ARTICLE IN PRESS

Information Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/infosys

A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs

Massimiliano de Leoni [a,*], Wil M.P. van der Aalst [a], Marcus Dees [b]

[a] Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, The Netherlands
[b] Uitvoeringsinstituut Werknemersverzekeringen (UWV), The Netherlands

ARTICLE INFO

Keywords:
Process mining
Decision and regression trees
Event-log manipulation
Event-log clustering

ABSTRACT

Process mining can be viewed as the missing link between model-based process analysis and data-oriented analysis techniques. Lion's share of process mining research has been focusing on process discovery (creating process models from raw data) and replay techniques to check conformance and analyze bottlenecks. These techniques have helped organizations to address compliance and performance problems. However, for a more refined analysis, it is essential to *correlate different process characteristics*. For example, do deviations from the normative process cause additional delays and costs? Are rejected cases handled differently in the initial phases of the process? What is the influence of a doctor's experience on treatment process? These and other questions may involve process characteristics related to different perspectives (control-flow, data-flow, time, organization, cost, compliance, etc.). Specific questions (e.g., predicting the remaining processing time) have been investigated before, but a generic approach was missing thus far. The proposed framework unifies a number of approaches for correlation analysis proposed in literature, proposing a general solution that can perform those analyses and many more. The approach has been implemented in ProM and combines process and data mining techniques. In this paper, we also demonstrate the applicability using a case study conducted with the UWV (Employee Insurance Agency), one of the largest "administrative factories" in The Netherlands.

1. Introduction

Process Aware Information Systems (PAISs) are increasingly used by organizations to support their businesses. Some of these systems are driven by process models, e.g., Business Process Management (BPM) and Workflow Management (WFM) systems. However, in most PAISs only an *implicit* process notion exists. Consider for example the enterprise software SAP and Oracle that is widely used to manage business operations and customer relations. Although these systems provide BPM/WFM functionality, most processes are partly hard-coded in application software and exist only in the minds of the people using the software. This provides flexibility and simplifies implementation efforts, but also results in poor management support. If there is no *explicit* process model that reflects reality, it is impossible to reason about compliance and performance in a precise and unified manner. Management dashboards provided by Business Intelligence (BI) software tend to oversimplify reality and do not use explicit process models. Fortunately, all these systems record the execution of process instances in so-called *event logs*. These logs thus capture information about

* Corresponding author.
E-mail addresses: m.d.leoni@tue.nl (M. de Leoni), w.m.p.v.d.aalst@tue.nl (W.M.P. van der Aalst), marcus.dees@uwv.nl (M. Dees).

http://dx.doi.org/10.1016/j.is.2015.07.003
0306-4379/© 2015 Elsevier Ltd. All rights reserved.

activities performed. Each event records the execution of an activity instance by a given resource at a certain point in time along with the output produced.

Event logs are the key enabler for *process mining*, which is capable of extracting, from event logs, in-depth insights in process-related problems that contemporary enterprises face [1]. Through the application of process mining, organizations can discover the processes as they were conducted in reality, check whether certain practices and regulations were really followed and gain insight into bottlenecks, resource utilization, and other performance-related aspects of processes.

Process mining often starts with *process discovery*, i.e., automatically learning process models based on raw event data. Once there is a process model (discovered or made by hand), the events can be replayed on the model to *check conformance* and to *uncover bottlenecks* in the process [1]. However, such analyses are often only the starting point for providing initial insights. When discovering a bottleneck or frequent deviation, one would like to understand why it exists. This requires the correlation of different *process characteristics*. These characteristics can be based on:

- the *control-flow* perspective (e.g., the next activity going to be performed);
- the *data-flow* perspective (e.g., the age of the patient or the amount of glucose in a blood sample);
- the *time* perspective (e.g., the activity duration or the remaining time to the end of the process);
- the *resource/organization* perspective (e.g., the resource going to perform a particular activity or the current workload), or,
- if a normative process model exists, the *conformance* perspective (e.g., the skipping of a mandatory activity or executing two activities in the wrong order).

There are of course other perspectives possible (e.g., the cost perspective), but these are often not process-specific and can be easily encoded in the data-flow perspective. For example, there may be data attributes capturing variable and fixed costs of an activity.

These problems are specific instances of a more general problem, which is concerned with *relating any process or event characteristic to other characteristics associated with single events or the entire process*. This paper proposes a framework to solve the more general correlation problem and provides a very powerful tool that unifies the numerous ad hoc approaches described in literature. This is achieved by providing (1) a broad and extendable set of characteristics related to control-flow, data-flow, time, resources, organizations and conformance, and (2) a generic framework where any characteristic (dependent variable) can be explained in terms of correlations with any set of other characteristics (independent variables). For instance, the involvement of a particular resource or routing decision can be related to the elapsed time, but also the other way around: the elapsed time can be related to resource behavior or routing.

The approach is fully supported by a new package that has been added to the open-source process mining framework *ProM*.[1] The evaluation of our approach is based on two case studies involving UWV, a Dutch governmental institute in charge of supplying benefits. For the first case study, the framework allows us to successfully answer process-related questions related to causes of observed problems within UWV (e.g., reclamations of customers). For some problems, we could show surprising root causes. For other problems, we could only show that some suspected correlations were not present, thus providing novel insights. A second case study was concerned with discovering the process model that describes the UWV's management of provisions of benefits for citizens who are unemployed and unable to look for a job for a relatively long period of time because of physical or mental illnesses. Analysis shows that there is a lot of variability in process execution, which is strongly related to the length of the benefit's provision. Therefore, the discovery of one single process led to unsatisfactory results as this variability cannot be captured in a single process model. After splitting the event log into clusters based on the distinguishing features discovered through our approach, the results significantly improved.

It is important to note that our framework does *not* enable analyses that previously were not possible. The novelty of our framework is concerned with providing a single environment where a broad range of process-centric analyses can be performed much quicker without requiring process analysts with a solid technical background. Without the framework, for instance, process analysts are confronted with database systems where they need to perform tedious operations to import data from external sources and, later, to carefully design complex SQL queries to perform joins and even self-joins that involve different database tables and views.

The remainder of the paper is as follows. Section 2 presents the overall framework proposed in this paper. The section also discusses how the problem of relating business process characteristics can be formulated and solved as a data-mining problem. In particular, we leverage of data-mining methods to construct a decision or regression tree. In the remainder, we refer them to as prediction trees if there is no reason to make a distinction. Our framework relies on the availability of key process characteristics. Therefore, Section 3 classifies and provides examples of these characteristics, along with showing how to extract them from raw event data. Section 4 presents event-selection filters to determine the instances of the learning problem. In Section 5 the filters and the process characteristics are used to provide an overview of the wide range of questions that can be answered using our framework. Several well-studied problems turn out to be specific instances of the more general problem considered in this paper. Section 6 discusses the notion of clustering parts of the log based on prediction trees. Section 7 illustrates the implementation of the framework as well as two case studies that have benefitted from the application of the framework. Finally, Section 8 positions

[1] See the *FeaturePrediction* package available in ProM 6.4 or newer, downloadable from http://www.promtools.org.

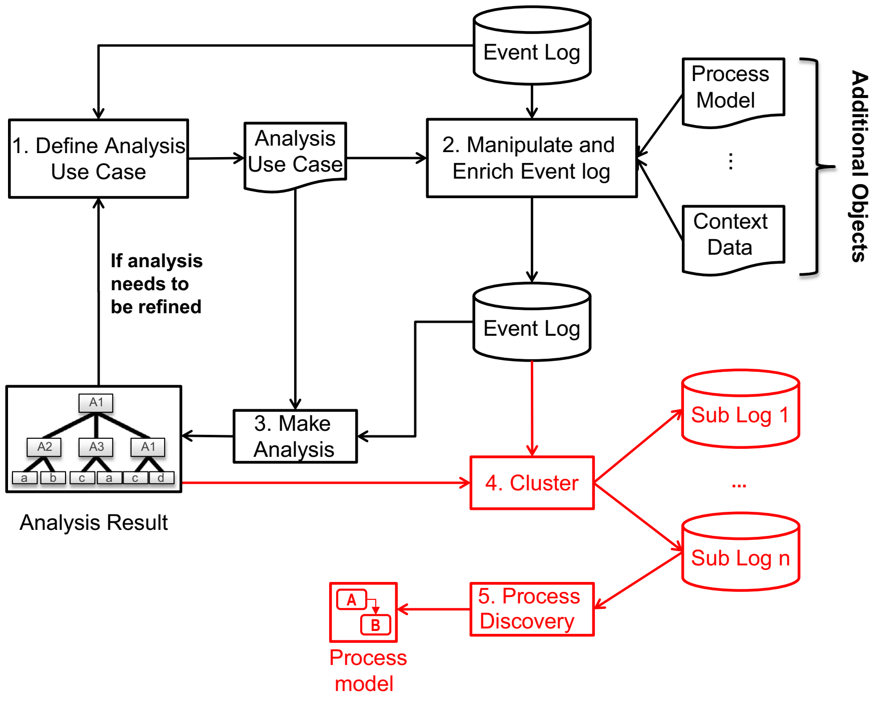


Fig. 1. The general framework proposed in this paper: based on an analysis use case the event log is preprocessed and used as input for classification. Based on the analysis result, the use case can be adapted to gather additional insights. The part in red (or gray if printed in gray scale) is optional. The result of the analysis use case provides a classification that can be used to split the event log into clusters: each tree's leaf becomes a different event log. Each sublog can be used by different process mining techniques (including, but not only, process discovery).

this work with respect to the state of the art and Section 9 concludes the paper.

2. A framework for correlating, predicting and clustering dynamic behavior based on event logs

This section introduces the framework reported on in this paper. To facilitate the understandability of this framework, Section 2.1 first provides a general overview, followed by Section 2.2 which provides a formalization of the framework. Section 2.3 then shows how the problem of performing an analysis use case can be transformed into a data-mining problem.

2.1. Overview

A high level overview of the framework proposed in this paper is shown in Fig. 1. Starting point is an *event log*. For each process instance (i.e., case) there is a trace, i.e., a sequence of events. Events have different *characteristics*. Mandatory characteristics are *activity* and *timestamp*. Other standard characteristics are the *resource* used to perform the activity, *transactional* information (start, complete, suspend, resume, etc.), and *costs*. However, any other characteristic can be associated to an activity (e.g., the age of a patient or size of an order). Characteristics are attached to events as name-value pairs: (*name_of_characteristic*, *value*). As mentioned in Section 1 and illustrated in Fig. 2(a), these characteristics can focus on the *control-flow* perspective, the *data-flow* perspective, the *resource/organization* perspective, the *time* perspective, and the *conformance* perspective.

Starting from the event log, a process analysis can define a so-called *analysis use case*, which requires the selection of a dependent characteristic, the independent characteristics and an event filter to describe which events to retain for the analysis.

Sometimes, such analysis requires the incorporation of characteristics that are not (yet) available in the event log. These characteristics can be added by deriving their values through computations over the event log or from external information sources. This motivates the need of the second step of our framework where the event log is manipulated and enriched with additional characteristics. As an example, the remaining time until case completion can be added to the event characteristics by simply comparing the time of the current event with the time of the last event for the same case. Another example is concerned with adding the workload of a resource to any event occurring at time t by scanning the event log and calculating the set of activity instances that are being executed by or are queued for the resource at time t.[2]

[2] Several definitions of workload are possible. Here and later in the paper, the workload of a resource r at time t is interpreted as the amount of work that is assigned to r at time t [2]. This accounts for activities that r is executing or that have been scheduled to r for later performance. The workload of a set of resources (e.g., within a department) is the sum of the workload of all resources in the set. Other workload definitions are possible as described in [2].

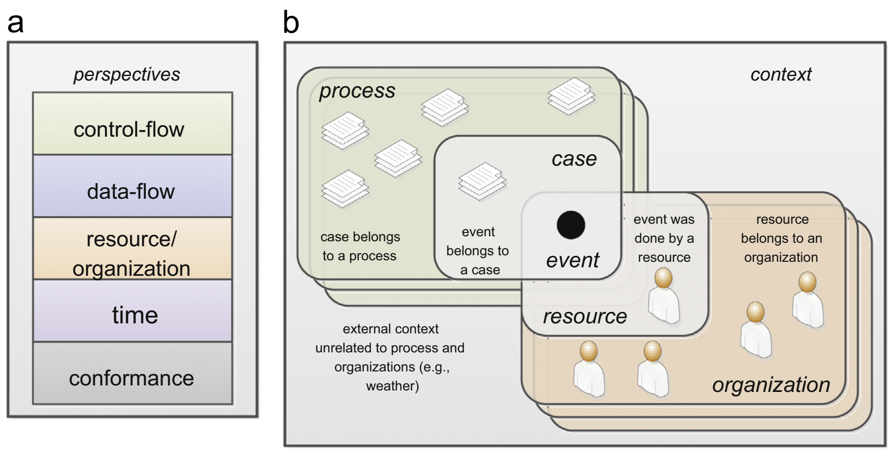

a
perspectives
control-flow
data-flow
resource/ organization
time
conformance
b
context
process
case
event
case belongs to a process
event belongs to a case
event was done by a resource
resource belongs to an organization
resource
organization
external context unrelated to process and organizations (e.g., weather)

Fig. 2. Overview of event characteristics. (a) Perspectives to which event characteristics refer. (b) Context of a selected event.

Fig. 2(b) provides an overview of the sources of information that can be used to add information to events. Event characteristics can derived over other events of the same process instance, other instances of the same process, instances of different processes but also over events executed by the same resource or by resources belonging to the same organization entity or to related organization entities. All of the above classes of event characteristics are solely based on the event log. However, event characteristics can be extracted through cross correlations with external data sources, as shown in Fig. 1. We may use a process model (e.g., for conformance checking) or context data (such as, the weather or stock market). For example, we may compute conformance metrics like fitness or precision and attach this to events. For instance, certain resources may be "meteoropathic": the level of their performance is influenced by weather conditions. It is worth analyzing whether, indeed, certain resources are more efficient when the weather is better. If we augment events with the weather conditions when they occurred, we can use the augmented log and check whether the efficiency of single resources is somehow related to the weather condition. This can be done by using the activity duration or resource workload as a dependent characteristic and the weather as one of the independent ones. If the characteristic occurs in the prediction tree, then there is some correlation between efficiency and weather. Some event characteristics (like weather information) cannot be derived from the event log. In general, the process context is acknowledged by many authors to be a huge source of valuable information to find correlations [3–5].

Once the analysis use case is defined and the event log is enriched, the analysis use case can be made. The outcome of performing the analysis is the generation of a decision or a regression tree, which aims to explain the values of the dependent characteristic as a function of the independent characteristics.

The analysis result obtained after the third step of the framework can be used for multiple purposes. For instance, the results can be used to derive which conditions may have a negative impact on certain KPIs; as an example, when certain process participants or entire company's branches are involved in execution of a given process, the customer's satisfaction is lower. As results, corrective actions can be put in place to avoid those conditions (e.g., the low-performing branches can be closed or the unsatisfactory process participants can be motivated, moved to different tasks or, as extreme solution, be fired). The results can also be used at run-time to provide decision support. For instance, suppose that certain conditions yield low values for certain KPIs. While executing the process, participants can be suggested to avoid certain decisions that have been observed to likely lead to those conditions. However, the production of the results is always done a posteriori using an event log that records past executions of process instances. If new process instances conclude, the result is not automatically updated. Conversely, the analysis needs to be repeated from scratch.

In addition to the three steps mentioned earlier, the general framework also envisions an optional fourth step that enables the clustering of traces of event logs on the basis of the correlation analysis results; see the red part (or gray if printed in gray scale) in Fig. 1. In many settings, an event-log clustering is an important preprocessing step for process mining. By grouping similar log traces together, it may be possible to construct partial process models that are easier to understand. If the event log shows a lot of behavioral variability, the process model discovered for all traces together is too complex to comprehend; in those cases, it makes sense to split the event log into clusters and apply discovering techniques to the log clusters, thus obtaining a simpler model for each cluster. This can be combined with process cubes where event data are stored in a multi-dimensional space [6]. Prediction trees provide a simple and visual way to perform such a clustering: each leaf is associated with a number of event-log traces. Therefore, a cluster is constructed for each leaf.

2.2. Formalization

As mentioned in Section 2.1, the main input of our framework is an event log. An event can have any number of attributes, here called characteristics. A trace is a sequence of events and an event log is a collection of traces.

Definition 1 (*Events, traces and log*). Let $\mathcal{C}$ and $\mathcal{U}$ be the universe of characteristics and the universe of possible values respectively. An *event e* is an assignment of values to characteristics, i.e. $e \in \mathcal{C} \nrightarrow \mathcal{U}$. In the remainder $\mathcal{E} = \mathcal{C} \nrightarrow \mathcal{U}$ is the universe of events. A *trace* $t \in \mathcal{E}^*$ is a sequence of events. Let $\mathcal{T} = \mathcal{E}^*$ be the universe of traces. An *event log L* is a multi-set of traces, i.e. $L \in \mathbb{B}(\mathcal{T})$.[3]

For each characteristic $c \in \mathcal{C}$, $type(c) \subseteq \mathcal{U}$ denotes the set of possible values. We use a special value $\perp$ for any characteristic c which an event e is not assigning a value to, i.e. $e(c) = \perp$ if $c \notin dom(e)$. This way $e \in \mathcal{E}$ can be used as a total function rather than a partial one. Note that $\perp \notin \mathcal{U}$.

Typically, an event refers to an activity that is performed within a certain case (i.e. process instance) by a resource at a given timestamp. In our framework, these are treated as ordinary event characteristics. We assume characteristics such as *Activity*, *Case*, *Resource*, and *Timestamp* but do not treat them differently. No two events can have exactly the same values for all characteristics. This is not a limitation, one can always introduce an event identifier as an additional characteristic to ensure uniqueness.

Our framework aims to support so-called *analysis use cases*.

Definition 2 (*Analysis use case*). An *analysis use case* is a triple $A = (c_r, \mathcal{C}_d, F)$ consisting of

- a dependent characteristic $c_r \in \mathcal{C} \setminus \mathcal{C}_d$,
- a set $\mathcal{C}_d \subseteq \mathcal{C} \setminus \{c_r\}$ of independent characteristics,
- an event-selection filter $F \subseteq \mathcal{E}$, which selects the events that are retained for the analysis.

The definition of analysis use cases specializes standard definitions of supervised learning [7] for the process-mining case. The specialization consists in the fact that the instances that are used for learning are a subset of the events in an event log. As a consequence, the dependent and independent characteristics are linked to the process attributes that are recorded in the event log.

The result of performing an analysis use case is a decision or a regression tree that relates the *independent characteristics* (also called the predictor or explanatory variables) to the *dependent characteristic* (also called the response variable). F selects the events for which we would like to make a prediction. Sometimes we select all events, sometimes only the last one in each case, and at other times all events corresponding to a particular activity or resource. The events selected through F correspond to the instances used to construct the decision or regression tree. If we would like to predict the remaining processing time (the dependent characteristic), then we select all events and may use independent characteristics such as the resource that worked on it, the last activity, the current workload, and the type of customer.

[3] Given any set X, $\mathbb{B}(X)$ denotes the set of all possible multisets over X.

As discussed in Section 2.1, many analysis use cases require the presence of dependent and/or independent characteristics that are not readily available in the event log. Similarly, using business domain knowledge, an analyst may want to verify a reasonable hypothesis of the existence of a correlation to a given set of independent characteristics, which may not be explicitly available in the event log.

Step 2 in Fig. 1 aims to enrich the event log with valuable information required for the analysis use cases we are interested in. We provide a powerful framework to manipulate event logs and obtain a new event log that suits the specific analysis use case, e.g. events are enriched with additional characteristics.

Definition 3 (*Trace and log manipulation*). Let $\mathcal{T}$ be the universe of traces and let $L \in \mathbb{B}(\mathcal{T})$ be an event log. A *trace manipulation* is a function $\delta_L \in \mathcal{T} \to \mathcal{T}$ such that for any $t \in L$: $|t| = |\delta_L(t)|$.

Hence, given a trace $t \in L$, $\delta_L(t)$ yields a new trace of the same length. A trace manipulation function can only add event characteristics, without removing or adding events.

$\delta_L(t)$ not only depends on t but may also depend on other traces in L. For example, to compute a resource's workload, we need to look at all traces. $\delta_L(t)$ may even depend on any other contextual information (although this is not formalized here). For example, we may consider multiple processes or multiple organizations at the same time and even include weather information.

By applying δ_L to all traces in L, we obtain a new log $L' = [\delta_L(t) | t \in L]$. Recall that, technically, a log is multiset of traces.

After applying various log manipulations, we obtain an enriched event log where an event may have any number of event characteristics. Then we apply the desired analysis use case $(c_r, \mathcal{C}_d, F)$. Each selected event corresponds to an instance of the supervised learning problem where the dependent characteristic c_r is explained in terms of the independent characteristics $\mathcal{C}_d$. Many analysis use cases require to only consider a subset of all log's events. For example, if we aim at relating the total case duration to the number of times a particular resource was involved in it, we want to consider each case as a whole and, hence, select the last event of the enriched event log. In this case the number of instances used to learn a tree classifier is same as the number of cases in the event log. If we would like to analyze a decision in the process, then the number of instances should equal the number of times the decision is made. Hence, F should select the events corresponding to the decision we are interested in.

The resulting tree produced by Step 3 in Fig. 1 may provide valuable information regarding performance (e.g., time, costs, and quality) and conformance. However, as Fig. 1 shows it is also possible to use the analysis result to cluster traces and create smaller event logs. These so-called sublogs can be used for any type of process mining. For example, we may compare process models learned for the different classes of traces identified in the decision tree.

In addition, it can be given as input to split the traces of the event logs into groups in order to, e.g., construct partial process models that are easier to understand. The trace clustering can be defined as follows[4]:

Definition 4 (*Trace clustering*). Let $\mathcal{T}$ be the universe of traces. A *trace clustering based on an analysis use case* $A = (c_r, \mathcal{C}_d, F)$ is a function $\lambda_A \in \mathbb{B}(\mathcal{T}) \to \mathbb{B}(\mathbb{B}(\mathcal{T}))$ such that for any $L \in \mathbb{B}(\mathcal{T})$: $L \supseteq \biguplus_{L' \in \lambda_A(L)} L'$ and for any $L', L'' \in \lambda_A(L)$, if $t \in L'$ and $t \in L''$, then $L' = L''$.

Function λ_A associates a multiset of trace clusters to an event log, where each cluster is again a multiset of traces (i.e., a sublog). Trace clustering cannot generate new traces and a trace can only belong to one cluster.

Note that λ_A does not necessarily partition the traces in the log, e.g., traces can be considered as outliers and, hence, discarded from any of the clusters.

Step 4 in Fig. 1 provides the possibility to conduct *comparative process mining* [6], i.e., comparing different variants of the process, different groups of cases, or different organizational units.

In the remainder of this paper, we provide further information on how make the concepts discussed operational. Note that the approach described using Fig. 1 is very generic and also fully implement, as extensively discussed in Section 7.

2.3. *Use of tree-learning methods to perform analysis use cases*

The result of performing an analysis use case is a prediction tree: decision or regression tree. Decision and regression trees classify instances (in our case the events selected through F) by sorting them down in a tree from the root to some leaf node. Each non-leaf node specifies a test of some attribute (in our case, an independent characteristic) and each branch descending from that node corresponds to a range of possible values for this attribute. Each leaf node is associated with a value of a class attribute (in our case, the dependent characteristic). A path from root to a leaf represents a rule for classification or regression. There exist numerous algorithms to build a decision or a regression tree starting from a training set [7]. Every algorithm for constructing a prediction tree takes three input parameters: (1) a dependent variable (a.k.a. class variable) c_r, (2) a set of independent variables $\mathcal{C}_d$, and (3) a multi-set $\mathcal{I} \in \mathbb{B}(\mathcal{E})$ of instances for constructing the tree. The multi-set of instances $\mathcal{I}$ is constructed starting from an event log L. Given the multi-set E of all events in L, i.e. $E = \biguplus_{t \in L} \biguplus_{e \in t} e$, the multi-set of instances for training is computed as follows: $\mathcal{I} = E|_F$.[5] Multi-set $\mathcal{I}$ contains every instance that can be used to train and test the tree classifier; instances that have been filtered out are discarded. If cross-validation is employed, any instance is going to be used for both training and testing. Obviously, it is also possible to clearly split $\mathcal{I}$ in a train and test set. We do not prevent any possibility from being used, as, for a specific case, it may be worthwhile exploring both.

Our framework is agnostic with respect to specific algorithms used to construct prediction trees. However, for a concrete implementation, we opted for the C4.5 algorithm to construct decision tree [7] and the REPTree algorithm to construct regression tree [8]. Decision trees require the dependent variable to be nominal. If not, the domain of the class attribute needs to be discretized. Literature provides several ways to discretize dependent variables. While any discretization technique can be employed, our implementation provides two specific ones: *equal-width binning* and *equal-frequency binning* [9].

3. Extracting relevant process characteristics

This section focuses on Step 2 in Fig. 1 and provides an overview of different manipulations to enrich the event log such that the required dependent and independent variables are present for building the desired decision or regression tree.

Remember that formally *trace manipulation* is a function $\delta_L \in \mathcal{T} \to \mathcal{T}$ transforming one trace into another while possibly using the broader context shown in Fig. 2(b). The trace manipulations described in this section always add one new event characteristic to each case in the event log. Recall that for any $t \in L$: $|t| = |\delta_L(t)|$, i.e., no events are added or removed.

The remainder of this section focuses on showing examples of manipulation for the different perspectives of Fig. 2(a). The lists of manipulations are far from being complete since the number of potential manipulations is infinite. We limit ourself to discuss the manipulations that are available in implementation at the time of writing. However, the framework is generic: one can easily plug-in new manipulations. In fact, also the implementation is such that one can easily encode new manipulations, as it will be discussed in Section 7.

In the remainder of Section 3, we also provide formal definitions for some of the provided examples of the manipulations. For this aim, we need to introduce some helper functions and operators that are going to be used in the definitions.

Helper functions and operators: As mentioned in Section 2.2, each event e is considered as unique. Therefore, given an event e, we can define functions *prefix*(e) and *postfix*$(e) \in \mathcal{E}^*$ that return the sequence of events that respectively precede and follow event e in the same trace. Events are aggregations of process characteristics. Given an event e, $e(c)$ returns the value of characteristic c for event e if it exists. If no value exists, the special value $\perp \notin \mathcal{U}$ is returned. Given an event e and a process characteristic as pair $(c, u) \in \mathcal{C} \times (\mathcal{U} \cup \{\perp\})$, we introduce the overriding operator $e' = e \odot (c, u) \in \mathcal{E}$ such that $e'(c) = u$ and, for each $c' \in \mathcal{C} \backslash \{c\}$, $e'(c) = e(c)$. We also introduce the trace concatenation operator $\oplus$ as follows: given two traces $t' = \langle e'_1, \ldots, e'_n \rangle$ and $t'' = \langle e''_1, \ldots, e''_m \rangle$, $t' \oplus t'' = \langle e'_1, \ldots, e'_n, e''_1, \ldots, e''_m \rangle$. Given a trace $t \in \mathcal{E}^*$, a characteristic $c \in \mathcal{C}$ and a set of values $\overline{\mathcal{U}} \subseteq \mathcal{U}$, function *firstOcc*$(t, c, \overline{\mathcal{U}})$ returns the first occurrence of any value $u \in \overline{\mathcal{U}}$ in trace t for

[4] The union operator $\uplus$ is defined over two multiset sets. Given 2 multi-sets A and B, $A \uplus B$ is a multiset that contains all elements of A and B; the cardinality of each element $e \in A \uplus B$ is the sum of the cardinalities of e in A and B.

[5] Operator $|_F$ is the defined as follows: given a multi-set E and a set F, $E|_F$ contains all elements that occur in both E and F and the cardinality of each element $e \in E|_F$ is equal to the cardinality of e in E.

characteristic c. It can be recursively defined as follows:

$$firstOcc(\langle\rangle, c, \overline{\mathcal{U}}) = \perp$$

$$firstOcc(\langle e\rangle \oplus t', c, \overline{\mathcal{U}}) = \begin{cases} e(c) & \text{if } e(c) \in \overline{\mathcal{U}} \\ firstOcc(t', c, \overline{\mathcal{U}}) & \text{if } e(c) \notin \overline{\mathcal{U}} \end{cases}$$

Similarly, one can define the last occurrence: $last\ Occ(\langle e_1, \ldots, e_n\rangle, c, \overline{\mathcal{U}}) = firstOcc(\langle e_n, \ldots, e_1\rangle, c, \overline{\mathcal{U}})$

3.1. Control-flow perspective

The control-flow perspective refers to the occurrence and ordering of activities.

Trace Manipulation *CFP1$_x$*: **Number of Executions of Activity** *x* **Until the Current Event.** This operation counts the number of times x was executed until the current event in the trace. This event characteristic is added to all traces. Formally, given a trace $t = \langle e_1, \ldots, e_n\rangle \in \mathcal{E}^*$, $CFP1_x(t) = \langle e'_1, \ldots, e'_n\rangle \in \mathcal{E}^*$ where, for $1 \le i \le n$, $e'_i = e_i \odot (CFP1, n)$ with $n = |\{e'' \in prefix(e) | e''(Activity) = e(Activity)\}|$.

Trace Manipulation *CFP2$_X$*: **First Occurrence of Any Activity in Set** *A* **After Current Event.** This operation augments every event e with the name of the activity that belongs to set X and firstly occurs in the trace after e. If there is no next event or no activity in X that occurs in the trace after e, special value $\perp$ is used. Set X can be technically empty but it would be meaningless: any event would be always augmented with the $\perp$ value. If set X contains all activities to which log's events refer, the event is augmented with the name of the activity that follows e in the trace, or $\perp$ if such an event is the last in the trace. Formally, given a trace $t = \langle e_1, \ldots, e_n\rangle \in \mathcal{E}^*$, $CFP2_x(t) = \langle e'_1, \ldots, e'_n\rangle \in \mathcal{E}^*$ where, for $1 \le i \le n$, $e'_i = e_i \odot (CFP2, u)$ with $u = firstOcc(postfix(e), Activity, X)$

Trace Manipulation *CFP3*: **Previous Activity in the Trace.** This operation determines the activity directly preceding the current event in the same case (i.e. the same log trace). If there is no previous event, a $\perp$ value is used. Formally, given a trace $t = \langle e_1, \ldots, e_n\rangle \in \mathcal{E}^*$, $CFP3(t) = \langle e'_1, \ldots, e'_n\rangle \in \mathcal{E}^*$ where $e'_1 = e_1 \odot (CFP3, \perp)$ and, for $1 < i \le n$, $e'_i = e_i \odot (CFP3, e_{i-1}(Activity))$.

Trace Manipulation *CFP4*: **Current Activity.** This operation determines the current event's activity name. Strictly speaking no manipulation is needed since this is already an event attribute. Nevertheless, we list it to be able to refer to it later.

One can think of many other control-flow trace manipulation, e.g., counting the number of times a given activity is followed by another activity.

3.2. Data-flow perspective

Next we consider the data-flow perspective. Here we focus on the data attributes attached to events, these may refer to the event itself or to properties of the case the event belongs to. Let us recall that $\mathcal{U}$ indicates that universe of possible values which a characteristic can take on.

Trace Manipulation *DFP1$_c$*: **Latest Recorded Value of Characteristic** *c* **Before Current Event.** This operation determines the latest value of some attribute c not considering the current event. Note that, when enriching a certain event e, there may be multiple events before e in the same trace that assign a value to c. Then, we consider the latest of these events. If the attribute was not present in any of the preceding events, a $\perp$ value is returned. As before all events are preserved in the trace; this characteristic is added to each event. Formally, given a trace $t = \langle e_1, \ldots, e_n\rangle \in \mathcal{E}^*$, $DFP1_c(t) = \langle e'_1, \ldots, e'_n\rangle \in \mathcal{E}^*$ where, for $1 \le i \le n$, $e'_i = e_i \odot (DFP1c, u)$ with $u = lastOcc(prefix(e), c, \mathcal{U})$.

Trace Manipulation *DFP2$_c$*: **Latest Recorded Value of Characteristic** *c* **Until Current Event.** This operation determines the latest value of c also considering the current event. If the current event has attribute c, then the corresponding value is the latest value. If not, the previous events for the same case are considered. Formally, given a trace $t = \langle e_1, \ldots, e_n\rangle \in \mathcal{E}^*$, $DFP2c(t) = \langle e'_1, \ldots, e'_n\rangle \in \mathcal{E}^*$ where, for $1 \le i \le n$, $e'_i = e_i \odot (DFP2c, u)$ with $u = lastOcc(prefix(e) \oplus \langle e\rangle, c, \mathcal{U})$.

Trace Manipulation *DFP3$_c$*: **Average Value of Characteristic** *c* **Until the Current Event.** This operation augments an event e with the average of the values assigned to characteristic c by e and by those events that are preceding e in the trace. Formally, given a trace $t = \langle e_1, \ldots, e_n\rangle \in \mathcal{E}^*$, $DFP3_c(t) = \langle e'_1, \ldots, e'_n\rangle \in \mathcal{E}^*$ where, for $1 \le i \le n$, $e'_i = e_i \odot (DFP3c, u)$ with $u = \text{avg}_{e'' \in prefix(e) \oplus \langle e\rangle} e(c)$.[6]

Trace Manipulation *DFP4$_c$*: **Maximum Value of Characteristic** *c* **Until the Current Event.** This operation augments an event e with the maximum value assigned to characteristic c by e and by those events that are preceding e in the trace. Formally, given a trace $t = \langle e_1, \ldots, e_n\rangle \in \mathcal{E}^*$, $DFP4_c(t) = \langle e'_1, \ldots, e'_n\rangle \in \mathcal{E}^*$ where, for $1 \le i \le n$, $e'_i = e_i \odot (DFP4c, u)$ with $u = \max_{e'' \in prefix(e) \oplus \langle e\rangle} e(c)$ (footnote 6).

Trace Manipulation *DFP5$_c$*: **Minimum Value of Characteristic** *c* **Until the Current Event.** This operation augments an event e with the minimum value assigned to characteristic c by e and by those events that are preceding e in the trace. Formally, given a trace $t = \langle e_1, \ldots, e_n\rangle \in \mathcal{E}^*$, $DFP5_c(t) = \langle e'_1, \ldots, e'_n\rangle \in \mathcal{E}^*$ where, for $1 \le i \le n$, $e'_i = e_i \odot (DFP5c, u)$ with $u = \min_{e'' \in prefix(e) \oplus \langle e\rangle} e(c)$ (footnote 6).

Trace Manipulation *DFP6$_c$*: **Sum Of Value of Characteristic** *c* **Until the Current Event.** This operation augments an event e with the sum of the values assigned to characteristic c by e and by those events that are preceding e in the trace. Formally, given a trace $t = \langle e_1, \ldots, e_n\rangle \in \mathcal{E}^*$, $DFP6_c(t) = \langle e'_1, \ldots, e'_n\rangle \in \mathcal{E}^*$ where, for $1 \le i \le n$, $e'_i = e_i \odot (DFP6c, u)$ with $u = \sum_{e'' \in prefix(e) \oplus \langle e\rangle} e(c)$ (footnote 6).

Note that, of course, manipulations $DFP3_c, \ldots, DFP6_c$ are only applicable if c is numerical, such as a timestamp, a cost or the customer's age.

[6] To keep the notation simple, we assume here that functions avg, min and max and the sum operator $\sum$ automatically exclude all occurrences of value $\perp$. Also, when the functions and the operator are applied over an empty set, they return $\perp$.

3.3. Resource and organization perspective

The resource and organization perspective focusses on the resource executing the event or corresponding activity instance. Resources may be grouped in different organizational entities (groups, roles, positions, departments, etc.). For the manipulations listed below we do not provide a formal introduction because it would not add much to the textual description, unless the workload calculations are explained in detail, which is out of the scope of this paper. To compute the total workload or that of a single resource, we employ the approach discussed in [2].

Trace Manipulation *ROP1*: **Workload per Resource.** Events may be related to resources. By scanning the event log one can get an indication of the amount of work performed in a particular period. Moreover, one can also see waiting work items that are later executed by a particular resource. The resource's workload at the time the event occurs can be attached to the event.

Trace Manipulation *ROP2*: **Total Workload.** The previous event characteristic is related to the particular resource performing the event. However, we can also take the sum over all running or queuing work items. For example, we can simply count the number of activities in-between a start and complete event.

Trace Manipulation *ROP3*: **Current Resource.** This operation determines the current event's resource (if any). Strictly speaking no manipulation is needed, because it is already part of the event. However, we list it to be able to refer to it later.

Trace Manipulation *ROP4*: **Current Role.** This operation determines the role of the current event. This can be already part of the event; in that case, no manipulation is needed. If not, it can be derived by cross-correlating organizational information, stored, e.g., in a database, and the current resource.

Many other resource-related event characteristics are possible: many ways exist to measure workload, utilization, queue lengths, response times, and other performance-related properties. Information may be derived from a single process or multiple processes.

3.4. Time perspective

The time perspective refers to various types of durations (service times, flow times, waiting times, etc.) in processes. Clearly the resource/organization perspective and the time perspective are closely related.

Some of the manipulations for the resource/organization perspective have already exploited *transactional information* in the event log. For example, to measure the number of running activity instances, we need to correlate the start events and the complete events. Schedule, start, complete, suspend, resume, etc. are examples of possible *transaction types* [1]. Each event has such a transaction type (default is *complete*). An *activity instance* corresponds to a set of events. For one activity instance there may be a *schedule* event, a *start* event, a *suspend* event, a *resume* event, and finally a *complete* event. The *duration* of an activity instance is the time between the *start* event and the *complete* event.

Trace Manipulation *TIP1*: **Activity Duration.** The characteristic adds information on activity instance durations to complete events. We can measure the duration of an activity instance by subtracting the timestamp of the start event from that of the complete event, under the assumption that never more than one instance of the given activity is concurrently being executed.[7] Formally, $t = \langle e_1, \ldots, e_n \rangle \in \mathcal{E}^*$, $TIP1(t) = \langle e'_1, \ldots, e'_n \rangle \in \mathcal{E}^*$ where, for $1 \le i \le n$:

- if $e_i(Transition) \neq$ "*Complete*", then $e'_i = e_i$;
- if $e_i(Transition) =$ "*Complete*", then $e'_i = e_i \odot (TIP1, e_i(Timestamp) - e_j(Timestamp))$ such that $j < i$ and $e_j(Activity) = e_i(Activity)$ and, for all $j < k < i$, $e_k(Activity) \neq e_i(Activity)$.

Note that start or complete events may be missing and that correlation may be non-trivial [1]. Hence, there are multiple ways to measure activity durations in these cases. We support the approach in [2] where the start time is approximated as follows: it either the time of completion of the previous activity within the same process instance or the time of completion of the previous activity by the same resource (possibly in a different process instance) depending on which one comes latest. This is based on the assumption that the resource could have started to work on the activity if the previous activity completed and the resource was not working on other activities. The full formalization is beyond the scope of this paper.

Trace Manipulation *TIP2*: **Time Elapsed since the Start of the Case.** Each non-empty case has a first event. The timestamp of this event marks the start of the case. Events corresponding to the same case also have a timestamp. The difference between both corresponds to the elapsed since the start of the case. This information is added to all events. Formally, given a trace $t = \langle e_1, \ldots, e_n \rangle \in \mathcal{E}^*$, $TIP2(t) = \langle e'_1, \ldots, e'_n \rangle \in \mathcal{E}^*$ where, for $1 \le i \le n$, $e'_i = e_i \odot (TIP2, e_i(Timestamp) - e_1(Timestamp))$.

Trace Manipulation *TIP3*: **Remaining Time Until the End of Case.** Each non-empty case has a last event. The timestamp of this event marks the end of the case. Events corresponding to the same case also have a timestamp. The difference between both corresponds to the remaining flow time. This information is added to all events. Formally, given a trace $t = \langle e_1, \ldots, e_n \rangle \in \mathcal{E}^*$, $TIP3(t) = \langle e'_1, \ldots, e'_n \rangle \in \mathcal{E}^*$ where, for $1 \le i \le n$, $e'_i = e_i \odot (TIP3, e_n(Timestamp)) - e_i(Timestamp))$.

Trace Manipulation *TIP4*: **Case duration.** This event characteristic measures the difference between the last event and first event of the case the current event belongs to. Formally, given a trace $t = \langle e_1, \ldots, e_n \rangle \in \mathcal{E}^*$, $TIP4(t) = \langle e'_1, \ldots, e'_n \rangle \in \mathcal{E}^*$ where, for $1 \le i \le n$, $e'_i = e_i \odot (TIP4, e_n(Timestamp)) - e_1(Timestamp))$.

Trace Manipulation *TIP5*: **Current Timestamp.** This information corresponds to the event's timestamp. Strictly

[7] Experience shows this assumption holds in the large majority of processes due to the fact that information systems rarely support the possibility of concurrently starting multiple instances of the same activity within the same process instance.

speaking no manipulation is needed since this is already an event attribute. Nevertheless, we list it to be able to refer to it later.

Other time-related event characteristics are also possible. For example, if *schedule* events are present, one can measure waiting times and attach this information to events.

3.5. Conformance perspective

The time perspective is important for performance-related questions. However, process mining also includes conformance-related questions in the context of compliance management, auditing, security, etc. To be able to answer conformance-related questions we need to have a specification of the desired behavior. This may be a complete process model or just a set of simple rules. Here we consider two forms of specified normative behavior: (1) a process model (typically a Petri net) and (2) a constraint expressed in terms of LTL (Linear Temporal Logic). The LTL-based checking is based on the technique described in [10]. The others rely on the ProM implementation of the techniques discussed in [11], which are concerned with finding an alignment of traces in the log with the control flow of process models. An extension of the technique is provided in [12] where the other perspectives are also considered. Note that these are just examples, any evaluation of the observed behavior with respect to the normative behavior can be used (including declarative languages like Declare).

To understand such trace manipulations, it is important to understand the seminal notion of *alignments* [11]. To establish an alignment between process model and event log we need to relate "moves" in the log to "moves" in the model. However, it may be the case that some of the moves in the log cannot be mimicked by the model and vice versa. Basically, there are three kinds of moves after alignment:

- $(e, \gg)$ is a "move on log", i.e., in reality an event e occurred that could not be related to a move of the model,
- $(\gg, a)$ is a "move on model", i.e., in the process model activity a needs to be executed, but in reality there was no related event that actually occurred, and
- (e,a) is a "synchronous move" if in reality an event e occurred that corresponds to an activity a executed in the model and vice versa.

In an optimal alignment the number of synchronous moves is maximized and the number of moves on log and moves on model is minimized. The optimal alignment of a trace wrt a given model is also used to compute the *fitness* of a trace. Fitness is measured as a value between 0 and 1; value 1 indicates perfect fitness, which means that the alignment only contains synchronous moves (thus, no deviations). As the percentage of moves on log and model over all moves increases, the fitness value decreases. Value 0 indicates a very poor fitness. Interested readers are referred to [12,13].

Trace Manipulation $COP1_P$: **Trace Fitness.** Given a process model P, it augments each event with the value of fitness of the trace to which the event belongs

Trace Manipulation $COP2_{P,a}$: **Number of Not Allowed Executions of Activity** a **Thus Far.** Using process model P an optimal alignment is created per trace. This trace manipulation augments each event e with an integer characteristic that denotes the number of "moves on log". It counts the number of times something happened in reality before e in the corresponding trace that was not possible or allowed according to the process model P [11].

Trace Manipulation $COP3_{P,a}$: **Number of Missing Executions of Activity** a **Thus Far.** This trace manipulation augments each event with the number of "moves on model". It counts the number of times something had to happen in model P before e in the corresponding trace although it did not happen in reality [11].

Trace Manipulation $COP4_{P,a}$: **Number of Correct Executions of Activity** a **Thus Far.** This trace manipulation augments each event the number of "synchronous moves", i.e., the number of times that, before e in the corresponding trace, an event actually occurred when it was expected according to the model P [11].

Trace Manipulation $COP5_F$: **Satisfaction of Formula** F **Considering the Whole Trace.** This trace manipulation augments each event with an additional boolean characteristic and assigns a true value if and only if formula F is satisfied. Up to this date, formula F can be defined through LTL and checked using the LTL checker discussed in [10].

3.6. Trace manipulation: a concrete example

The trace manipulations described thus far have all been implemented in ProM (see the *FeaturePrediction* package). Note that the set of manipulations is not fixed and can be extended. Also note that any context information can be used to enrich the event log with further information, i.e., it is not limited to the perspectives in Fig. 2(a) and may include truly external things like weather or traffic information.

Table 1 illustrates the application of two trace manipulation functions to a fragment of an event log. As a result, two event characteristics have been added to each event; characteristic *NextActivityInTrace* is nominal whereas *ElapsedTime* is numerical. This shows that the log manipulations are applied to create both nominal and numerical characteristics. Let us denote the set of all activity names seen in the event log with X_L, i.e. $X_L =$ {*Preoperative Screening, Laparoscopic Gastrectomy, Nursing, First Hospital Admission*}. Trace manipulation *First Occurrence of Any Activity in Set* X_L After Current Event ($CFP2_{X_L}$) was used to add the next activity to each event. Note that the last event of each case has value $\perp$ because there is no next activity. Trace manipulation *Time Elapsed since the Start of the Case (TIP2)* adds durations. Note that the last event of each case now has a characteristic capturing the case's flow time.

4. Event selection filter

As described in Definition 2 an analysis use case is a triplet $(c_r, \mathcal{C}_d, F)$. The trace manipulations discussed in

Table 1
Fragment of a hospital's event log with four traces. Let us denote the set of all activity names seen in the event log with X_L. The gray columns have been added after applying two trace manipulations: *First Occurrence of Any Activity in Set* X_L *After Current Event* ($CFP2_{A_{ex}}$) and *Time Elapsed since the Start of the Case* (*TIP2*). `NextActivityInTrace` and `ElapsedTime` are the names of the characteristics that are added as a result of these manipulations.

Case	Timestamp	Activity	Resource	Cost	NextActivityInTrace	ElapsedTime
1	1-12-2011:11.00	Preoperative Screening	Giuseppe	350	Laparoscopic Gastrectomy	0 days
1	2-12-2011:15.00	Laparoscopic Gastrectomy	Simon	500	Nursing	1.16 days
1	2-12-2011:16.00	Nursing	Clare	250	Laparoscopic Gastrectomy	1.20 days
1	3-12-2011:13.00	Laparoscopic Gastrectomy	Paul	500	Nursing	2.08 days
1	3-12-2011:15.00	Nursing	Andrew	250	First Hospital Admission	2.16 days
1	4-12-2011:9.00	First Hospital Admission	Victor	90	$\perp$	3.92 days
2	7-12-2011:10.00	First Hospital Admission	Jane	90	Laparoscopic Gastrectomy	0 days
2	8-12-2011:13.00	Laparoscopic Gastrectomy	Giulia	500	Nursing	1.08 days
2	9-12-2011:16.00	Nursing	Paul	250	$\perp$	2.16
3	6-12-2011:14.00	First Hospital Admission	Gianluca	90	Preoperative Screening	0 days
3	8-12-2011:13.00	Preoperative Screening	Robert	350	Preoperative Screening	1.96 days
3	10-12-2011:16.00	Preoperative Screening	Giuseppe	350	Laparoscopic Gastrectomy	4.08 days
3	13-12-2011:11.00	Laparoscopic Gastrectomy	Simon	500	First Hospital Admission	6.88 days
3	13-12-2011:16.00	First Hospital Admission	Jane	90	$\perp$	7.02 days
4	7-12-2011:15.00	First Hospital Admission	Carol	90	Preoperative Screening	0 days
4	9-12-2011:7.00	Preoperative Screening	Susanne	350	Laparoscopic Gastrectomy	0.66 days
4	13-12-2011:11.00	Laparoscopic Gastrectomy	Simon	500	Nursing	5.84 days
4	13-12-2011:13.00	Nursing	Clare	250	Nursing	5.92 days
4	13-12-2011:19.00	Nursing	Vivianne	250	$\perp$	6.16 days

Section 3 can be used to create the dependent characteristic c_r and the independent characteristics $\mathcal{C}_d$. In this section we elaborate on the choices for the event-selection filter $F \subseteq \mathcal{E}$ that selects the events that are retained for the analysis. Recall that each of the remaining events corresponds to an instance of the learning problem that aims to explain c_r in terms of $\mathcal{C}_d$.

Recall that we assume that events are uniquely identifiable, i.e., no two events have identical attributes. There are two extremes: we can retain all events or just one event per case. However, we can also retain selected events, e.g., all complete events.

Event Filter *EF1*: **Keep All Events.** No events are removed and all are kept. Formally, $EF1 = \mathcal{E}$.

Event Filter *EF2*: **Keep First Event Only.** Per case only the first event is retained. Given a trace $t\langle e_1, e_2, \ldots e_n\rangle$, only e_1 is kept and the rest is removed. Formally, $EF2 = \{e \in \mathcal{E} : prefix(e) = \langle\rangle\}$.

Event Filter *EF3*: **Keep Last Event Only.** Per case only the last event is retained. Given a trace $t\langle e_1, e_2, \ldots e_n\rangle$, only e_n is kept and the rest is removed. Formally, $EF3 = \{e \in \mathcal{E} : postfix(e) = \langle\rangle\}$.

Event Filter *EF4*: **Keep All Complete Events.** Only events that have as transaction type *complete* are retained. All other events (e.g., start events) are removed. Formally, $EF4 = \{e \in \mathcal{E} : e(Transition) = \text{“}Complete\text{”}\}$.

Event Filter *EF5*: **Keep All Start Events.** Only events that have as transaction type *start* are retained. Formally, $EF5 = \{e \in \mathcal{E} : e(Transition) = \text{″}Start\text{″}\}$.

Event Filter $EF6_a$: **Keep All Events Corresponding to Activity** *a*. Only events that correspond to activity *a* are retained. Formally, $EF6_a = \{e \in \mathcal{E} : e(Activity) = a\}$.

Event Filter $EF7_r$: **Keep All Events Corresponding to Resource** *r*. Only events that were performed by resource *r* are retained. Formally, $EF7_r = \{e \in \mathcal{E} : e(Resource) = r\}$.

Event Filter $EF8_F$: **Keep All Events Satisfying a given formula** *F*. *F* is a user-provided boolean expression defined over the set $\mathcal{C}$ of names of process characteristics. *F* can make use of relational operators $<, >, =$ and logical operators, namely conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$). This filter retains all events for which *F* evaluate to true. This generalizes filters $EF4, \ldots, EF7$. For instance, filter $EF4 = EF8_{Transition = \text{″}Complete\text{″}}$. Filter *EF1* is also a specialization: $EF1 = EF8_{\text{true}}$

It is also possible to combine a set of filters $\overline{F}_1, \ldots, \overline{F}_n$ where, for all $1 \le i \le n$, $\overline{F}_i \subseteq \mathcal{E}$. The combined filter $\widehat{F}$ is equal to $\bigcap_{i \in \mathbb{N}: 1 \le i \le n} \overline{F}_i$. For instance, if one wants to keep all *complete* events corresponding to activity *a*, the filter is $EF4 \cap EF6_a$.

The event-selection filter *F* selects the instances used to learn the desired tree. Some analysis use cases require to have one instance per case. Table 2 shows the application of event filter *Keep Last Event Only (EF3)* to Table 1.

5. Overview of process-related questions that can be answered

Literature proposes several research works to correlate specific process characteristics to each other. This section aims to illustrate that those works propose ad hoc solutions for a more general problem: finding any type of correlation among arbitrary process characteristics at any level (event, case, process, resource, etc.). Our framework attempts to solve the more general problem. In this way, we can certainly perform the same correlation analyses as those research works. However, by applying the appropriate trace manipulations and event-filter functions described earlier and by choosing the correct dependent and independent characteristics, we can provide an answer to many more correlation analyses.

Tables 3 and 4 illustrate concrete analysis use cases that allows for performing the same correlation analyses as in many earlier research works. This illustrates that it is possible to unify existing approaches. For the sake of simplifying the definition of the analysis use cases, we assume that all trace manipulations described in Section 3

Table 2
The retained events after applying event filter *Keep Last Event Only* (*EF*3) to the event log shown in Table 1.

Case	Timestamp	Activity	Resource	Cost	NextActivityInTrace	ElapsedTime
1	4-12-2011:9.00	Case	Victor	90	$\perp$	3.92 days
2	9-12-2011:16.00	Case	Paul	250	$\perp$	2.16 days
3	13-12-2011:16.00	Case	Jane	90	$\perp$	7.02 days
4	13-12-2011:19.00	Case	Vivianne	250	$\perp$	6.16 days

Table 3
Three analysis use cases that illustrate how our framework works to unify existing approaches to correlate specific process characteristics.

#1. **Analysis Use Case** $(COP5_F, \{CFP4, ROP3, \forall_{c \in \mathcal{C}_{Avail}} DFP1_c\}, EF1)$: **Run-time predictions of violations of formula** *F*. The aim of this use case is to predict, given the current status of the process instances, the next activities to work on to maximize the chances of achieving a given business goal expressed using some LTL formula *F*. The selected dependent characteristic is $COP5_F$ ("Satisfaction of Formula *F* Considering the Prefix Trace Until Current Event *e*"), the selected independent characteristics are *CFP4* ("Current Activity"), *ROP3* ("Current Resource"), and $DFP1_c$ ("Latest Recorded Value of Characteristic *c* Before Current Event") for all recorded characteristics *c*. The selected event-selection filter is *EF1* ("Keep All Events"). In [14], an ad hoc solution is proposed for this problem where formulas are expressed in LTL

#2. **Analysis Use Case** $(DFP2_{\texttt{Outcome}}, \{\forall_{c \in \mathcal{C}_{Avail} \setminus \{\texttt{Outcome}\}} DFP2_c\}, EF3)$: **Prediction of the outcomes of the executions of process instances, which is stored as characteristic `Outcome`**. The goal of this use case is to predict the outcome of a case. Predictions are computed using a set of complete process instances that are recorded in the event log. The last event of each trace is associated with a characteristic `Outcome`, which may be numerical or nominal (including boolean) depending on the specific setting. The prediction is done at the case level: one instance for learning is created for each trace in the event log. The dependent characteristic is $DFP2_{\texttt{Outcome}}$ ("Latest Recorded Value of Characteristic `Outcome` After Current Event") for a specific `Outcome` or interest. The selected independent characteristics are $DFP2_c$ for all event characteristics *c* except `Outcome`. The chosen event-selection filter is *EF3* ("Keep Last Event Only"). In [15], an ad hoc solution is proposed for this problem. In fact this analysis use case is very close to traditional data mining where the case is considered as a whole rather than rooming in on the individual events and behavior. Conforti et al. also proposes an ad hoc solution to predict whether or not an process instance is going to complete with a fault. If completed with a fault, its magnitude is also predicted [16]. In this case, the outcome is a numerical value that ranges between 0 and 1, whereas 0 indicates no fault and 1 the highest level of fault

#3. **Analysis Use Case** $(CFPA_X, \{\forall_{c \in \mathcal{C}_{Avail}} DFP2_c\}, EF4 \cap EF6_{\overline{x}})$: **Mining of conditions at a decision point that determine the activity to execute within a set $\mathbf{X} \subset \mathbf{X_L}$ after execution of an activity $\overline{\mathbf{x}} \in \mathbf{X_L}$**. This analysis use case aims to determine the conditions that enable the execution of one of multiple activities when the execution flow reaches a decision point. Using the BPMN notation, a decision point is represented through the *Exclusive Choice* construct. An example of decision point analysis is shown in Fig. 3 using the BPMN notation, where we aim to discover which conditions enable *a*, *b* or *c* for execution. For the analysis use case, we retain every event of type *complete* (i.e. filter *EF4* is applied) that is referring to executions of activity $\overline{x}$ (i.e. $EF6_{\overline{x}}$). For our example, $\overline{x} = z$, which is the activity before the decision point. These conditions are defined over the process variables, which are the characteristics $\mathcal{C}_{Avail}$ present in the original event log. Therefore, characteristic $DFP2_c$ ("Latest Recorded Value of Characteristic *c* Until Current Event") is used as an independent characteristic for all $c \in \mathcal{C}_{Avail}$. The dependent characteristic is $CFPA_X$ ("First Occurrence of Any Activity in Set *X* After Current Event") where *X* is the set of activities that are at the decision point; for our example, $X = \{a, b, c\}$. In [17], an ad hoc solution is proposed for this problem

are applied. Of course, in practice, one only needs to apply the sub-set of log manipulations that generate characteristics that are used in the definition of the analysis use case.

Let $\mathcal{C}_{Avail}$ be the set of characteristics available in the original event log *L*, which are not obtained through trace manipulations. Every event of a log always contains the activity name; hence, the *Activity* characteristic is always in $\mathcal{C}_{Avail}$. Let X_L denote the set of all values observed for the *Activity* characteristic in an event log *L*.

6. Trace clustering based on analysis use cases

As already discussed, the performance of an analysis use case produces a decision or a regression tree. These trees can be used to cluster event logs. Our clustering algorithm requires that each trace is associated with one single event; for instance, this is guaranteed by the event filters EF_2 and EF_3 in Section 4. As discussed in Section 2.3, each event becomes a training/test instance for learning decision or regression trees. Therefore, each trace is linked to one single instance that is exactly associated with one leaf of the tree. Our basic idea for clustering is that all log traces associated with the same leaf are grouped within the same cluster; therefore, there are as many clusters as the number of leaves in the decision tree.

For the sake of explanation, let us consider the second analysis use case in Table 3: "Prediction of the outcomes of the executions of process instances" and assume variable *Outcome* is numerical. The tree that results from this analysis use case can be used to cluster traces according to the outcome's value. Therefore, each cluster groups traces that record executions of process instances with similar outcomes. However, traces with similar outcomes may still be in different clusters if they have with different values for other process characteristics. This happens when the tree contains two different leaves with similar values for the Outcome variable.

The significant intervals of values of process characteristics that determine the cluster to which to add a given instance are not known in advance; conversely, they are computed as a result of the construction of the tree. If they were known in advance, one can easily define clusters by adding traces to cluster according to the values of such a

Table 4
Additional analysis use cases that further illustrate how our framework research works unify existing approaches to correlate specific process characteristics.

#**4. Analysis Use Case** ($ROP3, \{\forall_{c \in \mathcal{C}_{Avail}} DFP1_c, \ldots\}, EF6_x$): **Prediction of the executor of a certain activity** *x*. The goal is to discover the conditions that determine which resource is going to work on a given activity *x* during the process execution. The dependent characteristic is *ROP3* ("Current Resource"). The set of independent characteristics contain a characteristic $DFP2_c$ ("Latest Recorded Value of Characteristic *c* Before Current Event") for each $c \in \mathcal{C}_{Avail}$, at least. However, potentially any characteristic obtained through trace manipulation can be used to predict the executor; this explains the presence of "…" in the definition of the set of independent characteristics. Since the analysis use case is restricted to a single activity, the selected event-selection filter is $EF6_x$ ("Keep All Events Corresponding to Activity *x*")

#**5. Analysis Use Case** ($TIP3, \{\forall_{x \in X_L} CFP1_x, \forall_{c \in \mathcal{C}_{Avail}} DFP2_c, \ldots\}, EF1$): **Prediction of the remaining time to the end of the process instance**. The goal is, given a running process instance, to predict how long it would take for that instance to conclude. All the events are used to train the tree-based classifier and, hence, the event filter is *EF1* ("Keep all events"). The analysis use case aims to predict the remaining time; hence, the dependent characteristic is *TIP3* ("Remaining Time Until the End of Case"). As far as concerning the independent characteristics, any characteristics that can be obtained through trace manipulation may be suitable; this explains the presence of "…" in the definition of the set of independent characteristics. Certainly, the following characteristics should be included: $DFP2_c$ ("Latest Recorded Value of Characteristic *c* Until Current Event") for each $c \in \mathcal{C}_{Avail}$ as well as $CFP1_x$ ("The number of executions of activity *x* Until the Current Event") for each activity *x* observed in the event log (i.e. for all $x \in X_L$). It is similar to [18] when a multi-set abstraction is used; additionally, the current values of process variables are also taken into account

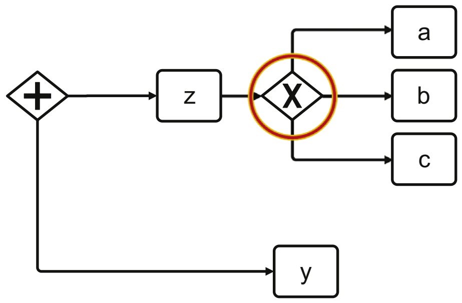


Fig. 3. Example of decision point analysis. Our framework can find the conditions that determine which activity is enabled at a decision point. In this figure, the decision point is highlighted through a circle: the analysis aims to discover the disjoined conditions that enable the three activities: *a*, *b* and *c*.

variable. However, even in the case, traces with similar values for the dependent characteristic but very different values for other relevant characteristics would be grouped in the same cluster, which, hence, would still contain traces with heterogenous behavior.

When using any classifier, it is extremely common to have outliers, which in our framework translates to having traces to be filtered out and included in no cluster.

According to our clustering criteria, one trace can belong to one cluster only. In the following, we discuss whether the trace should be retained in its own cluster or filtered out.

This depends on whether the result of performing an analysis use case is a decision or regression tree. Let $\mathcal{C}$ and $\mathcal{U}$ be the universe of characteristics and the universe of possible values, respectively; also $\mathcal{E} = \mathcal{C} \nrightarrow \mathcal{U}$ is the universe of events.

Decision Tree: Let us suppose to have a decision tree for a dependent characteristic $c \in \mathcal{C}$. This decision tree is learned from a set $\mathcal{I} \in \mathbb{B}(\mathcal{E})$ of instances, each of which is associated with one different trace. Let us suppose to have a log trace that corresponds to an instance $\mathbf{i} \in \mathcal{I}$. According to the decision tree, this instance is classified as belonging to a tree leaf *l* with value *v* for characteristic *c*. Trace *t* is retained in the cluster for *l* if $\mathbf{i}(c) = v$, i.e. it is correctly classified.

Regression Tree: Let us suppose to have a regression tree *tr* for a dependent characteristic *c*. This decision tree is learned from a multi-set $\mathcal{I}$ of instances. Let us suppose to have a log trace *t* that corresponds to an instance $\mathbf{i} \in \mathcal{I}$. According to regression tree *tr*, $\mathbf{i}(c)$ should be $v^{exp}_{tr,i,c} \in \mathbb{R}$. However, differences can occur; therefore, we introduce the mean absolute percentage error (MAPE) of instance $\mathbf{i}$ when predicting through regression tree *tr* as follows:

$$MAPE_{tr,i,c}(\mathbf{i}, c) = 100\% \left| \frac{\mathbf{i}(c) - v^{exp}_{tr,i,c}}{\mathbf{i}(c)} \right| \quad (1)$$

The trace associated with an instance $\mathbf{i}$ is retained if $MAPE_{tr,i,c}(\mathbf{i}, c) < X\%$ where *X* is a cut-off threshold that is defined by the process analyst. Clearly, a larger value for *X* filters out fewer instances. If $X=0$, only instances with exactly the same value for the dependent characteristic are retained. As said, this extreme is undesirable: it would probably remove most of traces, since it is unlikely to have exactly the same value for the dependent characteristic.

In sum, our approach has the advantage of dynamically computing the intervals of the dependent characteristic that determine the clusters to which instances are added. Furthermore, it can determine whether single traces are outliers and, thus, should be excluded from further analysis. When employing regression trees, the determination of the value to assign to the cut-off threshold is not always easy. Section 7.1 discusses an example where it has been crucial to discard a number of traces from a cluster because they were considered as noise. The example also shows how, sometimes, one wants to discard an entire cluster, as each of the comprised traces is, in fact, noise.

7. Implementation and evaluation

The general framework discussed above is implemented as a plug-in of ProM, an open-source "pluggable" framework for the implementation of process mining tools in a standardized environment (see http://www.promtools.org). The ProM framework is based on the concept of packages each of which is an aggregation of several plug-ins that are conceptually related. Our new plug-in is available in a new package named *FeaturePrediction*, which is available in ProM version 6.4.

The main input object of our plug-in is an event log, whereas the output is a decision or a regression tree. The construction of decision and regression trees relies on the implementations of algorithms C4.5 and RepTree developed in the Weka toolkit.[8] As mentioned before, our framework envisions the possibility to augment/manipulate the event logs with additional features. On this concern, the tool is easily extensible: a new log manipulation can be easily plugged in by (1) implementing 3 methods in a Java class that inherits from an abstract class and (2) programmatically adding it to a given Java set of available log manipulations.

This section illustrates how our framework can be used to help UWV. UWV (Employee Insurance Agency) is an autonomous administrative authority to implement employee insurances and provide labor market and data services. Within UWV, several processes are carried on to provide different types of social benefits for residents in the Netherlands. This paper analyzes two of their processes.

The first process is concerned with providing support for Dutch residents when they develop disabilities whilst entitled to unemployment benefits; later on, we refer to this process as *illness-management process*. UWV provides illness benefits for these ill people, hereafter referred to as customers, and provides customized paths for them to return to work. The customer is supported and assessed by a team that consists of a reintegration supervisor, an occupational health expert and a doctor. They cooperate to assess the remaining labor capacity of the employee during the illness. If the person is not completely recovered after two years, the employee can apply for the Work and Income according to Labor Capacity Act (WIA).

The second process is about ensuring that benefits are provided quickly and correctly when a Dutch resident ceases a work contract and cannot immediately find a new employment; later, we refer to this as *unemployment process*. An instance of this process starts when a person, hereafter referred to as customer, applies. Subsequently, checks are performed to verify the entitlement conditions. If the checks are positive, the instance is being executed for the entire period in which the customer receives the monetary benefits, which are paid in monthly installments. Entitled customers receive as many monthly installments as the number of years for which they were working. Therefore, an instance can potentially be executed for more than one year.

UWV is facing various undesired executions of these two processes. For the unemployment process, UWV is interested in discovering the root-causes of a variety of problems identified by UWV's management; for the illness-management process, UWV is interested to gain an insight about how process instances are actually being carried out.

For the *illness-management process*, UWV is interested to discover if there is any correlation between the characteristics of the process instances and the duration of their execution. The reason is that UWV is trying to reduce the duration of the execution of long instances with the purpose of speeding up the reintegration of the customers into the job market. In Section 7.1, we illustrate one analysis use case that aims to discover the causes that determine whether a process execution is longer or shorter. In addition, we show that it is useful to cluster the event log according to this analysis use case. Without clustering the event log, we discover a single process model that does not provide insights into how process instances are carried out: the resulting model is too imprecise as it would allow for too much behavior. Conversely, by splitting the event log into clusters according to the case duration, we can discover insightful process models.

For the *unemployment process*, during the entire period, customers must comply with certain duties, otherwise a customer is sanctioned and a reclamation is opened. When a reclamation occurs, this directly impacts the customer, who will receive lower benefits than expected or has to return part of the benefits. It also has negative impact from UWV's viewpoint, as this tends to consume lots of resources and time. Therefore, UWV is interested to know the root causes of opening reclamations to reduce their number. If the root causes are known, UWV can predict when a reclamation is likely going to be opened and, hence, it can enact appropriate actions to prevent it beforehand. Section 7.2 discusses a number of analysis use cases that aim to predict when reclamations are likely to happen.

7.1. Discovery of the illness management process

For the *illness management process*, UWV is interested to discover how process instances are being carried out. In order to answer this question through the application of process mining, we used an event log *L* containing 1000 traces.

Fig. 4 shows the BPMN process model that has been discovered from the event log with 1000 traces, mentioned above. In particular, we employ the so-called "Inductive Miner" [19] using a noise threshold of 30%. The Inductive Miner generates a Petri net, which has subsequently been converted into a BPMN model [20]. The resulting model contains a loop that internally comprises an exclusive choice between 19 activities. This means that these 19 activities can occur an arbitrary number of times in any order, interleaved with few additional activities. Therefore, the model is clearly

[8] http://weka.sourceforge.net/

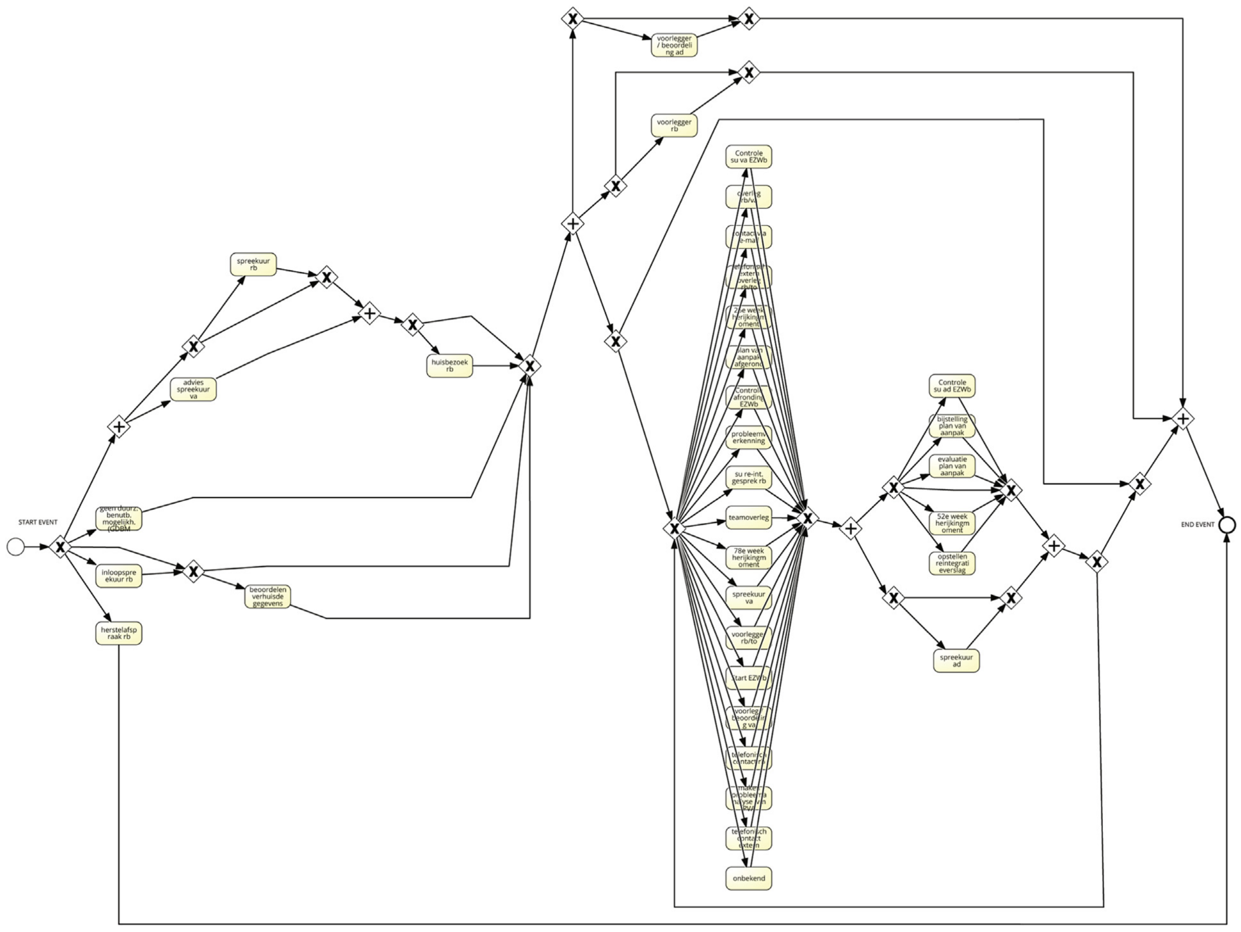


Fig. 4. Discovery of the BPMN process model for the ill-management process when using every trace in log.

unsatisfactory because it is not sufficiently precise: an excessively large variety of behavior is possible.[9]

This result is due to the fact that the miner tries to discover one single process model out of an event log that records heterogenous process behavior. In these situations, as mentioned above, it is advisable to split the event log in clusters, each comprising traces with similar behavior, and to discover one model per cluster.

UWV had the opinion that the behavioral heterogeneity can be tackled by splitting the event log based on the process-instance duration. If instances of the same duration have similar behavior, then this can lead to discover a set of more precise process models. For this purpose, we defined and performed an analysis use case where:

[9] There are various ways to quantify the notion of precision. Here, we employ the notion presented in [11,1]. A model is precise if it does not allow for “too much” behavior. A model that is not precise is “underfitting”. Underfitting is the problem that the model allows for behavior unrelated to example behavior seen in the log. Intuitively, precision is the ratio between the amount of behavior observed in the event log and allowed by the model and the total amount of behavior allowed by the model. Precision is 1 if all modeled behavior was actually observed in the event log. Values close to 0 suggest that the model is underfitting.

Dependent characteristic: Case duration (*TIP4*).

Independent characteristics: The number of executions of each activity x observed in the log ($DFP2_x$) and the latest value observed for each characteristic c presented in log.

Filter: The last event of each trace (*EF3*): each process instance needs to be considered as a whole.

Since the case duration is a numerical value, we opted for generating a regression tree.

7.1.1. Visual interface

Fig. 5 illustrates a screenshot of the tool where we show the regression tree that provides an answer for the analysis use case in question. Before continuing to report on the application of our framework, it is worthwhile to quickly discuss on how regression and decision trees are visualized in the tool.

In particular, each internal node refers to a different process characteristic, which concerns with one of the five process perspectives considered in Section 3: control-flow, data-flow, resource, time and conformance. Compared with the implementation reported in [21], the nodes of the trees are now filled in with a color. The color of an internal node depends on the perspective of the referred process characteristics. For

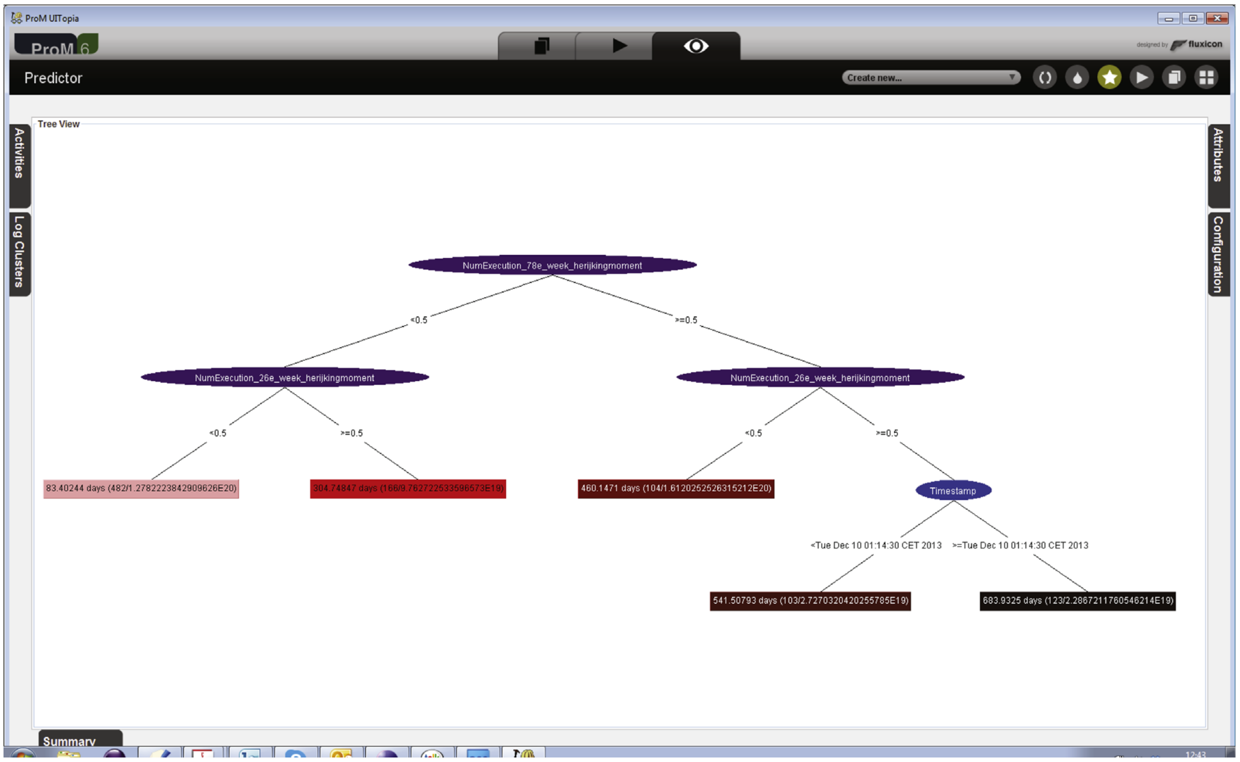

ProM UITopia
ProM 6
designed by fluxicon
Predictor
Create new...
Tree View
Activities
Log Clusters
Attributes
Configuration
NumExecution_78e_week_herijkingmoment
<0.5
>=0.5
NumExecution_26e_week_herijkingmoment
NumExecution_26e_week_herijkingmoment
<0.5
>=0.5
<0.5
>=0.5
83.40244 days (482/1.2782223842909626E20)
304.74847 days (166/9.762722533596573E19)
460.1471 days (104/1.6120252526315212E20)
Timestamp
<Tue Dec 10 01:14:30 CET 2013
>=Tue Dec 10 01:14:30 CET 2013
541.50793 days (103/2.7270320420255785E19)
683.9325 days (123/2.2867211760546214E19)
Summary

Fig. 5. A screenshot of the implementation for ProM. The screenshot shows the regression tree that correlates several process characteristics with the duration of process instances. (For interpretation of the references to colors in this figure caption, the reader is referred to the web version of this paper.)

instance, looking at Fig. 5, one can observe that the internal nodes about the number of executions of activities are colored purple (control-flow perspective), whereas the internal node referring to the timestamp of the process-instance completion is colored blue (time perspective). Leaves are filled in with a color that varies from white to black if a leaf is associated with the lowest or highest value for the dependent characteristic, it is colored white or black, respectively. The color shades from white till black, going through different intensities of yellow, red and brown, as the value associated with the leaf increases. When using a regression tree, each leaf is labeled with the average value of the instances associated with the leaf, followed in brackets by the number of instances for that leaf and the absolute error averaged over all instances associated with that leaf (in milliseconds).

Before concluding, it is good to stress that *end users can click on any tree node and save the associated instances in a file in Attribute-Relation File Format* (*ARFF*)[10] *to be externally loaded in data-mining tools, such as Weka, to employ different data-mining techniques.*

7.1.2. Log cluster

The tree in Fig. 5 contains five leaves and, hence, five clusters are going to be created. However, one cluster needs to be filtered out because it contains traces that have clearly issues, as discussed in the following. The tree illustrates the relation between the duration of process instances and the number of the executions of two activities, namely 26e_week_herijkingmoment (in English: 26th week recalibration time) and 78e_week_herijkingmoment (in English: 78th week recalibration time). As the names suggest, these activities are executed when the customer is being handled since 26 and 78 weeks, respectively.[11]. Therefore, it is normal that, if those are executed, the duration of process instances is respectively longer than 26 or 78 weeks (around 0.5 and 1.5 years).

From the discussion above, it is clear that, if activity 78e_week_herijkingmoment is executed, activity 26e_week_herijkingmoment should also be in the log trace. Nonetheless, in 10% of the process instances, activity 26e_week_herijkingmoment is not performed whereas activity 78e_week_herijkingmoment is executed. This clearly shows that a number of process instances have some issues because the assessment after 26 weeks was skipped, even though it is mandatory according to the internal UWV's procedures. This means that this entire cluster should be discarded as it only contains outlier log traces.

We conclude this discussion by observing that, when both 26e_week_herijkingmoment and 78e_week_herijkingmoment

[10] http://weka.wikispaces.com/ARFF

[11] In particular, these activities concern reassessments of customers to establish the causes why they have not been reintegrated into the job market yet and determine whether they will be ever reintegrated. In fact, if the illness results to remain permanent, it is possible that the customer cannot restart his/her previous employment and, hence, UWV helps finding a new job that suits the new physical situation.

are performed, the process-instance duration depends on when the instance concluded. Process instances that conclude before December 2013 run for a shorter time than those concluded after that date: 541 versus 683 days. Therefore, it is worth splitting the cluster of the traces containing executions of both activities according to the time of process-instance completion.

In the remainder of Section 7.1, we illustrate how the set of models that can be discovered from these clusters allow stakeholders to gain more insight into the process structure.

7.1.3. *Discovery of models from log clusters*

The first cluster refers to those traces where activities 26e_week_herijkingmoment and 78e_week_herijkingmoment were never executed and the process-instance duration was around 83.04 days ± 60%. This cluster contains 179 traces, which were used to discover a process model through the Inductive Miner, using the same noise threshold as previously, followed by the BPMN conversion. The resulting model is depicted in Fig. 6; compared with the model in Fig. 4, the model is much more desirable. First, the model is precise: it does not contain a large exclusive choice. Second, it does not contain arbitrary loops allowing for arbitrary executions of activities: each activity can be executed at most once.

It is worthy observing that, even though Fig. 5 shows that 482 traces are such that 26e_week_herijkingmoment and 78e_week_herijkingmoment were never executed, the corresponding cluster only contains 179 traces. This indicates that the choice of only retaining traces with an error less than 60% has caused around 62% of the traces to be filtered out. Although this may seem to not be a good choice, it can be easily motivated. Many of the traces that have been filtered out contain less than 5 events, which might indicate that those are incomplete cases.

One might raise objections concerning the fact that, if only 179 out of 1000 traces are retained, less behavior is observed in the event log and, thus, it is natural to obtain more precise models. However, this does not apply to this case study as discussed below. First, we sampled 200 random traces four times, thus generating 4 event logs. Afterwards, we mined 4 process models: all models were very imprecise, similar to the model in Fig. 4 and definitely different from Fig. 6.

Fig. 7 illustrates the process model discovered for the long cases, where activities 26e_week_herijkingmoment and 78e_week_herijkingmoment were both executed at least once, the timestamp when the case concluded was before 10 December 2013 and the process-instance duration was around 541.75 days ± 60%. The log cluster contained around 100 traces. The model is highly precise: no exclusive choice is presented among several alternative activities. The inclusion of the timestamp as independent characteristic in the analysis use case is very important to discover a precise model. If we exclude the timestamp, we create one single cluster for each trace in which both 26e_week_herijkingmoment and 78e_week_herijkingmoment occur, unrespectfully when each trace concludes. The discovered model is similar to the model in Fig. 4 and, hence, insufficiently precise. This clearly illustrates that during 2013, the structure of the process and the order with which activities are performed has gone through a radical change. In other words, using the consolidated process-mining terminology, the process has experienced a so-called *concept drift*. This is also confirmed by the UWV, which was aware of a concept drift at beginning of 2014, even though it did not expect to observe so radical changes.

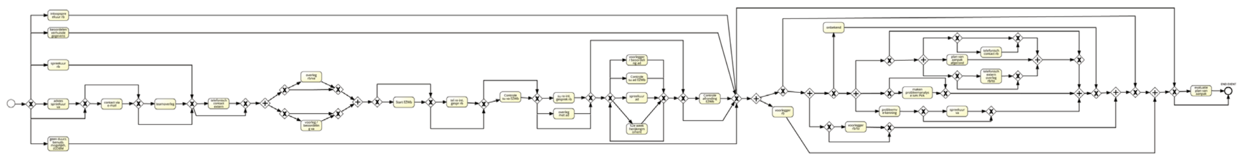


Fig. 6. Discovery of the BPMN process model using the clustering of the traces where activities 26e_week_herijkingmoment and 78e_week_herijkingmoment were never executed and the process-instance duration was around 83.04 days ± 60%. The model is significantly more precise than the model discovered when all traces are considered together.

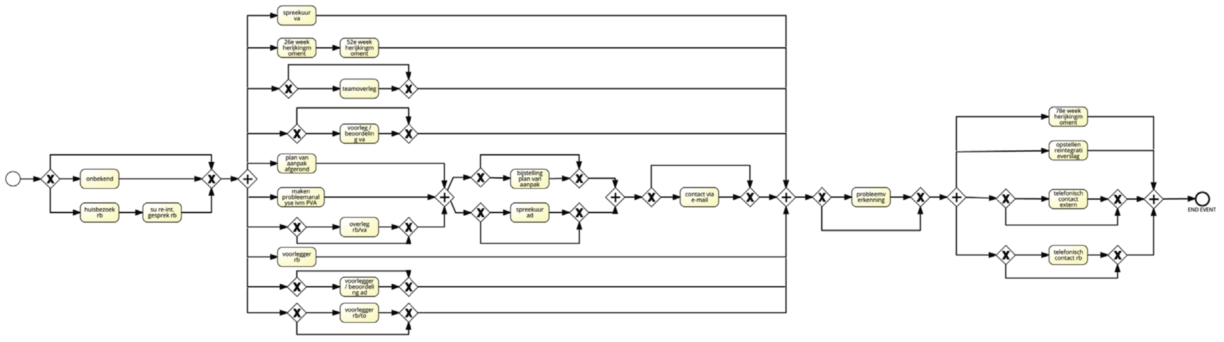

spreekuur va
26e week herijkingm oment
52e week herijkingm oment
teamoverleg
voorleg / beoordelin g va
onbekend
huisbezoek rb
su re-int. gesprek rb
plan van aanpak afgerond
maken probleemanal yse ivm PVA
bijstelling plan van aanpak
spreekuur ad
overleg rb/va
voorlegger rb
voorlegger / beoordeli ng ad
voorlegger rb/to
contact via e-mail
probleemv erkenning
78e week herijkingm oment
opstellen reintegrati everslag
telefonisch contact extern
telefonisch contact rb
END EVENT

Fig. 7. Discovery of the BPMN process model using the clustering of the traces where activities 26e_week_herijkingmoment and 78e_week_herijkingmoment were both executed at least once, the timestamp when the case concluded was before 10 December 2013 and the process-instance duration was around 541.75 days ± 60%. The model is again quite well structured and precise: there are no exclusive choices among several alternatives.

Space limitations prevent us from showing the model derived from the cases where activity 26e_week_herijkingmoment was executed at least once, activity 78e_week_herijkingmoment was never executed and the process-instance duration was around 304.75 days $\pm$ 60%. As mentioned above, we excluded from the analysis the cluster of the traces where activity 26e_week_herijkingmoment was never executed whereas activity 78e_week_herijkingmoment was at least once. Those traces show undesirable behavior: the customer assessment did not occur during the 26th week, even though the UWV's procedure prescribes it.

7.2. *Analysis of the unemployment process*

This section is looking at the process to deal with requests of unemployment benefits. As mentioned at the beginning of this section, UWV wants to investigate what factors yield reclamations to occur with the aim of preventing them from happening. In order to discover the root causes, UWV formulated four questions:

Q1 Are customer characteristics linked to the occurrence of reclamations? And if so, which characteristics are most prominent?
Q2 Are characteristics concerned with how process instances are executed linked to the occurrence of reclamations? And if any, which characteristics matter most?
Q3 If the prescribed process flow is not followed, will this influence whether or not a reclamation occurs?
Q4 When an instance of the unemployment-benefit payment process is being handled, is there any characteristic that may trigger whether a reclamation is going to occur?

Table 5 enumerates some of the analysis use cases that have been performed to answer the questions above. The analyses have been performed using a UWV's event log containing 2232 process instances and 77 551 events. The remainder of this section details how the analysis use cases have been used to answer the four questions above.

The use cases *U1*, *U2*, *U3* use the number of executions of *Reclamation* as dependent characteristic. For those analysis use cases, we opted to use decision trees and discretize the number of executions. This is because UWV is interested to clearly distinguish cases in which no reclamation occurs versus those in which reclamations occur. The number of executions of *Reclamation* is discretized as two values: `(0.0,0.0)` and `(0.0,5.0)`. When the number of executions of *Reclamation* is 0, this is shown as `(0.0,0.0)`; conversely, any value greater than 0 for the number of executions is discretized as `(0.0,5.0)`. We used the same discretization for *U1*, *U2*, *U3*.

Question Q1: To answer this question, we performed the use case **U1** in Table 5. The results of performing this analysis are represented through the decision tree in Fig. 8. In particular, the screenshot refers to our implementation in ProM. The implementation allows the end user to configure a number of parameters, such as the level of decision-tree pruning, the minimum number of instances per leaf or the discretization method. In this way, the user can try several configurations, thus, e.g., balancing between over- and under-fitting. In particular, the screenshot refers to the configuration in which the minimum number of instances per leaf is set to 100.

Looking at the tree in Fig. 8, some business rules seem to be derived. For instance, if the customer is a recurrent customer ($WW\_IND\_HERLEVING > 0$), a reclamation occurs, i.e. the leaf is labeled as `(0.0,5.0)`.[12] If this correlation really held, it would be quite unexpected: recurrent customers tend to disregard their duties. Nonetheless, the label is also annotated with `318.0/126.0`, which indicates that a reclamation is not opened for 126 out of the 318 recurrent customers (39%). Though not very strong, a correlation seems to exist between being recurrent customers and incurring in reclamations. Further investigation is certainly needed; perhaps, additional customer's characteristics might be needed to better discriminate, but they are currently not present in the event log used for analysis.

Question Q2: Firstly, we performed the analysis use case **U2**. We obtained a decision tree that showed correlations between the number of reclamations and certain characteristics that are judged as trivial by UWV. For instance, there was a correlation of the number of reclamations with (1) the method of payment of the benefit installments to customers and (2) the number of executions of activity *Call Contact door HH deskundige*, which is executed to push customers to perform their duties. Being these correlations considered trivial by UWV, the respective characteristics should be left out of the analysis. So, we excluded these characteristics from the set of independent characteristics and repeated the analysis. We refined the analysis use case multiple times by removing more and more independent characteristics. After 9 iterations, we performed an analysis use case that led to satisfactory results. The results of performing this analysis are represented through the decision tree in Fig. 9, which classifies 77% of the instances correctly.

This tree illustrates interesting correlation rules. Reclamations are usually not opened in those process instances in which (1) UWV never informs (or has to inform) a customer about changes in his/her benefits (the number of executions of *Brief Uitkering gewijzigd WW* is 0), (2) UWV's employees do not hand over work to each other (the number of executions of *Brief Interne Memo* is 0) and (3) either of the following conditions holds:

- no letter is sent to the customers (the number of executions of *Brief van UWV aan Klant* is 0);
- at least one letter is sent but UWV never calls the customer (the number of executions of *Call Telefoonnotitie* is equal to 0) and, also, the number of months for which the customer is entitled to receive a benefit is more than 12.

[12] Customers are recurrent if they apply for monetary benefits multiple times because they find multiple temporary jobs and, hence, they become unemployed multiple times.

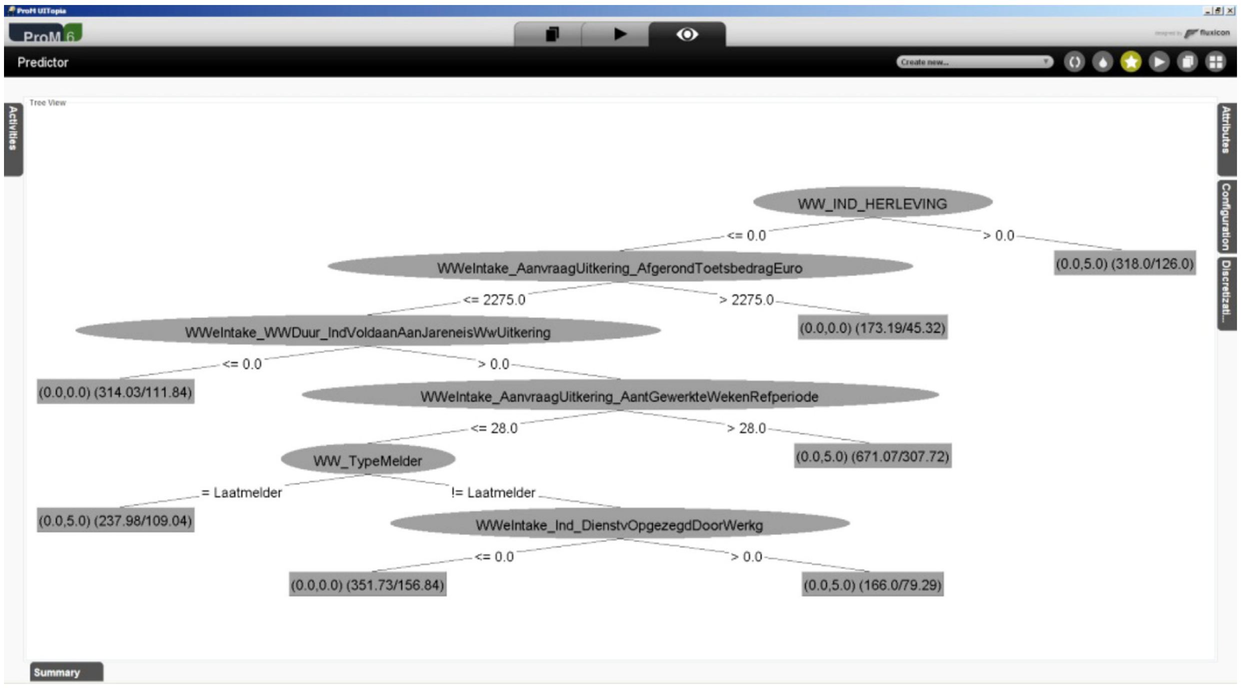
Table 5
Additional analysis use cases that illustrate how our framework works to unify existing approaches to correlate specific process characteristics.

U1. Analysis Use Case ($CFP1_{Reclamation}, \{\forall_{customercharacteristicsc}\ DFP1_c\}, EF2$): **Are customer characteristics linked to the occurrence of reclamations?** We aim to correlate the number of executions of activity *Reclamation* to the customer characteristics. Customer characteristics are a subset of the entire set of characteristics available in the original log, i.e. before applying any trace manipulation. For this analysis use case, we aim to analyze each trace as a whole and, hence, we use event filter *EF*2 to only retain the last event of each trace. The dependent characteristic is $CFP1_{Reclamation}$ ("The Number of Executions of Activity *Reclamation*") and the set of independent characteristics includes $DFP1_c$ ("Latest Recorded Value of Characteristic *c* Until Current Event") for each customer characteristic *c* of the original event log

U2. Analysis Use Case ($CFP1_{Reclamation}, \{\forall_{process\ characteristics\ c}\ DFP1_c, TIP2, \forall_{activity\ x\ different\ from\ Reclamation}\ CFP1_x, \}, EF2$): **Are characteristics concerned with how process instances are executed linked to the occurrence of reclamation?** We aim to correlate the number of executions of activity *Reclamation* to the number of executions of every activity, process-related characteristics, the elapsed time, i.e. the time since process instances are started. Process-related characteristics are a subset of the entire set of characteristics available in the original log, i.e. before applying any trace manipulation. They differ from the customer characteristics because they are related to the outcome of the management of the provision of unemployment benefits. For this analysis use case, we aim to analyze each trace as a whole and, hence, use event filter *EF2* to only retain the last event of each trace. The dependent characteristic is $CFP1_{Reclamation}$ ("The Number of Executions of Activity *Reclamation*") and the set of independent characteristics includes $DFP1_c$ ("Latest Recorded Value of Characteristic *c* Until Current Event") for each process-related characteristic *c* of the original event log, *TIP2* ("Elapsed Time since the Start of the Case") and $CFP1_x$ ("Number of Executions of Activity *x*") for each process activity *x*

U3. Analysis Use Case ($CFP1_{Reclamation}, \{COP1_P, \forall_{activity\ x\ different\ from\ "Reclamation"}\ COP2_{P,x}, COP3_{P,x}, COP4_{P,x}\}, EF2$): **If the prescribed process flow is not followed, will this influence whether or not a reclamation occurs?** We aim to correlate the number of executions of activity *Reclamation* to the trace fitness, the number of non-allowed, missing and correct executions of any other activity. For this analysis use case, a process model *P* is provided. Since we again aim to analyze each trace as a whole, we use event filter *EF2* to only retain the last event of each trace. The dependent characteristic is $CFP1_{Reclamation}$ ("The Number of Executions of Activity *Reclamation*") and the set of independent characteristics include the fitness of each trace wrt model *P* (manipulation $COP1_P$) as well as, for each activity *x* different from *Reclamation*, $COP2_{P,x}$ ("Number of Not Allowed Executions of Activity *x* Thus Far"), $COP3_{P,x}$ ("Number of Missing Executions of Activity *x* Thus Far") and $COP4_{P,x}$ ("Number of Correct Executions of Activity *x* Thus Far")

U4. Analysis Use Case ($CFP2_{All}, \{TIP2, \forall_{process\ characteristics}c\ DFP1_c, \forall_{activity\ x\ different\ from\ "Call\ Contact\ door\ HH\ deskundige"}\ CFP1_x, \}, EF4$) **When an instance of the unemployment process is handled, is there any characteristic that may trigger whether a reclamation is going to occur next?** We aim to predict when a reclamation is going to occur as the next activity during the execution of a process instance. For this purpose, we predict which activity is going to follow for any process activity and, then, we focus on the paths leading to predicting that the following activity is, indeed, *Reclamation*. For this analysis use case, differently from the others in the table, we use any complete event; thus, the event filter is *EF4* ("Keep All Complete Events"). The dependent characteristic is $CFP2_{All}$, indicating the first occurrence of any activity after the current event, i.e. the next activity in the trace. The set of independent characteristics includes *TIP2* ("Time Elapsed since the Start of the Case" as well as $DFP1_c$ ("Latest Recorded Value of Characteristic *c* Until Current Event") for each process-related characteristic *c* and $CFP1_x$ ("Number of Executions of Activity *x*") for each process activity *x* different from *Call Contact door HH deskundige*



WW_IND_HERLEVING
<= 0.0 | > 0.0 → (0.0,5.0) (318.0/126.0)
WWeIntake_AanvraagUitkering_AfgerondToetsbedragEuro
<= 2275.0 | > 2275.0 → (0.0,0.0) (173.19/45.32)
WWeIntake_WWDuur_IndVoldaanAanJareneisWwUitkering
<= 0.0 → (0.0,0.0) (314.03/111.84) | > 0.0
WWeIntake_AanvraagUitkering_AantGewerkteWekenRefperiode
<= 28.0 | > 28.0 → (0.0,5.0) (671.07/307.72)
WW_TypeMelder
= Laatmelder → (0.0,5.0) (237.98/109.04) | != Laatmelder
WWeIntake_Ind_DienstvOpgezegdDoorWerkg
<= 0.0 → (0.0,0.0) (351.73/156.84) | > 0.0 → (0.0,5.0) (166.0/79.29)

Fig. 8. A screenshot of the framework's implementation in ProM that shows the decision tree used to answer question Q1.

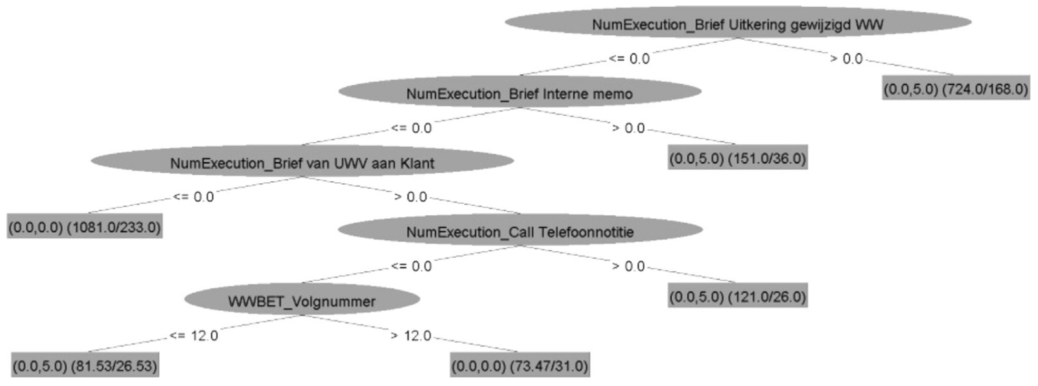


Fig. 9. The decision tree used to answer question Q2.

From this analysis, we can conclude that UWV should reduce the hand-over of work. Moreover, it should pay more attention to customers when their situation changes, e.g. they find a new job. When customers find a job, they start having a monetary income, again. The problem seems to be related to the customers who often do not provide information about the new job on time. In these cases, their benefits are not stopped or reduced when they should. Consequently, a reclamation needs to be opened because these customers need to return the amount that was overpaid to them. Conversely, if a customer has already received benefits for 12 months, it is unlikely that a reclamation is going to occur. This can be motivated quite clearly and it is again related to the presence of changes of the customer's job situation. If benefits are received for more than 12 months, the customer has not found a job in the latest 12 months and, thus, it is probably going to be hard for him to find one. So, UWV does not have to pay much attention to customers entitled to long benefits when they aim to limit the number of reclamations.

Question Q3: The answer to this question is given by performing the analysis use case **U3** in Table 5. This use case relies on a process model that describes the normal execution flow. This model was designed by hand, using knowledge of the UWV domain. The results of performing **U3** are represented by the decision tree in Fig. 10. Analyzing the decision tree, a correlation is clear between trace fitness and the number of reclamations. 610 out of the 826 process executions (nearly 70%) with fitness higher than 0.89 do not comprise any reclamation. Therefore, it seems crucial for UWV to make the best to follow the normal flow, although this is often made difficult by a hasty behavior of customers. This rule seems quite reliable and is also confirmed by the fact that 70% of the executions with fitness lower than 0.83 incur in reclamations.

The decision tree contains an intermediate node labeled *MREAL for IKF van Klant aan UWV*. This characteristic refers to the number of missing executions of activity *IKF van Klant aan UWV*. This activity is executed in a process instance every time that UWV receives a declaration form from the customer. UWV requests customers to send a form every month to declare whether or not their condition has changed in the last month, e.g. they found a job. The decision tree states that, when an execution deviates moderately, i.e. the fitness is roughly between 0.83 and 0.89, a reclamation is still unlikely being opened if the customer forgets to send the declaration form for at most 3 months (not necessarily in a row). Note that, since traces are quite long, considering how fitness is computed, a difference of 0.06 in fitness can be quite remarkable. This rule is quite reliable since it holds in 79% of cases. Therefore, it is worthwhile for UWV to enact appropriate actions (such as calling by phone) to increase the chances that customers send the declaration form every month.

Question Q4: The answer to this question is given by performing the analysis use case **U4** in Table 5. We built a decision tree for this use case by limiting the minimal number of instances per leaf to 50. We are interested in tree paths that lead to *Reclamation* as next activity in the trace. Unfortunately, the F-measure for *Reclamation* was very low (0.349), which indicates that *it is not possible to reliably estimate if a reclamation is going to occur at a certain moment of the execution of a process instance*. We also tried to reduce the limit of the minimum number of instances per leaf. Unfortunately, the resulting decision tree was not valuable since it overfitted the instance sets: the majority of the leaves were associated to less than 1% of the total number of instances. Conversely, the decision tree with 50 as minimum number of instances per leaf could be useful to predict when a payment is sent out to a customer: the F score for the payment activity is nearly 0.735. Unfortunately, finding this correlation does not answer question Q4.

7.3. Final remarks

As mentioned in Section 1, we do not claim that our framework is able to perform analyses that previously were not possible. For some types of analysis use cases, dedicated solutions were provided and, for other types of analysis use cases, the analyst had to manually go through several transformation and selection steps. The novelty of our framework that we provide a single environment where analyses can be performed much quicker and do not require process analysts to have solid technical background.

In the remainder of this section, we illustrate the complexity and time consumption of performing analysis use cases without the framework. For this aim, we

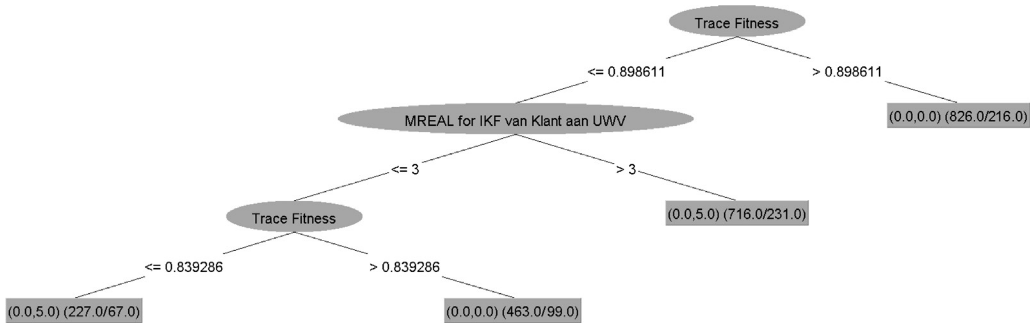


Fig. 10. The decision tree used to answer question Q3.

Table 6
Create of the set of learning instances using SQL in place of our framework.

```
CREATE FUNCTION NumReclamation (@cId)
AS (select count(*) from event_table as et where et.cId=@cId and et.activity='Reclamation')

CREATE FUNCTION LatestValue (@cId,@Attr) AS(
  select @Attr from event_table as et1 where et1.cId=@cId and et2.@Attr is not null
  and 0=(select count(*) from event_table as et2 where et2.cId=et1.cId and et2.@Attr is not null
    and et2.timestamp > et1.timestamp ))

select et1.*,NumReclamation(et.cID),LatestValue(et.cID,Attr_1),...,LatestValue(et.cID,Attr_n)
from event_table et1 where et.timestamp = (select max(timestamp) from event_table et2 where et2.cId = et.cId)
```

consider Question Q1 introduced in Section 7.2, whose corresponding analysis use case is in Table 5. For simplicity, let us suppose that the event log is stored in a database table `event_table` of a certain database. Each table's row is a different log's event. This table contains columns `cId`, `timestamp`, `activity`, which represent the identifier of the case (i.e., of the process instance), the event's timestamp and the activity name, respectively. In addition to that, each additional process attribute is characterized by a respective column `Attr_1`,…,`Attr_n`. In order to perform the analysis use case, we aim to write a SQL query that returns a tabular result that can be exported in a CSV file and loaded in data-mining tools. Table 6 shows the corresponding SQL query; in particular, to increase the readability, we introduce two SQL functions that compute the number of occurrences of activity *Reclamation* in a process instance with identifier `cId` and the latest value written for a certain process attribute `Attr` for a process instance with identifier `cId`.

Space limitations prevent us from detailing how the SQL query and the helper functions are structured. However, we believe that this shows how long, error-prone and time-consuming the task of manually defining such SQL queries is for each analysis use case. Furthermore, process analysts are required to possess a large knowledge of SQL, which is not always true. If we replaced SQL and databases with alternative languages, the complexity would be certainly comparable. Here we have showcased an analysis use case for Q1 that is relatively simple if compared with other analysis use cases discussed in this paper. For instance, without our framework the analysis use case for Q3 would require to write complex codes or scripts to import the conformance results.

Conversely, using our framework and our corresponding ProM implementation, the task of defining analysis use cases is simple while remaining relatively generic. This can be done in a matter of minutes by simply selecting the opportune dependent and independent characteristics from a provided list. The complexity would only be raised if one needed to incorporate new log-manipulation functions.

8. Related work

This paper has clearly shown that the study of process characteristics and how they influence each other is of crucial importance when an organization aims to improve and redesign its own processes. Many authors have proposed techniques to relate specific characteristics in an ad ad-hoc manner. For example, several approaches have been proposed to predict the remaining processing time of a case depending on characteristics of the partial trace executed [22,18,23]. Other approaches are only targeted to correlating certain predefined characteristics to the process outcome [15,24,16], process service time [25] or the violations of business rules [14]. Senderovich et al. [26] leverages on queue theory to correlate service time to the arrival of new activities to perform; the main drawback is that the approach can be only applied to single activities and assumes a certain arrival distribution.

The general framework proposed in this paper is in line with typical methodologies to discover knowledge, intended as a set of interesting and nontrivial patterns, from data sets with some degree of certainty. In [27], Fayyad et al. propose a methodology and guidelines to execute the process of knowledge discovery. A number of abstract steps are proposed: selection, pre-processing, transformation, data mining and interpretation. The similarity between these steps and those proposed by our framework is quite evident. The selection and pre-processing steps match the definition of the analysis use case, whereas the transformation step coincides with the manipulation of event logs as discussed in Section 3. Finally, the data mining step in the methodology of Fayyad et al. corresponds to the actual construction of decision and regression trees to provide answers to analysis use cases. While our framework shares commonalities with the methodology of Fayyad et al., several major differences exist. It is not in the scope of the methodology of Fayyad et al. to provide a concrete framework and operationalization that allow non-experts to easily conduct a process-related analysis.

Furthermore, our paper adapts the methodology of Fayyad et al. for the domain of process mining. The fundamental difference between data and process mining is that process mining is concerned with processes. As extensively discussed, processes take several perspectives into account. Therefore, these perspectives need to be elected as first-class citizens to obtain reliable correlations. The outcomes, the time for executing a certain activity or any other characteristic in a process instance may depend on several factors that are outside the activity in question. For instance, resource and group workloads, delays, deadlines, the customer type (e.g., silver, gold or platinum) or the number of undergone checks may have strong influence. Factors can be even outside the specific process instance; namely, these factors are part of the external context around which the activity is being executed. Without taking them into consideration, the correlation is certainly weaker. Therefore, each event cannot be considered in isolation but should consider the other events and, even, aspects outside the process in question. Somehow, Data Mining tends to take events, i.e. the learning instances, in isolation. This also motivates the importance of augmenting and manipulating the event log.

In addition to decision and regression trees, many other machine-learning techniques exist and some have already been applied in BPM, such as Bayesian Networks [28], Case-Based Reasoning [29], Markov Models [23], Naive Bayes Classifiers and Support Vector Machines [30]. These are certainly valuable but they are only able to make correlations for single instances of interest or to return significant examples of relevant instances. Conversely, we aim to aggregate knowledge extracted from the event logs and return it as decision rules. Association rules [3] could have been an alternative, but decision and regression trees have the advantage of clearly highlighting the characteristics that are most discriminating. Also, linear regression [7] could be applied; however, this could only be applied when dependent and independent characteristics are numerical, thus excluding many valuable process characteristics. Moreover, we believe that linear formulas would not be understood by non-expert users as much as decision or regression trees would. Last but not least, stochastic analysis has been used to predict the remaining service time [31]; the approach requires to have a process model and it is not clear how it can be generalized to any process characteristics.

There is also a body of work that is concerned with providing intuitive visual representations of root causes of problems or undesirable events, such as Cause–Effect or Why-Why Diagrams (see, e.g., [32, pp. 190–198]). We envision these intuitive visualizations as complementary to our framework and visualization. Furthermore, before drawing such diagrams, one needs to gain the necessary insights, which our framework facilitates to obtain.

This paper extends the framework initially proposed in [21] to allow grouping the traces of an event log into clusters. Literature proposes a large number of approaches to address this topic, e.g. [33–38]. All existing approaches share the same idea: similar traces are grouped together and placed in the same cluster; the difference among these approaches resides on the use of different notions of trace similarity. However, existing approaches propose a predetermined repertoire of similarity's notion, whereas this paper proposes an approach to trace clustering that is based on a configurable notion of trace similarity. We believe that there is no concept of trace similarity that is generally applicable in every situations; in fact, the similarity among traces depends on the specific process domain. Furthermore, the predetermined repertoire is mostly based on the control-flow perspective (e.g. the order with which activities are performed). Conversely, our framework enables to define the similarity concept as a function of characteristics related to several perspectives: the duration of the process-instance execution, the workload of the process participants when the instance was carried out, the execution's compliance and many others.

In addition to using different concepts of trace similarity, existing approaches also use different clustering techniques, such as agglomerative hierarchical clustering, k-means or DBScan [7]. When using these techniques for trace clustering, process analysts do not have a clear insight into the process characteristics that are common among all traces within the same cluster. Conversely, decision or regression trees clearly highlight the characteristics that discriminate. This means that, when using it for trace clustering, it becomes clear why a certain log trace belongs to a given cluster and not to another.

9. Conclusion

Process mining can be viewed as the missing link between model-based process analysis and data-oriented analysis techniques. Lion's share of process mining research has been focusing on process discovery (creating process models from raw data) and replay techniques to check conformance and analyze bottlenecks. It is crucial that certain phenomena can be explained, e.g., “Why are these cases delayed at this point?”, “Why do these deviations take place?”, “What kind of cases are more costly due to following this undesirable route?”, and “Why is the distribution of work so unbalanced”. These and other questions may involve process characteristics related to different perspectives (control-flow, data-flow, time, organization, cost, compliance, etc.). Specific questions (e.g., predicting the remaining processing time) have been investigated before, a generic framework for correlating business process characteristics was missing. In this paper, we presented such a framework and its implementation in ProM. By defining an analysis use case composed of three elements (one dependent characteristic, multiple independent characteristics and a filter), we can create a classification or regression problem. The results of performing an analysis use case is a decision or a regression tree that describes the dependent characteristic in terms of the independent characteristics. As discussed, the resulting decision or regression tree can also be used to split event logs into clusters of traces with similar behavior. Clustering is a necessary action when the event log shows such a heterogenous behavior that many process-mining techniques, such as for process discovery, cannot return insightful results. The advance of the proposed clustering technique is that the notion of trace similarity is configurable according to the specific application domain and can leverage on many process perspectives, instead of only the control-flow perspective.

The approach has been evaluated using a case study within the UWV, one of the largest "administrative factories" in the Netherlands. The evaluation has demonstrated the usefulness of performing correlation analyses to gain insight into processes as well as of clustering event logs according to the results of performing analysis use cases.

Acknowledgments

The work of Dr. de Leoni has received funding from the European Community's Seventh Framework Program FP7 under Grant agreement num. 603993 (CORE).

References

[1] W.M.P. van der Aalst, Process Mining: Discovery, Conformance and Enhancement of Business Processes, 1st edition, Springer Publishing Company, Incorporated, 2011.
[2] J. Nakatumba, Resource-aware business process management: analysis and support (Ph.D. thesis), Eindhoven University of Technology, ISBN: 978-90-386-3472-2, 2014.
[3] A. Dohmen, J. Moormann, Identifying drivers of inefficiency in business processes: a DEA and data mining perspective, in: Enterprise, Business-Process and Information Systems Modeling, LNBIP, vol. 50, Springer, Berlin, Heidelberg, 2010, pp. 120–132.
[4] L. Zeng, C. Lingenfelder, H. Lei, H. Chang, Event-driven quality of service prediction, in: Proceedings of the 8th International Conference of Service-Oriented Computing (ICSOC 2008), Lecture Notes in Computer Science, vol. 5364, Springer, Berlin, Heidelberg, 2008, pp. 147–161.
[5] W.M.P. van der Aalst, S. Dustdar, Process mining put into context, IEEE Internet Comput. 16 (1) (2012) 82–86.
[6] W.M.P. van der Aalst, Process cubes: slicing, dicing, rolling up and drilling down event data for process mining, in: M. Song, M. Wynn, J. Liu (Eds.), Asia Pacific Conference on Business Process Management (AP-BPM 2013), LNBIP, vol. 159, Springer-Verlag, Beijing, China, 2013, pp. 1–22.
[7] T.M. Mitchell, Machine Learning, 1st edition, McGraw-Hill, Inc., New York, NY, USA, 1997.
[8] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, 2nd edition, Morgan Kaufmann Series in Data Management Systems, 3rd edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2011.
[9] J. Dougherty, R. Kohavi, M. Sahami, Supervised and unsupervised discretization of continuous features, in: Proceedings of the Twelfth International Conference on Machine Learning (ICML'95), Morgan Kaufmann, Tahoe City, California, USA, 1995, pp. 194–202.
[10] W.M.P. van der Aalst, H.T. Beer, B.F. van Dongen, Process mining and verification of properties: an approach based on temporal logic, in: Conference on the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, Lecture Notes in Computer Science, vol. 3760, Springer, Berlin, Heidelberg, 2005, pp. 130–147.
[11] W.M.P. van der Aalst, A. Adriansyah, B.v. Dongen, Replaying history on process models for conformance checking and performance analysis, WIREs Data Min. Knowl. Discov. 2 (2) (2012) 182–192.
[12] M. de Leoni, W.M.P. van der Aalst, Aligning event logs and process models for multi-perspective conformance checking: an approach based on integer linear programming, in: Proceedings of the 11th International Conference on Business Process Management (BPM'13), Lecture Notes in Computer Science, vol. 8094, Springer-Verlag, Beijing, China, 2013, pp. 113–129.
[13] M. de Leoni, F.M. Maggi, W.M.P. van der Aalst, An alignment-based framework to check the conformance of declarative process models and to preprocess event-log data, Inf. Syst. 47 (2015) 258–277.
[14] F.M. Maggi, C.D. Francescomarino, M. Dumas, C. Ghidini, Predictive monitoring of business processes, in: Proceedings of the 26th International Conference on Advanced Information Systems Engineering (CAiSE 2014), Lecture Notes in Computer Science, vol. 8484, 2014, pp. 457–472.
[15] J. Ghattas, P. Soffer, M. Peleg, Improving business process decision making based on past experience, Decis. Support Syst. 59 (2014) 93–107.
[16] R. Conforti, M. de Leoni, M. La Rosa, W.M.P. van der Aalst, Supporting risk-informed decisions during business process execution, in: Proceedings of the 25th International Conference on Advanced Information Systems Engineering (CAISE'13), Lecture Notes in Computer Science, vol. 7908, Springer-Verlag, Valencia, Spain, 2013, pp. 116–132.
[17] A. Rozinat, W.M.P. van der Aalst, Decision mining in ProM, in: Proceedings of the 4th International Conference on Business Process Management (BPM'06), Lecture Notes in Computer Science, Springer-Verlag, Vienna, Austria, 2006, pp. 420–425.
[18] W.M.P. van der Aalst, M.H. Schonenberg, M. Song, Time prediction based on process mining, Inf. Syst. 36 (2) (2011) 450–475.
[19] S.J. Leemans, D. Fahland, W.M.P. van der Aalst, Discovering block-structured process models from incomplete event logs, in: Proceedings of the 35th International Conference on Application and Theory of Petri Nets and Concurrency (Petri Net 2014), vol. 8489, Springer International Publishing, Tunis, Tunisia, 2014, pp. 91–110.
[20] A. Kalenkova, M. de Leoni, W.M.P. van der Aalst, Discovering, analyzing and enhancing BPMN models using ProM, in: Proceedings of the Demo Sessions of the 12th International Conference on Business Process Management (BPM 2014), CEUR Workshop Proceedings, vol. 1295, CEUR-WS.org, 2014, p. 36.
[21] M. de Leoni, W.M. van der Aalst, M. Dees, A general framework for correlating business process characteristics, in: Proceedings of the 11th Business Process Management (BPM'14), Lecture Notes in Computer Science, vol. 8659, Springer International Publishing, Eindhoven, The Netherlands, 2014, pp. 250–266.
[22] F. Folino, M. Guarascio, L. Pontieri, Discovering context-aware models for predicting business process performances, in: On the Move to Meaningful Internet Systems: OTM 2012, Lecture Notes in Computer Science, vol. 7565, Springer, Berlin, Heidelberg, 2012, pp. 287–304.
[23] G. Lakshmanan, D. Shamsi, Y. Doganata, M. Unuvar, R. Khalaf, A Markov prediction model for data-driven semi-structured business processes, Knowl. Inf. Syst. (2013) 1–30.
[24] A. Kim, J. Obregon, J.-Y. Jung, Constructing decision trees from process logs for performer recommendation, in: Proceedings of 2013 Business Process Management Workshops, LNBIP, vol. 171, Springer, Beijing, China, 2014, pp. 224–236.
[25] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Mining resource scheduling protocols, in: Proceedings of the 11th Business Process Management (BPM'14), Lecture Notes in Computer Science, vol. 8659, Springer International Publishing, Eindhoven, The Netherlands, 2014, pp. 200–216.
[26] A. Senderovich, M. Weidlich, A. Gal, A. Mandelbaum, Queue mining – predicting delays in service processes, in: Proceedings of CAiSE, Lecture Notes in Computer Science, vol. 7565, Springer, Thessaloniki, Greece, 2014, pp. 42–57.
[27] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, From data mining to knowledge discovery: an overview, in: U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy (Eds.), Advances in Knowledge

Discovery and Data Mining, American Association for Artificial Intelligence, 1996, pp. 37–54.

[28] R.A. Sutrisnowati, H. Bae, J. Park, B.-H. Ha, Learning bayesian network from event logs using mutual information test, in: Proceedings of the 6th International Conference on Service-Oriented Computing and Applications (SOCA 2013), 2013, pp. 356–360.

[29] A. Aamodt, E. Plaza, Case-based reasoning: foundational issues, methodological variations, and system approaches, AI Commun. 7 (1) (1994) 39–59.

[30] M. Polato, A. Sperduti, A. Burattin, M. de Leoni, Data-aware remaining time prediction of business process instances, in: Proceedings of the 2014 International Joint Conference on Neural Networks (IJCNN), 2014.

[31] A. Rogge-Solti, M. Weske, Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays, in: Proceedings of ICSOC, Lecture Notes in Computer Science, vol. 8274, Springer, Berlin, Germany, 2013.

[32] M. Dumas, M. la Rosa, J. Mendling, H.A. Reijers, Fundamentals of Business Process Management, Springer, Berlin, Heidelberg, 2013.

[33] C.C. Ekanayake, M. Dumas, L. Garca-Bauelos, M. La Rosa, Slice, mine and dice: complexity-aware automated discovery of business process models, in: Proceedings of the 10th Business Process Management (BPM'13), Lecture Notes in Computer Science, vol. 8094, Springer, Berlin, Heidelberg, 2013, pp. 49–64.

[34] R.P.J.C. Bose, Process mining in the large: preprocessing, discovery, and diagnostics (Ph.D. thesis), Eindhoven University of Technology, Eindhoven, 2012.

[35] G.M. Veiga, D.R. Ferreira, Understanding spaghetti models with sequence clustering for ProM, in: Business Process Management Workshops, Lecture Notes in Business Information Processing, vol. 43, Springer, Berlin, Heidelberg, 2010, pp. 92–103.

[36] G. Greco, A. Guzzo, L. Pontieri, Mining taxonomies of process models, Data Knowl. Eng. 67 (1) (2008) 74–102.

[37] J. de Weerdt, S. vanden Broucke, J. Vanthienen, B. Baesens, Active trace clustering for improved process discovery, IEEE Trans. Knowl. Data Eng. 25 (12) (2013) 2708–2720.

[38] M. Sole, J. Carmona, A high-level strategy for c-net discovery, in: Proceedings of the 12th International Conference on Application of Concurrency to System Design (ACSD), 2012, pp. 102–111.