Event Abstraction for Process Mining Using Supervised Learning Techniques

Niek Tax[1(✉)], Natalia Sidorova[1], Reinder Haakma[2], and Wil M.P. van der Aalst[1]

[1] Department of Mathematics and Computer Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{n.tax,n.sidorova,w.m.p.v.d.aalst}@tue.nl
[2] Philips Research, Prof. Holstlaan 4, 5665 AA Eindhoven, The Netherlands
reinder.haakma@philips.com

Abstract. Process mining techniques focus on extracting insight in processes from event logs. In many cases, events recorded in the event log are too fine-grained, causing process discovery algorithms to discover incomprehensible process models or process models that are not representative of the event log. We show that when process discovery algorithms are only able to discover an unrepresentative process model from a low-level event log, structure in the process can in some cases still be discovered by first abstracting the event log to a higher level of granularity. This gives rise to the challenge to bridge the gap between an original low-level event log and a desired high-level perspective on this log, such that a more structured or more comprehensible process model can be discovered. We show that supervised learning can be leveraged for the event abstraction task when annotations with high-level interpretations of the low-level events are available for a subset of the sequences (i.e., traces). We present a method to generate feature vector representations of events based on XES extensions, and describe an approach to abstract events in an event log with Condition Random Fields using these event features. Furthermore, we propose a sequence-focused metric to evaluate supervised event abstraction results that fits closely to the tasks of process discovery and conformance checking. We conclude this paper by demonstrating the usefulness of supervised event abstraction for obtaining more structured and/or more comprehensible process models using both real life event data and synthetic event data.

Keywords: Process mining · Event abstraction · Probabilistic graphical models

1 Introduction

Process mining is a fast growing discipline that combines knowledge and techniques from computational intelligence, data mining, process modeling and process analysis [1]. Process mining focuses on the analysis of event logs, which

© Springer International Publishing AG 2018
Y. Bi et al. (eds.), *Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016*, Lecture Notes in Networks and Systems 15, DOI 10.1007/978-3-319-56994-9_18

consists of sequences of real-life events observed from process executions, originating e.g. from logs from ERP systems. An important subfield of process mining is process discovery, which is concerned with the task of finding a process model that is representative of the behavior seen in an event log. Many different process discovery algorithms exist [2–6], and many different types of process models can be discovered by process discovery methods, including BPMN models, Petri nets, process trees, and statecharts.

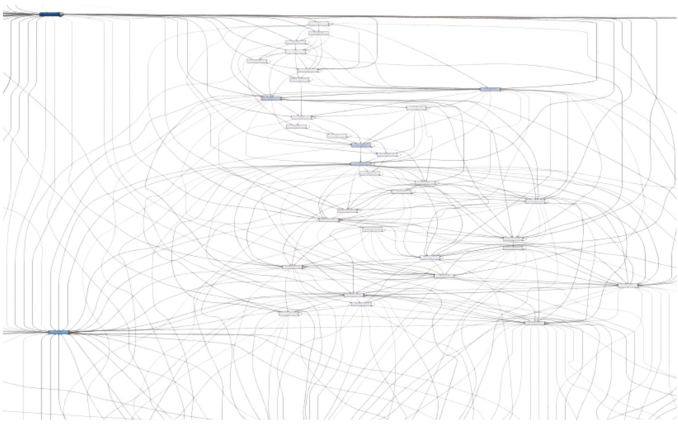


Fig. 1. An excerpt of a "spaghetti"-like process model.

As event logs are often not generated specifically for the application of process mining, events granularity of the event log at hand might be too low level. It is vital for successful application of process discovery techniques to have event logs at an appropriate level of abstraction. Process discovery techniques when the input event log is too low level might result in process model with one or more undesired properties. First of all, the resulting process model might be "spaghetti"-like, as shown in Fig. 1, containing of an uninterpretable mess of nodes and connections. The aim of process discovery is to discover a structured, "lasagna"-like, process model as shown in Fig. 2, which is much more interpretable than a "spaghetti"-like model. Secondly, the activities in the process model might have too specific, non-meaningful, names. Third, as we show in Sect. 4, process discovery algorithms are sometimes not able to discover a process model that represents the low-level event log well, while being able to discover to discover a representative process model from a corresponding high-level event log. The problems mentioned illustrate the need for a method to abstract too low-level event logs into higher level event logs.

Several methods have been explored within the process mining field that address the challenge of abstracting low-level events to higher level events [7–9]. Existing event abstraction methods rely on unsupervised learning techniques to abstract low-level into high-level events by clustering together groups of low-level events into one high-level event. However, using unsupervised learning

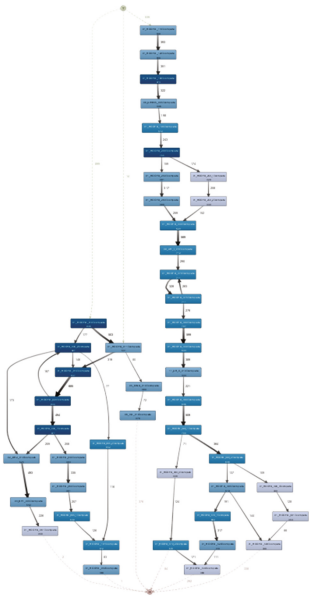


Fig. 2. A structured, or "lasagna"-like, process model.

introduces two new problems. First, it is unclear how to label high-level events that are obtained by clustering low-level events. Current techniques require the user/process analyst to provide high-level event labels themselves based on domain knowledge, or generate long labels by concatenating the labels of all low-level events incorporated in the cluster. However, long concatenated labels quickly become unreadable for larger clusters, and it is far from trivial for a user to come up with sensible labels manually. In addition, unsupervised learning approaches for event abstraction give no guidance with respect to the desired level of abstraction. Many existing event abstraction methods contain one or more parameters to control the degree in which events are clustered into higher level events. Finding the right level of abstraction providing meaningful results is often a matter of trial-and-error.

In some cases, training data with high-level target labels of low-level events are available, or can be obtained, for a subset of the traces. In many settings, obtaining high-level annotations for all traces in an event log is infeasible or too expensive. Learning a supervised learning model on the set of traces where high-level target labels are available, and applying that model to other low-level traces where no high-level labels are available, allows us to build a high-level interpretation of a low-level event log, which can then be used as input for process mining techniques.

In this paper we describe a method for supervised event abstraction that enables process discovery from too fine-grained event logs. This method can be applied to any event log where higher level training labels of low level events

are available for a subset of the traces in the event log. We start by giving an overview of related work from the activity recognition field in Sect. 2. In Sect. 3 we introduce basic concepts and definitions used throughout the rest of the paper. Section 4 explains the problem of not being able to mine representative process models from low-level data in more detail. In Sect. 5 we describe a method to automatically retrieve a feature vector representation of an event that can be used with supervised learning techniques, making use of certain aspects of the XES standard definition for event logs [10]. In the same section we describe a supervised learning method to map low-level events into target high-level events. Sections 6 and 7 respectively show the added value of the described supervised event abstraction method for process mining on a real life event log from a smart home environment and on a synthetic log from a digital photocopier respectively. Section 8 concludes the paper.

2 Related Work

Supervised event abstraction is an unexplored problem in process mining. A related field is activity recognition within the field of ubiquitous computing. Activity recognition focuses on the task of detecting human activity from either passive sensors [11,12], wearable sensors [13,14], or cameras [15]. Activity recognition methods generally work on discrete time windows over the time series of sensor values and aim to map each time window onto the correct type of human activity, e.g. *eating* or *sleeping*. Activity recognition methods can be classified into probabilistic approaches [11–14] and approaches based on ontology reasoning [16,17]. The strength of probabilistic system based approaches compared to methods based on ontology reasoning is their ability to handle noise, uncertainty and incomplete in sensor data [16].

Tapia [12] was the first to explore supervised learning methods to infer human activity from passive sensors, using a naive Bayes classifier. More recently, probabilistic graphical models started to play an important role in the activity recognition field [11,18]. Van Kasteren et al. [11] explored the use Conditional Random Fields [19] and Hidden Markov Models [20]. Van Kasteren and Kröse [18] applied Bayesian Networks [21] on the activity recognition task. Kim et al. [22] found Hidden Markov Models to be incapable of capturing long-range or transitive dependencies between observations, which results in difficulties recognizing multiple interacting activities (concurrent or interwoven). Conditional Random Fields do not posses these limitations.

The main differences between existing work in activity recognition and the approach presented in this paper are the input data on which they can be applied and the generality of the approach. Activity recognition techniques consider the input data to be a multidimensional time series of the sensor values over time based on which time windows are mapped onto human activities. An appropriate time window size is determined based on domain knowledge of the data set. In supervised event abstraction we aim for a generic method that works for all XES event logs in general. A time window based approach contrasts with our

aim for generality, as no single time window size will be appropriate for all event logs. Furthermore, the durations of the events within a single event log might differ drastically (e.g. one event might take seconds, while another event takes months), in which case time window based approaches will either miss short events in case of larger time windows or resort to very large numbers of time windows resulting in very long computational time. Therefore, we map each individual low-level event to a high-level event and do not use time windows. In a smart home environment context with passive sensors, each change in a binary sensor value can be considered to be a low-level event.

3 Preliminaries

In this section we introduce basic concepts used throughout the paper.

We use the usual sequence definition, and denote a sequence by listing its elements, e.g. we write $\langle a_1, a_2, \ldots, a_n \rangle$ for a (finite) sequence $s : \{1, \ldots, n\} \to S$ of elements from some alphabet S, where $s(i) = a_i$ for any $i \in \{1, \ldots, n\}$.

3.1 XES Event Logs

We use the XES standard definition of event logs, an overview of which is shown in Fig. 3. XES defines an event *log* as a set of *traces*, which in itself is a sequence of *event*s. The log, traces and events can all contain one or more *attribute*s, which consist of a *key* and a *value* of a certain type. Event or trace attributes may be *global*, which indicates that the attribute needs to be defined for each event or trace respectively. A log contains one or more *classifier*s, which can be seen as labeling functions on the events of a log, defined on global event attributes. *Extension*s define a set of attributes on log, trace, or event level, in such a way that the semantics of these attributes are clearly defined. One can view XES extensions as a specification of attributes that events, traces, or event logs themselves frequently contain. XES defines the following standard extensions:

Concept	Specifies the generally understood name of the event/trace/log (attribute 'Concept:name').
Lifecycle	Specifies the lifecycle phase (attribute 'Lifecycle:transition') that the event represents in a transactional model of their generating activity. The *Lifecycle* extension also specifies a standard transactional model for activities.
Organizational	Specifies three attributes for events, which identify the actor having caused the event (attribute 'Organizational:resource'), his role in the organization (attribute 'Organizational:role'), and the group or department within the organization where he is located (attribute 'Organizational:group').
Time	Specifies the date and time at which an event occurred (attribute 'Time:timestamp').

Semantic Allows definition of an activity meta-model that specifies higher-level aggregate views on events (attribute 'Semantic:modelReference').

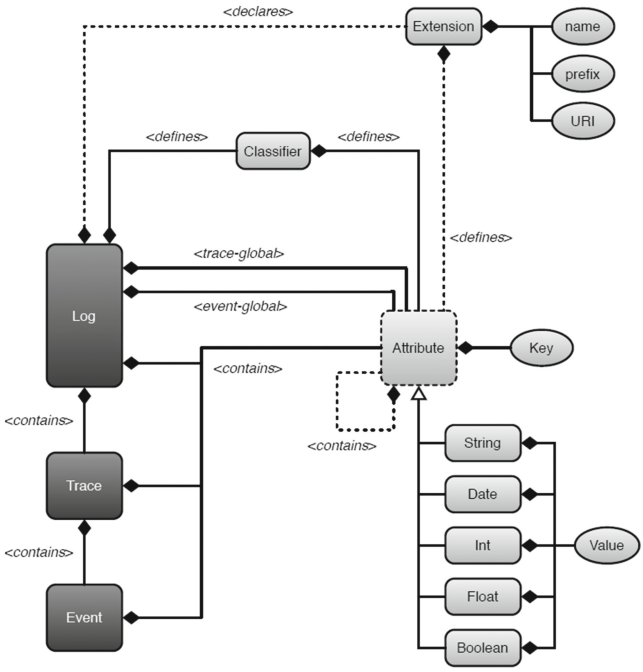


Fig. 3. XES event log meta-model, as defined in [10].

We introduce a special attribute of type *String* with key *label*, which represents a high-level version of the generally understood name of an event. The *concept* name of a event is then considered to be a low-level name of an event. The *Semantic* extension closely resembles the *label* attribute, however, by specifying relations between low-level and high-level events in a meta-model, the *Semantic* extension assumes that all instances of a low-level event type belong to the same high-level event type. The *label* attribute specifies the high-level label for each event individually, allowing for example one low-level event of low-level type *Dishes & cups cabinet* to be of high-level type *Taking medicine*, and another low-level event of the same type to be of high-level type *Eating*. Note that for some traces high-level annotations might be available, in which case its events contain the *label* attribute, while other traces might not be annotated. High-level interpretations of unannotated traces, by inferring the *label* attribute based on information that is present in the annotated traces, allow the use of unannotated traces for process discovery and conformance checking on a high level.

3.2 Petri Nets

A process modeling notation frequently used as output of process discovery techniques is the Petri net. Petri nets are directed bipartite graphs consisting of transitions and places, connected by arcs. Transitions represent activities, while places represent the status of the system before and after execution of a transition. Labels are assigned to transitions to indicate the type of activity that they represent. A special label τ is used to represent invisible transitions, which are only used for routing purposes and do not represent any real activity.

Definition 1 (Labeled Petri net). A labeled Petri net is a tuple $N = (P, T, F, R, \ell)$ where P is a finite set of places, T is a *finite set* of transitions such that $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called the flow relation, R is a finite set of labels representing event types, with $\tau \notin R$ is a label representing an invisible action, and $\ell : T \rightarrow R \cup \tau$ is a labeling function that assigns a label to each transition.

The state of a Petri net is defined w.r.t. the state that a process instance can be in during its execution. A state of a Petri net is captured by the marking of its places with tokens. In a given state, each place is either empty, or it contains a certain number of tokens. A transition is enabled in a given marking if all places with an outgoing arc to this transitions contain at least one token. Once a transition fires (i.e. is executed), a token is removed from all places with outgoing arcs to the firing transition and a token is put to all places with incoming arcs from the firing transition, leading to a new state.

Definition 2 (Marking, Enabled transitions, and Firing). A marked Petri net is a pair (N, M), where $N = (P, T, F, L, \ell)$ is a labeled Petri net and where $M \in \mathbb{B}(P)$ denotes the marking of N. For $n \in (P \cup T)$ we use $\bullet n$ and $n\bullet$ to denote the set of inputs and outputs of n respectively. Let $C(s, e)$ indicate the number of occurrences (count) of element e in multiset s. A transition $t \in T$ is enabled in a marking M of net N if $\forall p \in \bullet t : C(M, p) > 0$. An enabled transition t may fire, removing one token from each of the input places $\bullet t$ and producing one token for each of the output places $t\bullet$.

Figure 4 shows three Petri nets, with the circles representing places, the squares representing transitions. The black squares represent invisible transitions, or, τ transitions. Places annotated with an **f** belong to the final marking, indicating that the process execution can terminate in this marking.

The topmost Petri net in Fig. 4 initially has one token in the place $p1$, indicated by the dot. Firing of silent transition $t1$ takes the token from $p1$ and puts a token in both $p2$ and $p3$, enabling both $t2$ and $t3$. When $t2$ fires, it takes the token from $p2$ and puts a token in $p4$. When $t3$ fires, it takes the token from $p3$ and puts a token in $p5$. After $t2$ and $t3$ have both fired, resulting in a token in both $p4$ and $p5$, $t4$ is enabled. Executing $t4$ takes the token from both $p4$ and $p5$, and puts a token in $p6$. The **f** indicates that the process execution can stop in the marking consisting of this place. Alternatively, it can fire $t5$, taking the

token from $p6$ and placing a token in $p2$ and $p5$, which allows for execution of MC and W to reach the marking consisting of $p6$ again. We refer the interested reader to [23] for an extensive review of Petri nets.

3.3 Conditional Random Field

We view the recognition of high-level event labels as a sequence labeling task in which each event is classified as one of the higher-level events from a high-level event alphabet. Conditional Random Fields (CRFs) [19] are a type of probabilistic graphical model which has become popular in the fields of language processing and computer vision for the task of sequence labeling. A Conditional Random Field models the conditional probability distribution of the label sequence given an observation sequence using a log-linear model. We use Linear-chain Conditional Random Fields, a subclass of Conditional Random Fields that has been widely used for sequence labeling tasks, which takes the following form:

$$p(y|x) = \frac{1}{Z(x)} exp(\sum\nolimits_{t=1} \sum\nolimits_{k} \lambda_k f_k(t, y_{t-1}, y_t, x))$$

where $Z(x)$ is the normalization factor, $X = \langle x_1, \ldots, x_n \rangle$ is an observation sequence, $Y = \langle y_1, \ldots, y_n \rangle$ is the associated label sequence, f_k and λ_k respectively are feature functions and their weights. Feature functions, which can be binary or real valued, are defined on the observations and are used to compute label probabilities. In contrast to Hidden Markov Models [20], feature functions are not assumed to be mutually independent.

4 Motivating Example

Figure 4 shows on a simple example how a process can be structured at a high level while this structure is not discoverable from a low-level log of this process. The bottom right Petri net shows the example process at a high-level. The high-level process model allows for any finite length alternating sequence of *Taking medicine* and *Eating* activities. The *Taking medicine* high-level activity is defined as a subprocess, corresponding to the topmost Petri net, which consists of low-level events *Medicine cabinet (MC)*, *Dishes & cups cabinet (DCC)*, and *Water (W)*. The *Eating* high-level event is also defined as a subprocess, shown in the bottom left Petri net, which consists of low-level events *Dishes & cups cabinet (DCC)* and *Cutlery drawer (CD)* that can occur an arbitrary number of times in any order and low-level event *Dishwasher (D)* which occurs exactly once, but at an arbitrary point in the *Eating* process.

When we apply the Inductive Miner process discovery Algorithm [6] to low-level traces generated by the hierarchical process of Fig. 4, we obtain the process model shown in Fig. 5. The obtained process model allows for almost all possible sequences over the alphabet $\{CD, D, DCC, MC, W\}$, as the only constraint introduced by the model is that DCC and D are required to be executed starting

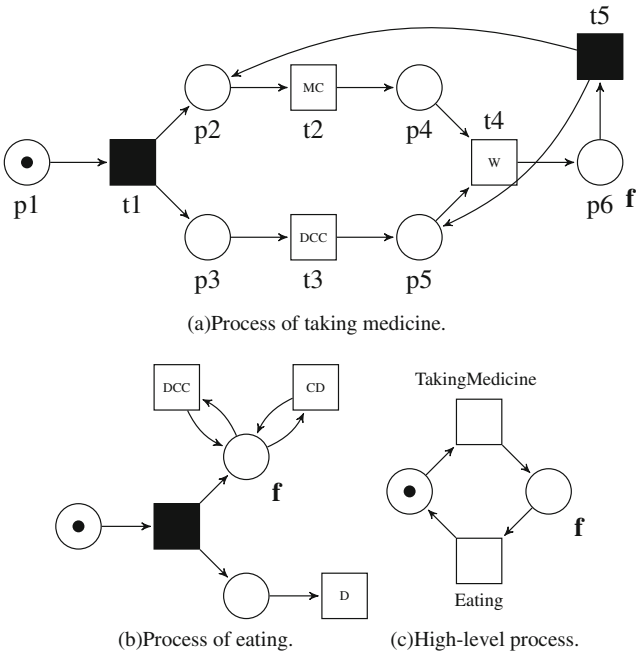


Fig. 4. A high-level process consisting of two unstructured subprocesses that overlap in event types.

from the initial marking to end up with the same marking. Firing of all other transitions in the model can be skipped. Behaviorally this model is very close to the so called "flower" model [1], the model that allows for all behavior over its alphabet. The alternating structure between *Taking medicine* and *Eating* that was present in the high-level process in Fig. 4 cannot be observed in the process model in Fig. 5. This is caused by high variance in start and end events of the high-level event subprocesses of *Taking medicine* and *Eating* as well as by the overlap in event types between these two subprocesses.

When the event log would have consisted of the high-level *Eating* and *Taking medicine* events, process discovery techniques have no problems to discover the alternating structure in the bottom right Petri net of Fig. 4. To discover the high-level alternating structure from a low-level event log it is necessary to first abstract the events in the event log. Through supervised learning techniques the mapping from low-level events to high-level events can be learned from examples, without requiring a hand-made ontology. Similar approaches have been explored in activity recognition in the field of ubiquitous computing, where low-level sensor signals are mapped to high-level activities from a human behavior perspective. The input data in this setting are continuous time series from sensors. Change points in these time series are triggered by low-level activities like *opening/closing the fridge door*, and the annotations of the higher

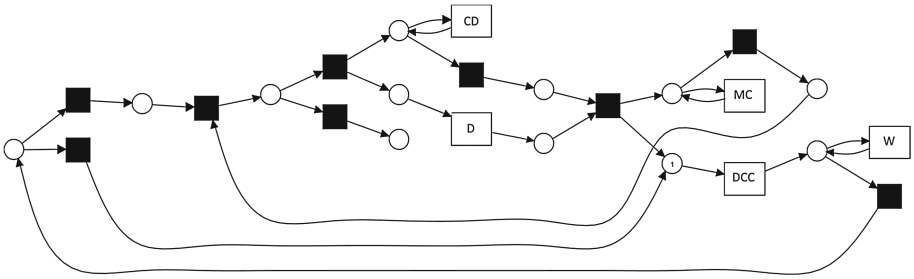


Fig. 5. Result of the Inductive Miner on the low-level traces, reduced using Murata reduction rules [24].

level events (e.g. *cooking*) are often obtained through manual activity diaries. In contrast to unsupervised event abstraction, the annotations in supervised event abstraction provide guidance on how to label higher level events and guidance for the target level of abstraction.

5 Event Abstraction as a Sequence Labeling Task

In this section we describe an approach to supervised abstraction of events based on Conditional Random Fields. Additionally, we describe feature functions on XES event logs in a general way by using XES extensions. Figure 6 provides a conceptual overview of the supervised event abstraction method. The approach takes two inputs, (1) a set of annotated traces, which are traces where the high-level event that a low-level event belongs to (the *label* attribute of the low-level event) is known for all low-level events in the trace, and (2) a set of unannotated traces, which are traces where the low-level events are not mapped to high-level events. Conditional Random Fields are trained of the annotated traces to create a probabilistic mapping from low-level events to high-level events. This mapping, once obtained, can be applied to the unannotated traces in order to estimate the corresponding high-level event for each low-level event (its *label* attribute). Often sequences of low-level events in the traces with high-level annotations will have the same *label* attribute. We make the working assumption that multiple high-level events are executed in parallel. This enables us to interpret a sequence of identical *label* attribute values as a single instance of a high-level event. To obtain a true high-level log, we *collapse* sequences of events with the same value for the *label* attribute into two events with this value as *concept* name, where the first event has a *lifecycle* 'start' and the second has the *lifecycle* 'complete'. Table 1 illustrates this collapsing procedure through an input and output event log.

The method described in this section is implemented and available for use as a plugin for the ProM 6 [25] process mining toolkit and is based on the GRMM [26] implementation of Conditional Random Fields.

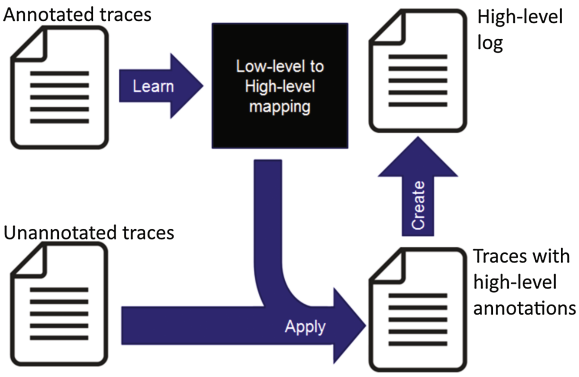


Fig. 6. Conceptual overview of supervised event abstraction.

Table 1. Left: a trace with predicted high-level annotations (*label*) and, Right: the resulting high-level log after collapsing subsequent identical label values.

Case	Time:timestamp	Concept:name	label
1	03/11/2015 08:45:23	Medicine cabinet	Taking medicine
1	03/11/2015 08:46:11	Dishes & cups cabinet	Taking medicine
1	03/11/2015 08:46:45	Water	Taking medicine
1	03/11/2015 08:47:59	Dishes & cups cabinet	Eating
1	03/11/2015 08:47:89	Dishwasher	Eating
1	03/11/2015 17:10:58	Dishes & cups cabinet	Taking medicine
1	03/11/2015 17:10:69	Medicine cabinet	Taking medicine
1	03/11/2015 17:11:18	Water	Taking medicine

Case	Time:timestamp	Concept:name	Lifecycle:transition
1	03/11/2015 08:45:23	Taking medicine	Start
1	03/11/2015 08:46:45	Taking medicine	Complete
1	03/11/2015 08:47:59	Eating	Start
1	03/11/2015 08:47:89	Eating	Complete
1	03/11/2015 17:10:58	Taking medicine	Start
1	03/11/2015 17:11:18	Taking medicine	Complete

We now show for each XES extension how it can be translated into useful feature functions for event abstraction. Note that we do not limit ourselves to XES logs that contain all XES extensions; when a log contains a subset of the extensions, a subset of the feature functions will be available for the supervised learning step. This approach leads to a feature space of unknown size, potentially causing problems related to the curse of dimensionality, therefore we use L1-regularized Conditional Random Fields. L1 regularization causes the vector of feature weights to be sparse, meaning that only a small fraction of the features have a non-zero weight and are actually used by the prediction model. Since the L1-norm is non-differentiable, we use OWL-QN [27] to optimize the model.

5.1 From a XES Log to a Feature Space

Concept Extension

The low-level labels of the preceding events in a trace can contain useful contextual information for high-level label classification. Based on the n-gram of n last-seen events in a trace, we can calculate the probability that the current event has a label l. A multinoulli distribution is estimated for each n-gram of n consecutive events, based on the training data. The Conditional Random Field model requires feature functions with numerical range. A concept extension based

feature function with two parameters, n and l, is valued with the multinoulli-estimated probability of the current event having high-level label l given the n-gram of the last n low-level event labels.

Organizational Extension
Similar to the concept extension feature functions, multinoulli distributions can be estimated on the training set for n-grams of *resource*, *role*, or *group* attributes of the last n events. Likewise, an organizational extension based feature function with three parameters, n-gram size n, $o \in \{resource, role, group\}$, and label l, is valued with the multinoulli-estimated probability of label l given the n-gram of the last n event resources/roles/groups.

Time Extension
In terms of time, there are several potentially existing patterns. A certain high-level event might for example be concentrated in a certain parts of a day, of a week, or of a month. This concentration can however not be modeled with a single Gaussian distribution, as it might be the case that a high-level event has high probability to occur in the morning or in the evening, but low probability to occur in the afternoon in-between. Therefore we use a Gaussian Mixture Model (GMM) to model the probability of a high-level label l given the timestamp. Bayesian Information Criterion (BIC) [28] is used to determine the number of components of the GMM, which gives the model an incentive to not combine more Gaussians in the mixture than needed. A GMM is estimated on training data, modeling the probabilities of each label based on the time passed since the start of the day, week or month. A time extension based feature function with two parameters, $t \in \{day, week, month, \dots\}$ and label l, is valued with the GMM-estimated probability of label l given the t view on the event timestamp.

Lifecycle Extension and Time Extension
The XES standard [10] defines several lifecycle stages of a process. When an event log possesses both the lifecycle extension and the time extension, time differences can be calculated between different stages of the life cycle of a single activity. For a *complete* event for example, one could calculate the time difference with the associated *start* event of the same activity. Finding the associated *start* event becomes nontrivial when multiple instances of the same activity are in parallel, as it is then unknown which *complete* event belongs to which *start* event. We assume consecutive lifecycle steps of activities running in parallel to occur in the same order as the preceding lifecycle step. For example, when we observe two *start* events of an activity of type A in a row, followed by two *complete* events of type A, we assume the first *complete* to belong to the first *start*, and the second *complete* to belong to the second *start*.

We estimate a Gaussian Mixture Model (GMM) for each tuple of two lifecycle steps for a certain activity on the time differences between those two lifecycle steps for this activity. A feature based on both the lifecycle and the time extension,

with a label parameter l and lifecycle c, is valued with the GMM-estimated probability of label l given the time between the current event and lifecycle c. Bayesian Information Criterion (BIC) [28] is again used to determine the number of components of the GMM.

5.2 Evaluating High-Level Event Predictions for Process Mining Applications

Existing approaches in the field of activity recognition take as input time windows where each time window is represented by a feature vector that describes the sensor activity or status during that time window. Hence, evaluation methods in the activity recognition field are window-based, using evaluation metrics like the percentage of correctly classified time slices [11,12,18]. There are two reasons to deviate from this evaluation methodology in a process mining setting. First, our method operates on events instead of time windows. Second, the accuracy of the resulting high level sequences is much more important for many process mining techniques (e.g. process discovery, conformance checking) than the accuracy of predicting each individual minute of the day.

We use *Levenshtein similarity* that expresses the degree in which two traces are similar using a metric based on the Levenshtein distance (also known as edit distance) [29], which is defined as $Levenshtein\_similarity(a, b) = 1 - \frac{Levenshtein\_distance(a,b)}{max(|a|,|b|)}$. The division of the Levenshtein distance by $max(|a|, |b|)$, which is the worst case number of edit operations needed to transform any sequence of length $|a|$ into any sequence of length $|b|$, causes the result to be a number between 0 (completely different traces) and 1 (identical traces).

6 Case Study 1: Smart Home Environment

We use the smart home environment log described by Van Kasteren et al. [11] to evaluate our supervised event log abstraction method. The Van Kasteren log consists of multidimensional time series data with all dimensions binary, where each binary dimension represents the state of an in-home sensor. These sensors include motion sensors, open/close sensors, and power sensors (discretized to 0/1 states).

6.1 Experimental Setup

We transform the multidimensional time series data from sensors into events by regarding each sensor change point as an event. Cases are created by grouping events together that occurred in the same day, with a cut-off point at midnight. High-level labels are provided for the Van Kasteren data set.

The generated event log based on the Van Kasteren data set has the following XES extensions:

Concept The sensor that generated the event.
Time The time stamp of the sensor change point.
Lifecycle *Start* when the event represents a sensor value change from 0 to 1 and *Complete* when it represents a sensor value change from 1 to 0.

Note that annotations are provided for all traces in the obtained event log. To evaluate how well the supervised event abstraction method generalized to unannotated traces, we artificially use a part of the traces to train the abstraction model and apply them on a test set where we regard the annotations to be non-existent. We evaluate the obtained high-level labels against the ground truth labels. We use a variation on Leave-One-Out-Cross-Validation where we leave out one trace to evaluate how well this mapping generalizes to unseen events and cases.

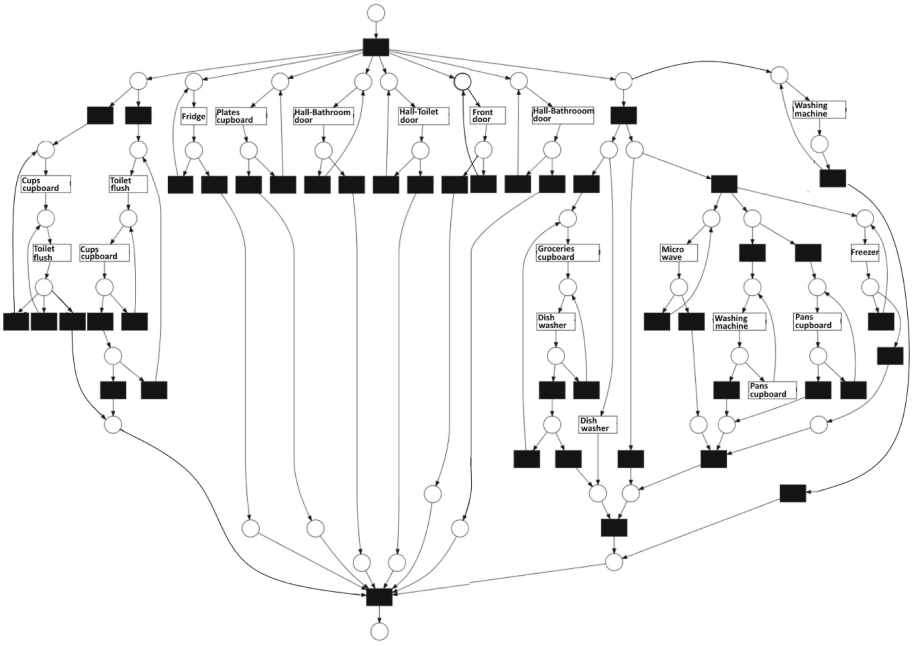
6.2 Results

Figure 7a shows the result of the Inductive Miner [6] for the low-level events in the Van Kasteren data set. The resulting process model starts with many parallel activities that can be executed in any order and contains many unobservable transitions back. This closely resembles the flower model, which allows for any behavior in any arbitrary order. From the process model we can learn that *toilet flush* and *cups cupboard* frequently co-exists. Furthermore, the process model indicates that *groceries cupboard* is often followed by *dishwasher*. There seems to be very little structure on this level of event granularity.

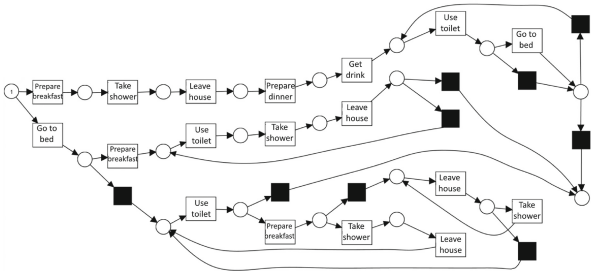
The average Levenshtein similarity between the predicted high-level traces in the Leave-One-Trace-Out-Cross-Validation experimental setup and the ground truth high-level traces is 0.7042, which shows that the supervised event abstraction method produces traces which are fairly similar to the ground truth.

Figure 7b shows the result of the Inductive Miner on the aggregated set of predicted test traces. Figure 7b shows that the process discovered at the high level of granularity is more structured than the process discovered at the original level of granularity (Fig. 7a). In Fig. 7b, we can see that the main daily routine starts with breakfast, followed by a shower, after which the subject leaves the house to go to work. After work the subject prepares dinner and has a drink. The subject mainstream behavior is to go to the toilet before going to bed, but he can then wake up later to go to the toilet and then continue sleeping. Note that the day can also start with going to bed. This is related to the case cut-off point of a trace at midnight. Days when the subject went to bed after midnight result in a case where going to bed occurs at the start of the trace. On these days, the subject might have breakfast and then perform the activity sequence use toilet, take shower, and leave house, possibly multiple times. Another possibility on days when the subject went to bed after midnight is that he starts by using the toilet, then has breakfast, then has the possibility to leave the house, then takes a shower, after which he always leaves the house. Prepare dinner activity is not performed on these days.

This case study shows that we can find a structured high-level process from a low-level event log where the low-level process is unstructured, using supervised event abstraction and process discovery.



(a) Inductive Miner result on the low-level events from the low-level Van Kasteren event log.



(b) Inductive Miner result on the high-level events discovered from the low-level van Kasteren event log

Fig. 7. Comparison of process models discovered from the low-level and high-level Van Kasteren event log.

7 Case Study 2: Artificial Digital Photocopier

Bose et al. [30,31] created a synthetic event log based on a digital photocopier to evaluate his unsupervised methods of event abstraction. In this case study we show that the described supervised event abstraction method can accurately abstract to high-level labels.

7.1 Experimental Setup

We annotated each low-level event with the correct high-level event using domain knowledge from the actual process model as described by Bose et al. [30,31]. This event log is generated by a hierarchical process, where high-level events *Capture Image*, *Rasterize Image*, *Image Processing* and *Print Image* are defined in terms of a process model. The *Print Image* subprocess amongst others contains the events *Writing*, *Developing* and *Fusing*, which are themselves defined as a subprocess. In this case study we set the task to transform the log such that subprocesses *Capture Image*, *Rasterize Image* and *Image Processing*, *Writing*, *Fusing* and *Developing*. Subprocesses *Writing* and *Developing* both contain the low-level event types *Drum Spin Start* and *Drum Spin Stop*. In this case study we focus in particular on the *Drum Spin Start* and *Drum Spin Stop* events, as they make the abstraction task non-trivial in the sense that no one-to-one mapping from low-level to high-level events exists.

The artificial digital photocopier data set has the concept, time and lifecycle XES extensions. On this event log annotations are available for all traces. On this data set we use a 10-Fold Cross-Validation setting on the traces to evaluate how well the supervised event abstraction method abstracts low-level events to high-level events on unannotated traces, as this data set is larger than the Van Kasteren data set and Leave-One-Out-Cross Validation would take too much time.

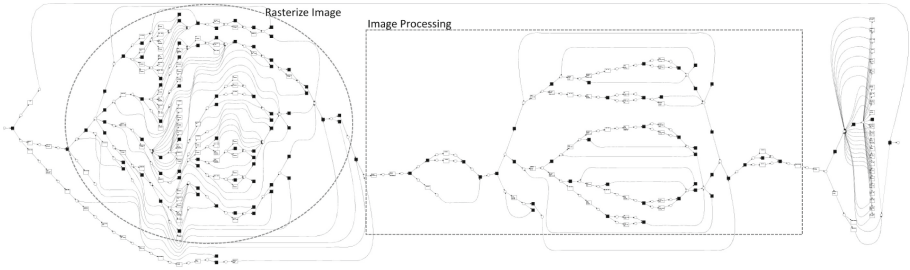
7.2 Results

The confusion matrix in Table 2 shows the aggregated results of the mapping of low-level events *Drum Spin Start* and *Drum Spin Stop* to high-level events *Developing* and *Writing*. The results show that the supervised event abstraction method is capable of detecting the many-to-many mappings between the low-level and high-level labels, as it maps these low-level events to the correct high-level event without making errors. The Levenshtein similarity between the aggregated set of test fold high-level traces and the ground truth high-level traces is close to perfect: 0.9667.

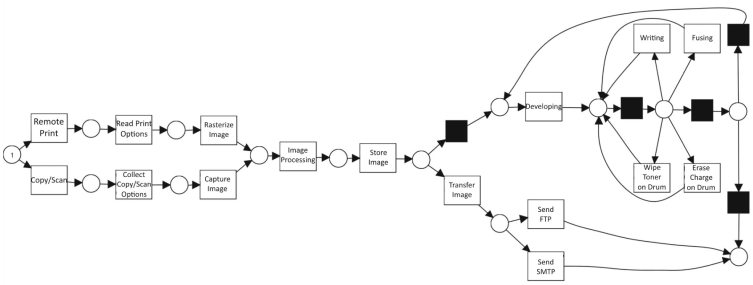
Table 2. Confusion matrix for classification of *Drum Spin Start* and *Drum Spin Stop* low-level events into high-level events *Writing* and *Developing*.

	Developing	Writing
Developing	6653	0
Writing	0	917

Figure 8a shows the process model obtained with the Inductive Miner on the low-level events in the artificial digital photocopier dataset. The two sections in the process model that are surrounded by dashed lines are examples of high-level events within the low-level process model. Even though the low-level process contains structure, the size of the process model makes it hard to comprehend.



(a) Inductive Miner result on the low-level Artificial Digital Photo Copier event log.



(b) Inductive Miner result on the discovered high-level events on the Artificial Digital Photo Copier event log.

Fig. 8. Comparison of process models discovered from the low-level and high-level Artificial Digital Photo Copier event log.

Figure 8b shows the process model obtained with the same process discovery algorithm on the aggregated high-level test traces of the 10-fold cross validation setting. This model is in line with the official artificial digital photocopier model specification, with the *Print Image* subprocess unfolded, as provided in [30,31]. In contrast to the event abstraction method described by Bose et al. [31] which found the high-level events that match specification, supervised event abstraction is also able to find suitable event labels for the generated high-level events. This allows us to discover human-readable process models on the abstracted events without performing manual labeling, which can be a tedious task and requires domain knowledge.

Instead of event abstraction on the level of the event log, unsupervised abstraction methods that work on the level of a model (e.g. [32]) can also be applied to make large complex models more comprehensible. Note that such methods also do not give guidance on how to label resulting transitions in the process model. Furthermore, such methods do not help in cases where the process on a low-level is unstructured, like in the case study as described in Sect. 6.

This case study shows that supervised event abstraction can help generating a comprehensible high-level process model from a low-level event log, when a low-level process model would be too large to be understandable.

8 Conclusion

In this paper we described a method to abstract events in a XES event log that is too low-level, based on supervised learning. The method consists of an approach to generate a feature representation of a XES event, and of a Conditional Random Field based learning step. An implementation of the method described has been made available as a plugin to the ProM 6 process mining toolkit. We introduced an evaluation metric for predicted high-level traces that is closer to process mining than time-window based methods that are often used in the sequence labeling field. Using a real life event log from a smart home domain, we showed that supervised event abstraction can be used to enable process discovery techniques to generate high-level process insights even when process models discovered by process mining techniques on the original low-level events are unstructured. Finally, we showed on a synthetic event log that supervised event abstraction can be used to discover smaller, more comprehensible, high-level process models when the process model discovered on low level events is too large to be interpretable.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Heidelberg (2011)
2. van der Aalst, W.M.P., Weijters, A.J.M.M., Maruster, L.: Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng. **16**(9), 1128–1142 (2004)
3. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining-adaptive process simplification based on multi-perspective metrics. In: Business Process Management, pp. 328–343. Springer (2007)
4. Van der Werf, J.M.E.M., van Dongen, B.F., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. In: Applications and Theory of Petri Nets, pp. 368–387. Springer (2008)
5. Weijters, A.J.M.M., Ribeiro, J.T.S.: Flexible heuristics miner (FHM). In: Proceedings of the 2011 IEEE Symposium on Computational Intelligence and Data Mining, pp. 310–317. IEEE (2011)
6. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs—a constructive approach. In: Application and Theory of Petri Nets and Concurrency. LNCS, pp. 311–329. Springer (2013)
7. Bose, R.P.J.C., van der Aalst, W.M.P.: Abstractions in process mining: a taxonomy of patterns. In: Business Process Management. LNCS, pp. 159–175. Springer (2009)
8. Günther, C.W., Rozinat, A., van der Aalst, W.M.P.: Activity mining by global trace segmentation. In: Business Process Management Workshops. LNBIP, pp. 128–139. Springer (2010)
9. van Dongen, B.F., Adriansyah, A.: Process mining: fuzzy clustering and performance visualization. In: Business Process Management Workshops. LNBIP, pp. 158–169. Springer (2010)
10. Günther, C.W., Verbeek, H.M.W.: XES-standard definition (2014). BPMcenter.org
11. van Kasteren, T., Noulas, A., Englebienne, G., Kröse, B.: Accurate activity recognition in a home setting. In: Proceedings of the 10th International Conference on Ubiquitous Computing, pp. 1–9. ACM (2008)

12. Tapia, E.M., Intille, S.S., Larson, K.: Activity recognition in the home using simple and ubiquitous sensors. In: Ferscha, A., Mattern, F. (eds.) Pervasive Computing. LNCS, pp. 158–175. Springer (2004)
13. Bao, L., Intille, S.S.: Activity recognition from user-annotated acceleration data. In: Ferscha, A., Mattern, F. (eds.) Pervasive Computing. LNCS, pp. 1–17. Springer (2004)
14. Kwapisz, J.R., Weiss, G.M., Moore, S.A.: Activity recognition using cell phone accelerometers. ACM SIGKDD Explor. Newslett. **12**(2), 74–82 (2011)
15. Poppe, R.: A survey on vision-based human action recognition. Image Vis. Comput. **28**(6), 976–990 (2010)
16. Chen, L., Nugent, C.: Ontology-based activity recognition in intelligent pervasive environments. Int. J. Web Inf. Syst. **5**(4), 410–430 (2009)
17. Riboni, D., Bettini, C.: OWL 2 modeling and reasoning with complex human activities. Pervasive Mob. Comput. **7**(3), 379–395 (2011)
18. van Kasteren, T., Kröse, B.: Bayesian activity recognition in residence for elders. In: Proceedings of the 3rd IET International Conference on Intelligent Environments, pp. 209–212. IEEE (2007)
19. Lafferty, J., McCallum, A., Pereira, F.C.N.: Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: Proceedings of the 18th International Conference on Machine Learning. Morgan Kaufmann (2001)
20. Rabiner, L.R., Juang, B.-H.: An introduction to hidden Markov models. ASSP Mag. **3**(1), 4–16 (1986)
21. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Mach. Learn. **29**(2–3), 131–163 (1997)
22. Kim, E., Helal, S., Cook, D.: Human activity recognition and pattern discovery. Pervasive Comput. **9**(1), 48–53 (2010)
23. Reisig, W.: Petri Nets: An Introduction, vol. 4. Springer, New York (2012)
24. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
25. Verbeek, H.M.W., Buijs, J.C.A.M., Van Dongen, B.F., van der Aalst, W.M.P.: ProM 6: the process mining toolkit. In: Proceedings of the Business Process Management Demonstration Track, pp. 34–39 (2010). CEUR-WS.org
26. Sutton, C.: GRMM: graphical models in mallet (2006). http://mallet.cs.umass.edu/grmm
27. Andrew, G., Gao, J.: Scalable training of L1-regularized log-linear models. In: Proceedings of the 24th International Conference on Machine Learning, pp. 33–40. ACM (2007)
28. Schwarz, G.: Estimating the dimension of a model. Ann. Stat. **6**(2), 461–464 (1978)
29. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady **10**, 707–710 (1966)
30. Bose, R.P.J.C., Verbeek, H.M.W., van der Aalst, W.M.P.: Discovering hierarchical process models using ProM. In: IS Olympics: Information Systems in a Diverse World, pp. 33–48. LNBIP. Springer (2012)
31. Bose, R.P.J.C.: Process mining in the large: preprocessing, discovery, and diagnostics. Ph.D. dissertation, Technische Universiteit Eindhoven (2012)
32. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data Knowl. Eng. **68**(9), 793–818 (2009)