log.

(b) Abstracted event log only showing the high-level *Shift* events captured by pattern p_a.
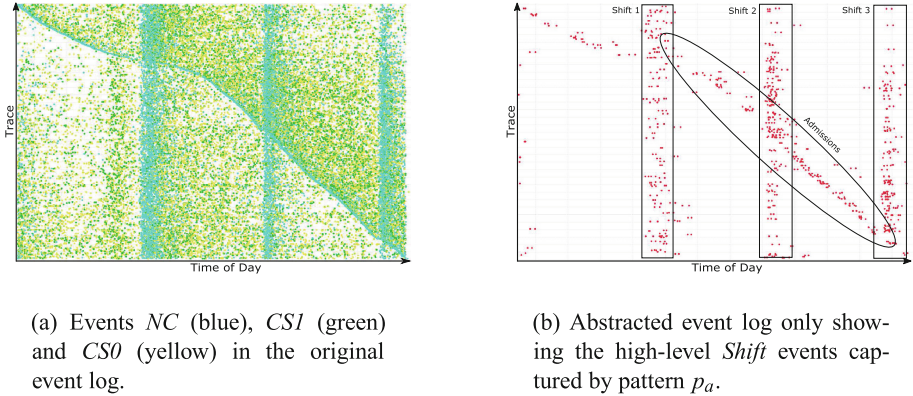
Fig. 6. Dotted charts of events related to the activity *Shift*. Traces are shown on the y-axis and sorted by the time of day of the first event in a trace. (Color figure online)

The activities under consideration can be grouped into three categories: (1) actions related to patient logistics, (2) actions related to the call signal system and handover between the nurses, and (3) actions related to ordered examinations and surgeries. Given the absence of a perfectly abstracted event log as ground truth, we evaluate our method by comparing the results obtained using three process analytics techniques *with* and *without* the abstraction. Using the abstracted event log, we obtained several insights into work practices of nurses in clinical processes. A domain expert from the hospital stated that the analysis: *"[... gives insight beyond the usual reports and analysis that we have access to. It gives a fresh and "new" perspective on how we understand the processes involved in running a ward or department."* By contrast, we show that using the low-level event log directly does not lead to any insights for stakeholders, because the semantics of low-level events are unclear. We used the ProM plug-ins *Log Projection* (LP), *Inductive Visual Miner* (IVM), and *Multi-perspective Explorer* (MPE).

Log Projection (LP). Fig. 6 shows two dotted charts created with LP. Figure 6(a) is created using the original event log. It shows the distribution of events *NC*, *CS1* and *CS0* over the course of a day. As expected, the *NC* event (i.e., the responsible nurse changed) mostly occurs when a patient is admitted (i.e., on the blue diagonal) and during one of the three shift changes (i.e., the three blue vertical lines). Still, the responsible nurse also changes between those well-defined times. Yet, from Fig. 6(a) it is not evident whether nurses use the call signal system when visiting a patient after their shift started. Looking at Fig. 6(b), which shows only the event *Shift* (p_a) from the abstracted event log, it is clearly visible that our assumption was correct. Activity pattern p_a captured a meaningful high-level activity. Figure 6(b) shows that nurses do use the call signal system to indicate their presence in the room of the patient after taking responsibility for a patient. In contrast to the dotted chart in Fig. 6(a), event *Shift* only occurs after admissions (dots on the main diagonal) and after shift changes (three vertical lines). Still, by comparing the number of activity instances in Table 3 it is clear that activity
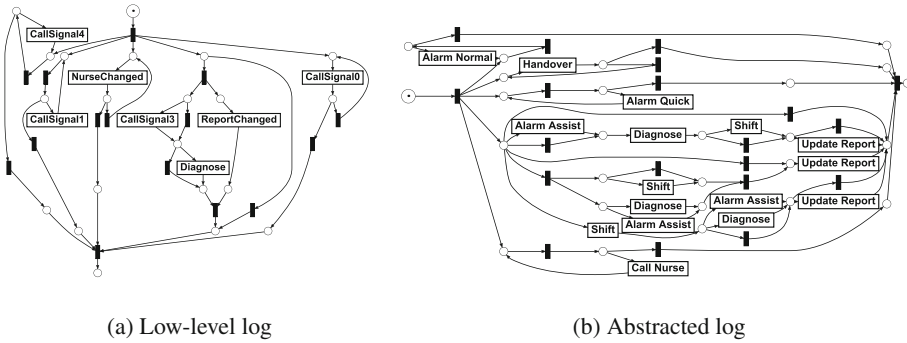
(a) Low-level log   (b) Abstracted log

Fig. 7. Petri nets of the nurse handover and call signal system discovered by IVM

Shift (405 times) happens rarely in comparison to activity *Handover* (24,228 times). Two likely reasons for this are that nurses do either not attend the patient after a shift change, or that they do not use the system to indicate their presence. This is a valuable insight on how the whiteboard system is used in practice. Notably, this could not be concluded without the use of our abstraction method.

Inductive Visual Miner (IVM). We analyzed two parts of the whiteboard system by discovering process models with IVM [15]. We used only those events from the original event log that are used in the respective activity patterns. This indicates what results could be obtained by only filtering the original log based on some knowledge about the low-level events. Figure 7 shows Petri nets discovered using IVM for events related to *nurse handovers and the call signal system based* on the original event log (Fig. 7(a)), and the abstracted event log (Fig. 7(b)). The model in Fig. 7(a) gives little insights into the usage of the call signal system. Most events can be repeated in any order, expect for *CallSignal3*, *Diagnose*, and *ReportChanged* which may only occur once. The model in Fig. 7(b), instead, contains recognizable activities that can be used to investigate the usage of the call signal system further. The model indicates that activities *Shift* and *Alarm Assist* occur together, recording a *diagnose* and updating the report cell (*Update Report*) that is used to store medical information about patients. This finding deserves further investigation. Regarding the other variants of using the call signal system (*Alarm Quick*, *Alarm Normal*, and *Call Nurse*), no specific ordering among the high-level activities is discovered. This could be expected given the flexible nature of the whiteboard system. Moreover, we compared models discovered for low-level and high-level events related to *examinations and patient logistics* (Fig. 8). Again, the model that is discovered from the low-level event log, shown in Fig. 8(a), does not offer insights into the work at the observation unit. The only visible structure is that event *TreatmentChanged*, which is related to some examination, is executed in parallel with the event *Abdom. Pain*. Figure 8(b) shows the process model discovered by IVM *with* the abstracted event log. The control-flow structure of this model is more specific than the model shown in Fig. 8(a). It shows several interesting structures that could be used to investigate further with people working in the hospital. For example, for multiple patients the planning and
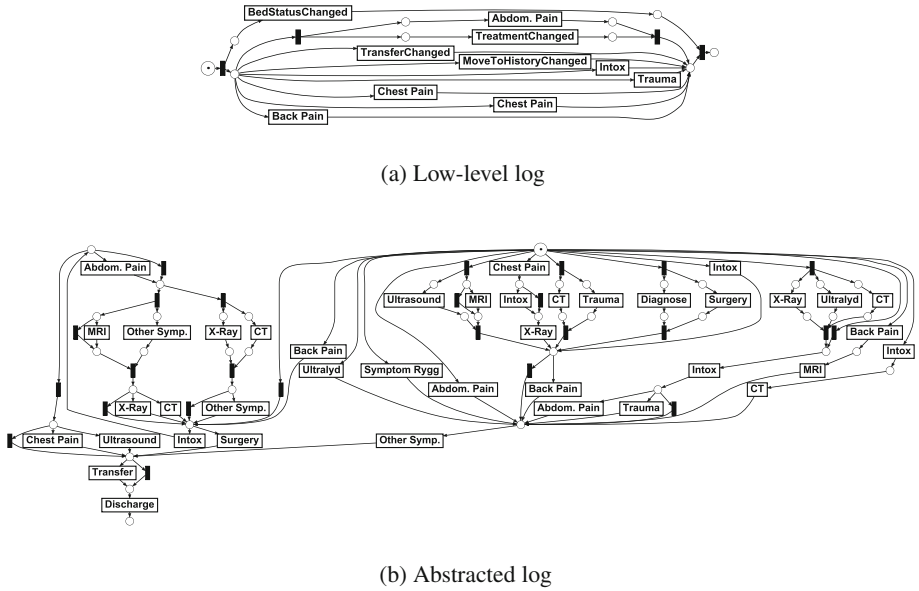
(a) Low-level log

(b) Abstracted log

Fig. 8. Petri nets of the examinations and patient logistics discovered by IVM

execution of *surgeries* and updating the *diagnose* occur together in parallel. One path reveals that 300 patients with chest pain receive an *X-Ray*. Another path shows that for 1,300 patients with abdominal pain no examinations are ordered. By using activities on the same abstraction level, the process model in Fig. 8(b) offers a better insight into the process. Moreover, it allows to discuss the observations with process workers. Please note that the high-level model in Fig. 8(b) contains more activities since the same event is shared among multiple, high-level activities.

Multi-Perspective Explorer (MPE). Finally, we used the MPE to analyze differences between the different ways nurses respond to patient-initiated call signals. It was found that the assumed activities *Alarm Normal* (AN) and *Alarm Quick* (AQ), indeed, correspond to different work practices by nurses. For activity AN the nurse first indicates her presence in the room by using a button on the call signal system, after which she attends the patient. However, within activity AQ nurses do not use this functionality. The average service time for activity AN (7.3 min) is longer than for activity AQ (1.5 min). A hypothesis is that nurses do not use the full functionality of the call signal system for minor tasks, which may be important for the hospital to investigate further.

5   Conclusion

We presented a new method for supervised *event abstraction* using behavioral activity patterns. Activity patterns encode assumptions on how high-level activities manifest themselves in terms of recorded low-level events. We obtain an abstracted event

log based on an alignment between activity patterns and the low-level event log. Two quality measures (fitness, matching error) are defined that can be used to evaluate the quality of the abstraction result. We used this method to analyze the work of nurses in a Norwegian hospital. The case study shows that our abstraction method can be successfully applied in complex real-life environments. We obtained an abstracted event log from a system, in which (1) multiple high-level activities share low-level events with the same label, (2) high-level activities occur concurrently, and (3) erroneous events (i.e., noise) are recorded. We applied state-of-the-art process mining tools on both the original and the abstracted event log The results obtained from the abstracted even log reveal insights that cannot be obtained when using the original event log. Moreover, the results are more useful in the communication with stakeholders, since they refer to recognizable activities. Future work may still be needed to address some limitations of our method. At this point, if a sequence of events fits two activity patterns perfectly, one of them will be chosen arbitrarily. A prioritization of activity patterns used during the alignment computation could be introduced. Moreover, alignment techniques require a lot of resources for event logs with very long traces. Work on decomposing the alignment computation could help to alleviate this limitation.

References

1. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer, Berlin (2011)
2. Günther, C.W., Rozinat, A., van der Aalst, W.M.P.: Activity mining by global trace segmentation. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 128–139. Springer, Heidelberg (2010)
3. Baier, T., Mendling, J., Weske, M.: Bridging abstraction layers in process mining. Inf. Syst. **46**, 123–139 (2014)
4. Jagadeesh Chandra Bose, R.P., van der Aalst, W.M.P.: Abstractions in process mining: a taxonomy of patterns. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 159–175. Springer, Heidelberg (2009)
5. Cook, D.J., Krishnan, N.C., Rashidi, P.: Activity discovery and activity recognition: a new partnership. IEEE Trans. Cybern. **43**(3), 820–828 (2013)
6. Ferreira, D.R., Szimanski, F., Ralha, C.G.: Improving process models by mining mappings of low-level events to high-level activities. J. Intell. Inf. Syst. **43**(2), 379–407 (2014)
7. Folino, F., Guarascio, M., Pontieri, L.: Mining multi-variant process models from low-level logs. In: Abramowicz, W. (ed.) BIS 2015. LNBIP, vol. 208, pp. 165–177. Springer, Heidelberg (2015)
8. Baier, T., Rogge-Solti, A., Mendling, J., Weske, M.: Matching of events and activities: an approach based on behavioral constraint satisfaction. In: SAC, pp. 1225–1230. ACM (2015)
9. Ferreira, D.R., Szimanski, F., Ralha, C.G.: Mining the low-level behaviour of agents in high-level business processes. IJBPIM **6**(2), 146–166 (2013)
10. Fazzinga, B., Flesca, S., Furfaro, F., Masciari, E., Pontieri, L.: A probabilistic unified framework for event abstraction and process detection from log data. In: Debruyne, C., Panetto, H., Meersman, R., Dillon, T., Weichhart, G., An, Y., Ardagna, C.A. (eds.) OTM 2015 Conferences. LNCS, vol. 9415, pp. 320–328. Springer, Heidelberg (2015)

11. Baier, T.: Matching events and activities. Ph.D. thesis, Universität Potsdam (2015)
12. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. Computing **98**(4), 407–437 (2016)
13. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P., Toussaint, P.J.: From low-level events to activities - a pattern-based approach. Technical report, BPMcenter.org, BPM Center Report BPM-02-06 (2016)
14. Wong, H.J., Caesar, M., Bandali, S., Agnew, J., Abrams, H.: Electronic inpatient whiteboards: improving multidisciplinary communication and coordination of care. Int. J. Med. Inform. **78**(4), 239–247 (2009)
15. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Using life cycle information in process discovery. In: Reichert, M., Reijers, H. (eds.) BPM Workshops 2015. LNBIP, vol. 256, pp. 204–217. Springer, Heidelberg (2016)