Specification-Driven Multi-perspective Predictive Business Process Monitoring

Ario Santoso[1,2(✉)]

[1] Department of Computer Science, University of Innsbruck, Innsbruck, Austria
[2] Faculty of Computer Science, Free University of Bozen-Bolzano, Bolzano, Italy
ario.santoso@uibk.ac.at

Abstract. Predictive analysis in business process monitoring aims at forecasting the future information of a running business process. The prediction is typically made based on the model extracted from historical process execution logs (event logs). In practice, different business domains might require different kinds of predictions. Hence, it is important to have a means for properly specifying the desired prediction tasks, and a mechanism to deal with these various prediction tasks. Although there have been many studies in this area, they mostly focus on a specific prediction task. This work introduces a language for specifying the desired prediction tasks, and this language allows us to express various kinds of prediction tasks. This work also presents a mechanism for automatically creating the corresponding prediction model based on the given specification. Thus, different from previous studies, our approach enables us to deal with various kinds of prediction tasks based on the given specification. A prototype implementing our approach has been developed and experiments using a real-life event log have been conducted.

Keywords: Predictive business process monitoring
Prediction task specification · Automatic prediction model creation
Multi-perspective prediction

1 Introduction

Process mining [1] provides a collection of techniques for extracting process-related information from the logs of business process executions (event logs). One important area in this field is predictive business process monitoring, which aims at forecasting the future information of a running process based on the models extracted from event logs. Through predictive analysis, potential future problems can be detected and preventive actions can be taken in order to avoid unexpected situation (e.g., processing delay, SLA violations). Many techniques have been proposed for tackling various prediction tasks such as predicting the outcomes of a process [9,15,21,31], predicting the remaining processing time [2,23–25,30], predicting the future events [10,11,30], etc (cf. [5,8,11,17,18,22,27]).

In practice, different business areas might need different kinds of prediction tasks. For instance, an online retail company might be interested in predicting

J. Gulden et al. (Eds.): BPMDS 2018/EMMSAD 2018, LNBIP 318, pp. 97–113, 2018.
https://doi.org/10.1007/978-3-319-91704-7_7

the processing time until an order can be delivered to the customer, while for an insurance company, predicting the outcomes of an insurance claim process would be interesting. On the other hand, both of them might be interested in predicting whether their processes comply with some business constraints (e.g., the processing time must be finished within a certain amount of time).

When it comes to predicting the outcomes of a process or predicting an unexpected behaviour, it is important to specify the desired outcomes or the unexpected behaviour precisely. For instance, in the area of customer problem management, to increase customer satisfaction as well as to promote efficiency, we might be interested in predicting the possibility of "*ping-pong behaviour*" among the Customer Service (CS) officers while handling the customer problems. However, the definition of a ping-pong behaviour could be varied. For instance, when a CS officer transfers a customer problem into another CS officer who belongs to the same group, it can already be considered as a ping-pong behaviour since both of them should be able to handle the same problem. Another possible definition would be when a CS officer transfers a problem into another CS officer who has the same expertise, and the problem is transfered back into the original CS officer.

To have a suitable prediction service for our domain, we need to understand and specify the desired prediction tasks properly. Thus, we need a means to express the specification. Once we have characterized the prediction objectives and are able to express them properly, we need a mechanism to create the corresponding prediction model. To automate the prediction model creation, the specification should be machine processable. As illustrated above, such specification mechanism should also allow us to specify some constraints over the data, and compare some data values at different time points. For example, to characterize the ping-pong behaviour, one possibility is to specify the behaviour as follows: "*there is an event at a certain time point in which the CS officer is different with the CS officer in the event at the next time point, but both of them belong to the same group*". Note that here we need to compare the information about the CS officer names and groups at different time points.

In this work, we tackle those problems by providing the following contributions: *(i)* We introduce a rich language for expressing the desired prediction tasks. This language allows us to specify various kinds of prediction tasks. In some sense, this language also allows us to specify how to create the desired prediction models based on the event logs. *(ii)* We devise a mechanism for building the corresponding prediction model based on the given specification. Once created, the prediction model can be used to provide predictive analysis service in business process monitoring. *(iii)* We exhibit how our approach can be used for tackling various kinds of prediction tasks (cf. Section 3.3). *(iv)* We develop a prototype that implements our approach and enables the automatic creation of prediction models based on the specified prediction objective. *(v)* To demonstrate the applicability of our approach, we carry out experiments using a real-life event log that was provided for the BPI Challenge 2013 [29].

Roughly speaking, in our approach, we specify various desired prediction tasks by specifying how we want to map each (partial) business processes execution information into the expected predicted information. Based on this specification, we automatically train either classification or regression models that will serve as the prediction models. By specifying a set of desired prediction tasks, we can obtain multi-perspective prediction services that enable us to focus on various aspects and predict various information. Our approach is independent with respect to the classification/regression model that is used. In our implementation, to get the expected quality of predictions, the users are allowed to choose the desired classification/regression model as well as the feature encoding mechanisms (to allow some sort of feature engineering). Supplementary materials containing more explanations, examples and experiments are available at [26].

2 Preliminaries

This section provides some background concepts for the rest of the paper.

Trace, Event and Event Log. We follow the usual notion of event logs as in process mining [1]. An event log captures historical information about the execution of business processes. In an event log, each execution of a process is represented as a trace. Each trace has several events, and each event in the trace captures the information about a particular event that happens during the process execution. Events are characterized by various attributes, e.g., *timestamp* (the time at which the event occurred).

Let $\mathcal{E}$ be the *event universe* (i.e., the set of all event identifiers), and $\mathcal{A}$ be the set of *attribute names*. For any event $e \in \mathcal{E}$, and attribute name $n \in \mathcal{A}$, $\#_{\mathrm{n}}(e)$ denotes the *value of the attribute* n of e. E.g., $\#_{\mathrm{timestamp}}(e)$ denotes the timestamp of the event e. If an event e does not have an attribute named n, then $\#_{\mathrm{n}}(e) = \perp$ (undefined value). A *finite sequence over* $\mathcal{E}$ *of length* n is a mapping $\sigma : \{1, \ldots, n\} \to \mathcal{E}$, and such a sequence is represented as a tuple of elements of $\mathcal{E}$, i.e., $\sigma = \langle e_1, e_2, \ldots, e_n \rangle$ where $e_i = \sigma(i)$ for $i \in \{1, \ldots, n\}$. The set of all *finite sequences* over $\mathcal{E}$ is denoted by $\mathcal{E}^*$. The *length* of a sequence σ is denoted by $|\sigma|$.

A *trace* τ is a finite sequence over $\mathcal{E}$ such that each event $e \in \mathcal{E}$ occurs at most once in τ, i.e., $\tau \in \mathcal{E}^*$ and for $1 \leq i < j \leq |\tau|$, we have $\tau(i) \neq \tau(j)$, where $\tau(i)$ refers to the *event of the trace* τ *at the index* i. Let $\tau = \langle e_1, e_2, \ldots, e_n \rangle$ be a trace, $\tau^k = \langle e_1, e_2, \ldots, e_k \rangle$ denotes the *k-length prefix* of τ (for $0 < k < n$). For example, let $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7\} \subset \mathcal{E}$, $\tau = \langle e_3, e_7, e_6, e_4, e_5 \rangle \in \mathcal{E}^*$ is an example of a trace, $\tau(3) = e_6$, and $\tau^2 = \langle e_3, e_7 \rangle$. Finally, an *event log* L is a set of traces such that each event occurs at most once in the entire log, i.e., for each $\tau_1, \tau_2 \in L$ such that $\tau_1 \neq \tau_2$, we have that $\tau_1 \cap \tau_2 = \emptyset$, where $\tau_1 \cap \tau_2 = \{e \in \mathcal{E} \mid \exists i, j \in \mathbb{Z}^+ \,.\, \tau_1(i) = \tau_2(j) = e\}$.

An IEEE standard for representing event logs, called XES (eXtensible Event Stream), has been introduced in [13]. The standard defines the XML format for organizing the structure of traces, events and attributes in event logs. It also introduces some extensions that define some attributes with pre-defined meaning such as: *(i)* "*concept:name*", which stores the name of event/trace;

(ii) "*org:resource*", which stores the name/identifier of the resource that triggered the event (e.g., a person name); *(iii)* "*org:group*", which stores the group name of the resource that triggered the event.

Classification and Regression. In machine learning, a classification and regression model can be seen as a function $f : \vec{X} \rightarrow Y$ that takes some *input features/variables* $\vec{x} \in \vec{X}$ and predicts the corresponding *target value/output* $y \in Y$. The key difference is that the output range of the classification task is a finite number of discrete categories (qualitative outputs) while the output range of the regression task is continous values (quantitative outputs) [12]. Both of them are supervised machine learning techniques where the models are trained with labelled data. I.e., the inputs for the training are the pairs of input variables $\vec{x}$ and target value y. This way, the models learn how to map certain inputs $\vec{x}$ into the expected target value y.

3 Approach

Our approach for obtaining a predictive process monitoring service consists of the following main steps: *(i)* specify the desired prediction tasks and *(ii)* automatically create the prediction model based on the given specification. Once created, we can use the models to predict the future information. In the following, we elaborate these steps.

3.1 Specifying the Desired Prediction Tasks

This section explains the mechanism for specifying the desired prediction task. Here we introduce a language that is able to capture the desired prediction task in terms of the specification on how to map each (partial) trace in the event log into the desired prediction results. Such specification can be used to train a classification/regression model that will be used as the prediction model.

In our approach, the specification of a particular prediction task is specified as an analytic rule, where an *analytic rule* R is an expression of the form

$$R = \langle \mathsf{Cond}_1 \Longrightarrow \mathsf{Target}_1,\ \ldots,\ \mathsf{Cond}_n \Longrightarrow \mathsf{Target}_n,\ \mathsf{DefaultTarget} \rangle.$$

Each Cond_i in R is called *condition expression*, while Target_i and $\mathsf{DefaultTarget}$ are called *target expression* (for $i \in \{1, \ldots, n\}$). We explain and formalize how to specify a condition and target expression after providing some intuitions below.

An analytic rule R will be interpreted as a function that maps (partial) traces into the values obtained from evaluating the target expressions. The mapping is based on the condition that is satisfied by the corresponding trace. Let τ be a (partial) trace, such function R can be illustrated as follows (the formal definition will be given later):

$$R(\tau) = \begin{cases} \mathit{evaluate}(\mathsf{Target}_1) & \text{if } \tau \text{ satisfies } \mathsf{Cond}_1, \\ \vdots & \vdots \\ \mathit{evaluate}(\mathsf{Target}_n) & \text{if } \tau \text{ satisfies } \mathsf{Cond}_n, \\ \mathit{evaluate}(\mathsf{DefaultTarget}) & \text{otherwise} \end{cases}$$

We will see that a target expression essentially specifies the desired prediction result or expresses the way how to compute the desired prediction result. Thus, an analytic rule R can also be seen as a means to map (partial) traces into the desired prediction results, or to compute the expected prediction results of (partial) traces.

To specify a condition expression in analytic rules, we introduce a language called First-Order Event Expression (FOE). Roughly speaking, an FOE formula is a First-Order Logic (FOL) formula [28] where the atoms are expressions over some event attribute values and some comparison operators (e.g., $=$, $\neq$, $>$). Moreover, the quantification in FOE is restricted to the indices of events (so as to quantify the time points). The idea of condition expressions is to capture a certain property of (partial) traces. To give some intuition, before we formally define the language, consider the ping-pong behaviour that can be specified as follows:

$$\begin{aligned}\mathsf{Cond}_{\text{pp}} = \exists i.(\ \ & i > \mathsf{curr} \ \ \wedge\ \ \mathsf{e}[i]\text{. org:resource} \neq \mathsf{e}[i+1]\text{. org:resource} \ \wedge \\ & i+1 \leq \mathsf{last} \ \wedge\ \ \mathsf{e}[i]\text{. org:group} = \mathsf{e}[i+1]\text{. org:group})\end{aligned}$$

where "$\mathsf{e}[i+1]$. org:group" is an expression for getting the "org:group" attribute value of the event at the index $i+1$. The formula $\mathsf{Cond}_{\text{pp}}$ basically says that "*there exists a time point i that is bigger than the current time point (i.e., in the future), in which the resource (the person in charge) is different with the resource at the time point $i+1$ (i.e., the next time point), their groups are the same, and the next time point is still not later than the last time point*". As for the target expression, some simple examples would be some strings such as "Ping-Pong" and "Not Ping-Pong". Based on these, we can create an example of analytic rule

$$R_1 = \langle \mathsf{Cond}_{\text{pp}} \Longrightarrow \text{"Ping-Pong"},\ \text{"Not Ping-Pong"} \rangle,$$

where $\mathsf{Cond}_{\text{pp}}$ is as above. In this case, R_1 specifies a task for predicting the ping-pong behaviour. In the prediction model creation phase, we will create a classifier that classifies (partial) traces based on whether they satisfy $\mathsf{Cond}_{\text{pp}}$ or not. During the prediction phase, such classifier can be used to predict whether a given (partial) trace will lead into ping-pong behaviour or not.

The target expression can be more complex than merely a string. For instance, it can be an expression that involves arithmetic operations over numeric values such as

$$\mathsf{Target}_{\text{remainingTime}} = \mathsf{e}[\mathsf{last}]\text{. time:timestamp} - \mathsf{e}[\mathsf{curr}]\text{. time:timestamp},$$

which computes "*the time difference between the timestamp of the last event and the current event (i.e., remaining processing time)*". Then we can create an analytic rule

$$R_2 = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{Target}_{\text{remainingTime}},\ 0 \rangle,$$

which specifies a task for predicting the remaining time, because R_2 will map each (partial) trace into its remaining processing time. In this case, we will create

a regression model for predicting the remaining processing time of a given (partial) trace. Section 3.3 provides more examples of prediction tasks specification using our language.

Formalizing the Condition and Target Expressions. As we have seen in the examples above, we need to refer to a particular index of an event within a trace. To capture this, we introduce the notion of *index expression* idx defined as follows:

$$\mathsf{idx} ::= i \mid \mathit{pint} \mid \mathsf{last} \mid \mathsf{curr} \mid \mathsf{idx}_1 + \mathsf{idx}_2 \mid \mathsf{idx}_1 - \mathsf{idx}_2$$

where *(i)* i is an *index variable*. *(ii)* *pint* is a positive integer (i.e., $\mathit{pint} \in \mathbb{Z}^+$). *(iii)* last and curr are special indices in which the former refers to the index of the last event in a trace, and the latter refers to the index of the current event (i.e., last event of the trace prefix under consideration). For instance, given a k-length prefix τ^k of the trace τ, curr is equal to k (or $|\tau^k|$), and last is equal to $|\tau|$. *(iv)* $\mathsf{idx} + \mathsf{idx}$ and $\mathsf{idx} - \mathsf{idx}$ are the usual arithmetic addition and subtraction operation over indices.

The semantics of index expression is defined over k-length trace prefixes. Since an index expression can be a variable, given a k-length trace prefix τ^k of the trace τ, we first introduce a *variable valuation* ν, i.e., a mapping from index variables into $\mathbb{Z}^+$. Then, we assign meaning to index expression by associating to τ^k and ν an *interpretation function* $(\cdot)_\nu^{\tau^k}$ which maps an index expression into $\mathbb{Z}^+$. Formally, $(\cdot)_\nu^{\tau^k}$ is inductively defined as follows:

$$\begin{array}{lll} (i)_\nu^{\tau^k} = \nu(i) & (\mathsf{curr})_\nu^{\tau^k} = k & (\mathsf{idx}_1 + \mathsf{idx}_2)_\nu^{\tau^k} = (\mathsf{idx}_1)_\nu^{\tau^k} + (\mathsf{idx}_2)_\nu^{\tau^k} \\ (\mathit{pint})_\nu^{\tau^k} = \mathit{pint} \in \mathbb{Z}^+ & (\mathsf{last})_\nu^{\tau^k} = |\tau| & (\mathsf{idx}_1 - \mathsf{idx}_2)_\nu^{\tau^k} = (\mathsf{idx}_1)_\nu^{\tau^k} - (\mathsf{idx}_2)_\nu^{\tau^k} \end{array}$$

To access the value of an event attribute, we introduce *event attribute accessor*, which is an expression of the form

$$\mathsf{e}[\mathsf{idx}]\mathbf{.}\ \mathrm{attName}$$

where *attName* is an attribute name and idx is an index expression. To define the semantics of event attribute accessor, we extend the definition of our interpretation function $(\cdot)_\nu^{\tau^k}$ such that it interprets an event attribute accessor expression into the attribute value of the corresponding event at the given index. Formally, $(\cdot)_\nu^{\tau^k}$ is defined as follows:

$$(\mathsf{e}[\mathsf{idx}]\mathbf{.}\ \mathrm{attName})_\nu^{\tau^k} = \begin{cases} \#_{\mathrm{attName}}(e) & \text{if } (\mathsf{idx})_\nu^{\tau^k} = i,\ 1 \le i \le |\tau|, \text{ and } e = \tau(i) \\ \bot & \text{otherwise} \end{cases}$$

E.g., "$\mathsf{e}[i]$**.** org:resource" refers to the value of the attribute "org:resource" of the event at the position i.

The value of an event attribute can be either numeric (e.g., 26, 3.86) or non-numeric (e.g., "sendOrder"), and we might want to specify properties that involve arithmetic operations over numeric values. Thus, we introduce the notion of *numeric expression* and *non-numeric expression* as expressions defined as follows:

$$\begin{aligned} \mathsf{nonNumExp} &::= \mathsf{true} \mid \mathsf{false} \mid \mathsf{String} \mid \mathtt{e}[\mathsf{idx}].\,\text{NonNumericAttribute} \\ \mathsf{numExp} &::= \mathsf{number} \mid \mathsf{idx} \mid \mathtt{e}[\mathsf{idx}].\,\text{NumericAttribute} \\ &\mid \mathsf{numExp}_1 + \mathsf{numExp}_2 \mid \mathsf{numExp}_1 - \mathsf{numExp}_2 \end{aligned}$$

where *(i)* true and false are the usual boolean values, *(ii)* String is the usual string, *(iii)* number is real numbers, *(iv)* e[idx]. *NonNumericAttribute* (resp. e[idx]. *NumericAttribute*) is event attribute accessor for accessing an attribute with non-numeric values (resp. numeric values), *(v)* $\mathsf{numExp}_1 + \mathsf{numExp}_2$ and $\mathsf{numExp}_1 - \mathsf{numExp}_2$ are the usual arithmetic operations over numeric expressions.

To give the semantics for *numeric expression* and *non-numeric expression*, we extend the definition of our interpretation function $(\cdot)_\nu^{\tau^k}$ by interpreting true, false, String, and number as themselves (e.g., $(3)_\nu^{\tau^k} = 3$, ("sendOrder")$_\nu^{\tau^k}$ = "sendOrder"), and by interpreting the arithmetic operations as usual, i.e., for the addition operator we have

$$(\mathsf{numExp}_1 + \mathsf{numExp}_2)_\nu^{\tau^k} = (\mathsf{numExp}_1)_\nu^{\tau^k} + (\mathsf{numExp}_2)_\nu^{\tau^k}$$

The definition is similar for the subtraction operator. Note that the value of an event attribute might be undefined $\perp$. In this work, we define that the arithmetic operations involving $\perp$ give $\perp$ (e.g., $26 + \perp = \perp$).

We are now ready to specify the notion of *event expression* as follows:

$$\begin{aligned} \mathsf{eventExp} ::= {} & \mathsf{numExp}_1 \ \mathbf{acop}\ \mathsf{numExp}_2 \mid \mathsf{nonNumExp}_1 \ \mathbf{lcop}\ \mathsf{nonNumExp}_2 \\ & \mid \mathsf{eventExp}_1 \ \mathbf{lcop}\ \mathsf{eventExp}_2 \mid \mathsf{true} \mid \mathsf{false} \end{aligned}$$

where *(i)* **lcop** stands for a logical comparison operator ($=$ or $\neq$). *(ii)* **acop** stands for an arithmetic comparison operator ($<, >, \leq, \geq, =$ or $\neq$). We interpret each logical/arithmetic comparison operator as usual (e.g., $26 \geq 3$ is interpreted as true, "receivedOrder" = "sendOrder" is interpreted as false). It is easy to see how to extend the definition of our interpretation function $(\cdot)_\nu^{\tau^k}$ towards interpreting event expressions, therefore we omit the details.

Finally, we are ready to define the language for specifying condition expression, namely First-Order Event Expression (FOE). An FOE formula is a First Order Logic (FOL) formula where the atoms are event expressions and the quantification is ranging over event indices. Syntactically FOE is defined as follows:

$$\varphi ::= \mathsf{eventExp} \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \rightarrow \varphi_2 \mid \forall i.\varphi \mid \exists i.\varphi$$

Where eventExp is an event expression. The semantics of FOE constructs is based on the usual FOL semantics. Formally, given a k-length trace prefix τ^k of the trace τ, and index variables valuation ν, we extend the definition of our interpretation function $(\cdot)_\nu^{\tau^k}$ as follows[1]:

[1] We assume that variables are standardized apart, i.e., no two quantifiers bind the same variable (e.g., $\forall i.\exists i.(i > 3)$), and no variable occurs both free and bound (e.g., $(i > 5) \wedge \exists i.(i > 3)$). As usual in FOL, every FOE formula can be transformed into a semantically equivalent formula where the variables are standardized apart by applying some variable renaming [28].

$$
\begin{array}{lll}
(\neg\varphi)_{\nu}^{\tau^k} = \mathsf{true} & \text{if} & (\varphi)_{\nu}^{\tau^k} = \mathsf{false} \\
(\varphi_1 \wedge \varphi_2)_{\nu}^{\tau^k} = \mathsf{true} & \text{if} & (\varphi_1)_{\nu}^{\tau^k} = \mathsf{true}, \text{ and } (\varphi_2)_{\nu}^{\tau^k} = \mathsf{true} \\
(\exists i.\varphi)_{\nu}^{\tau^k} = \mathsf{true} & \text{if} & \text{for some } c \in \{1,\ldots,|\tau|\}, \text{ we have } (\varphi)_{\nu[i\mapsto c]}^{\tau^k} = \mathsf{true} \\
(\forall i.\varphi)_{\nu}^{\tau^k} = \mathsf{true} & \text{if} & \text{for every } c \in \{1,\ldots,|\tau|\}, \text{ we have that } (\varphi)_{\nu[i\mapsto c]}^{\tau^k} = \mathsf{true}
\end{array}
$$

where $\nu[i \mapsto c]$ stands for a new index variable valuation obtained from ν as follows:

$$\nu[i \mapsto c](x) = \begin{cases} c & \text{if } x = i \\ \nu(x) & \text{if } x \neq i \end{cases}$$

Intuitively, $\nu[i \mapsto c]$ substitutes each variable i with c, while the other variables are substituted the same way as ν is defined. The semantics of $\varphi_1 \vee \varphi_2$ and $\varphi_1 \rightarrow \varphi_2$ is as usual in FOL. When φ is a closed formula, its truth value does not depend on the valuation for the index variables, and we denote the interpretation of φ simply by $(\varphi)^{\tau^k}$. We also say that τ^k *satisfies* φ, written $\tau^k \models \varphi$, if $(\varphi)^{\tau^k} = \mathsf{true}$.

Finally, the condition expression in analytic rules is specified as closed FOE formulas, while the target expression is specified as either numeric expression or non-numeric expression, except that target expressions are not allowed to have index variables (Thus, they do not need variable valuation).

Essentially, FOE has the following main features: *(i)* it allows us to specify constraints over the data; *(ii)* it allows us to (universally/existentially) quantify different event time points and to compare different event attribute values at different event time points; *(iii)* it supports arithmetic expressions/operations over the data.

Checking Whether a Condition Expression is Satisfied. Given a k-length trace prefix τ^k of the trace τ, and a condition expression φ (which is expressed as an FOE formula), to explain how to check whether $\tau^k \models \varphi$, we first introduce some properties of FOE formula below. Let φ be an FOE formula, we write $\varphi[i \mapsto c]$ to denote a new formula obtained by substituting each variable i in φ by c.

Theorem 1. *Given an FOE formula $\exists i.\varphi$, and a k-length trace prefix τ^k of the trace τ,*

$$\tau^k \models \exists i.\varphi \;\; \textit{iff} \;\; \tau^k \models \bigvee_{c\in\{1,\ldots|\tau|\}} \varphi[i \mapsto c]$$

Proof (sketch). By the semantics definition, τ^k satisfies $\exists i.\varphi$ iff there exists an index $c \in \{1,\ldots,|\tau|\}$, such that τ^ksatisfies the formula ψ that is obtained from φ by substituting each variable i in φ with c. Thus, it is the same as satisfying the disjunction of formulas that is obtained by considering all possible substitutions of the variable i in φ (i.e., $\bigvee_{c\in\{1,\ldots|\tau|\}} \varphi[i \mapsto c]$). This is the case because such disjunction of formulas will be satisfied by τ^k when there is a formula in the disjunction that is satisfied by τ^k. □

Theorem 2. *Given an FOE formula $\forall i.\varphi$, and a k-length trace prefix τ^k of the trace τ,*

$$\tau^k \models \forall i.\varphi \quad \textit{iff} \quad \tau^k \models \bigwedge_{c \in \{1,\ldots|\tau|\}} \varphi[i \mapsto c]$$

Proof (sketch). Similar to Theorem 1, except that we use conjunctions of formulas. □

To check whether $\tau^k \models \varphi$, we perform the following three steps: (1) Eliminate all quantifiers. This can be easily done by applying Theorems 1 and 2. As a result, each variable will be instantiated with a concrete value. (2) Evaluate each event attribute accessor expression based on the event attributes in τ. From this step, we will have a formula which is constituted by only concrete values composed by logical/arithmetic/comparison operators. (3) Last, we evaluate all logical, arithmetic and comparison operators.

Formalizing the Analytic Rule. With this machinery in hand, now we can formalize the semantics of analytic rules as introduced above. Formally, given an analytic rule

$$R = \langle \mathsf{Cond}_1 \Longrightarrow \mathsf{Target}_1, \ \ldots, \ \mathsf{Cond}_n \Longrightarrow \mathsf{Target}_n, \ \mathsf{DefaultTarget} \rangle.$$

R is interpreted as a function that maps (partial) traces into the values obtained from evaluating the target expressions defined below

$$R(\tau^k) = \begin{cases} (\mathsf{Target}_1)^{\tau^k} & \text{if } \ \tau^k \models \mathsf{Cond}_1, \\ \vdots & \vdots \\ (\mathsf{Target}_n)^{\tau^k} & \text{if } \ \tau^k \models \mathsf{Cond}_n, \\ (\mathsf{DefaultTarget})^{\tau^k} & \text{otherwise} \end{cases}$$

where τ^k is k-length trace prefix of the trace τ, and recall that $(\mathsf{Target}_i)^{\tau^k}$ is the application our interpretation function $(\cdot)^{\tau^k}$ to the target expression Target_i in order to evaluate the expression and get the value. Checking whether $\tau^k \models \mathsf{Cond}_i$ can be done as explained above. We also require that an analytic rule to be *coherent*, i.e., all target expressions of an analytic rule should be either only numeric or non-numeric expressions. An analytic rule in which all of its target expressions are numeric expressions is called *numeric analytic rule*, while an analytic rule in which all of its target expressions are non-numeric expressions is called *non-numeric analytic rule*.

Given a k-length trace prefix τ^k and an analytic rule R, we say that R *is well-defined for* τ^k if R maps τ^k into exactly one target value, i.e., for every condition expressions Cond_i and Cond_j in which $\tau^k \models \mathsf{Cond}_i$ and $\tau^k \models \mathsf{Cond}_j$, we have that $(\mathsf{Target}_i)^{\tau^k} = (\mathsf{Target}_j)^{\tau^k}$. The notion of well-defined can be generalized to event logs. Given an event log L and an analytic rule R, we say that R *is well-defined for* L if for each possible k-length trace prefix τ^k of each trace τ in L, we have that R is well-defined for τ^k. This condition can be easily checked for the given event log L and an analytic rule R.

Note that our notion of well-defined is more relaxed than requiring that each condition must not be overlapped, and this gives flexibility for making a specification using our language. For instance, one can specify several

characteristics of ping-pong behaviour in a more convenient way by specifying several conditional-target rules (i.e., $\mathsf{Cond}_1 \implies$ "Ping-Pong", $\mathsf{Cond}_2 \implies$ "Ping-Pong", ...) instead of using disjunctions of these several characteristics. From now on we only consider the analytic rules that are coherent and well-defined for the event logs under consideration.

3.2 Building the Prediction Model

Given an analytic rule R and an event log L, if R is a numeric analytic rule, we build a regression model. Otherwise, if R is a non-numeric analytic rule, we build a classification model. Note that our aim is to create a prediction function that takes (partial) traces as inputs. Thus, we train a classification/regression function in which the inputs are the features obtained from the encoding of trace prefixes in the event log L (the training data). There are several ways to encode (partial) traces into input features for training a machine learning model. For instance, [14] studies various encoding techniques such as index-based encoding, boolean encoding, etc. In [30], the authors use the so-called *one-hot encoding* of event names, and also add some time features (e.g., the time increase with respect to the previous event). In general, an encoding technique can be seen as a function enc that takes a trace τ as the input and produces a set $\{x_1, \ldots, x_m\}$ of features (i.e., $\mathsf{enc}(\tau) = \{x_1, \ldots, x_m\}$).

In our approach, users are allowed to choose the desired encoding mechanism by specifying a set Enc of preferred encoding functions (i.e., $\mathsf{Enc} = \{\mathsf{enc}_1, \ldots, \mathsf{enc}_n\}$). This allows us to do some sort of feature engineering (note that the desired feature engineering approach, that might help increasing the prediction performance, can also be added as one of these encoding functions). The set of features of a trace is then obtained by combining all features produced by applying each of the selected encoding functions into the corresponding trace. In the implementation (cf. Sect. 4), we provide some encoding functions that can be selected in order to encode a trace.

The procedure for creating the prediction model takes the following three inputs: *(i)* an analytic rule R; *(ii)* an event log L; and *(iii)* a set $\mathsf{Enc} = \{\mathsf{enc}_1, \ldots, \mathsf{enc}_n\}$ of encoding functions. The steps for creating the prediction model are as follows: (1) for each k-length trace prefix τ^k of each trace τ in the event log L (where $k \in \{2, \ldots, |\tau|\}$), we do the following three steps: *(i)* we apply each encoding function $\mathsf{enc}_i \in \mathsf{Enc}$ into τ^k, and combine all obtained features (This step gives us the encoded trace prefix τ^k_{encoded}); *(ii)* we compute the expected prediction result (target value) by applying the analytical rule R to τ^k (i.e., the target value is equal to $R(\tau^k)$); *(iii)* we add a new training instance by specifying that the prediction function $\mathcal{P}$ maps the encoded trace prefix τ^k_{encoded} into the target value computed in the previous step. (2) Finally, after processing each k-length trace prefix of each trace in the event log as in the step 1, we train the prediction function $\mathcal{P}$ based on the training instances obtained from the step 1 and get the desired prediction function. A more formal explanation of this procedure can be seen in [26].

3.3 Showcase of Our Approach: Multi-perspective Predictive Analysis Service

An analytic rule R specifies a particular prediction task of interest. To specify several desired prediction tasks, we only have to specify several analytic rules, i.e., $R_1, \ldots, R_2$. Given a set $\mathcal{R}$ of analytic rules, i.e., $\mathcal{R} = \{R_1, \ldots, R_2\}$, our approach allows us to construct a prediction model for each analytic rule $R \in \mathcal{R}$. This way, we can get a *multi-perspective prediction analysis service* provided by all of the constructed prediction models where each of them focus on a particular prediction objective.

In Sect. 3.1 we have seen some examples of prediction task specification for predicting the ping-pong behaviour and the remaining processing time. In the following, we show other examples of specifying prediction task using our language.

Predicting Unexpected Behaviour. We can specify a task for predicting unexpected behaviour by first expressing the characteristics of the unexpected behaviour. The condition expression $\mathsf{Cond}_{\text{pp}}$ (in Sect. 3.1) expresses a possible characteristic of ping-pong behaviour. Another possible characterization of this behaviour is shown below:

$$\begin{aligned}\mathsf{Cond}_{\text{pp2}} = \exists i.(\quad & i > \mathsf{curr} \ \wedge \ \mathsf{e}[i]\text{. org:resource} \neq \mathsf{e}[i+1]\text{. org:resource} \ \wedge \\ & i+1 \leq \mathsf{last} \ \wedge \ \mathsf{e}[i]\text{. org:resource} = \mathsf{e}[i+2]\text{. org:resource} \ \wedge \\ & i+2 \leq \mathsf{last} \ \wedge \ \mathsf{e}[i]\text{. org:group} \quad = \mathsf{e}[i+1]\text{. org:group} \\ & \wedge \ \mathsf{e}[i]\text{. org:group} \quad = \mathsf{e}[i+2]\text{. org:group})\end{aligned}$$

essentially, $\mathsf{Cond}_{\text{pp2}}$ characterizes the condition where "*an officer transfers a task into another officer of the same group, and then the task is transfered back into the original officer*". In the event log, this situation is captured by the changes of the org:resource value in the next event, but then it changes back into the original value in the next two events, while the values of org:group remain the same. We can then specify an analytic rule for specifying the ping-pong behaviour prediction task as follows:

$$R_3 = \langle \mathsf{Cond}_{\text{pp}} \implies \text{"Ping-Pong"}, \ \mathsf{Cond}_{\text{pp2}} \implies \text{"Ping-Pong"}, \ \text{"Not Ping-Pong"} \rangle.$$

During the training phase, R_3 maps each trace prefix τ^k that satisfies either $\mathsf{Cond}_{\text{pp}}$ or $\mathsf{Cond}_{\text{pp2}}$ into the target value "Ping-Pong", and those prefixes that neither satisfy $\mathsf{Cond}_{\text{pp}}$ nor $\mathsf{Cond}_{\text{pp2}}$ into "Not Ping-Pong". After the training based on this rule, we get a classifier that is trained for distinguishing between (partial) traces that will and will not lead into ping-pong behaviour. This example also exhibits the ability of our language to specify a behaviour that has multiple characteristics.

Predicting Next Event. The task for predicting the next event is specified as follows: $R_4 = \langle \mathsf{curr} + 1 \leq \mathsf{last} \implies \mathsf{e}[\mathsf{curr} + 1]\text{. concept:name}, \ \bot \rangle$. In the training phase, R_4 maps each k-length trace prefix τ^k into its next event name, because "$\mathsf{e}[\mathsf{curr} + 1]$. concept:name" is evaluated into the name of the event at the index $\mathsf{curr} + 1$ (i.e., $|\tau^k|+1$). If $k = |\tau|$, then R_4 maps τ^k into $\bot$ (undefined).

After the training, we get a classifier that is trained to give the next event name of the given (partial) trace.

Predicting the Next Event Timestamp. This task can be specified as follows:[2]

$$R_5 = \langle \mathsf{curr} + 1 \leq \mathsf{last} \implies \mathsf{e}[\mathsf{curr} + 1].\ \text{time:timestamp},\ \bot \rangle.$$

R_5 maps each k-length trace prefix τ^k into the next event timestamp. Hence, we train a regression model that outputs the next event timestamp of the given (partial) trace.

Predicting SLA/Business Constraints Compliance. Using FOE, we can easily specify expressive SLA conditions/business constraints, and automatically create the corresponding prediction model using our approach. E.g., we can specify a constraint:

$$\begin{aligned}\forall\, i.(\mathsf{e}[i].\ \text{concept:name} = \text{“OrderCreated”} \ \rightarrow\ \exists\, j.(j > i\ \wedge \\ \mathsf{e}[j].\ \text{concept:name} = \text{“OrderDelivered”}\ \wedge\ \mathsf{e}[i].\ \text{orderID} = \mathsf{e}[j].\ \text{orderID}\ \wedge \\ (\mathsf{e}[j].\ \text{time:timestamp} - \mathsf{e}[i].\ \text{time:timestamp}) < 10.800.000))\end{aligned}$$

which essentially says “*whenever there is an event where an order is created, eventually there will be an event where the order is delivered and the time difference between the two events (the processing time) is less than* 10.800.000 ms (3 h)”.

4 Implementation and Experiment

As a proof of concept, by using Java and WEKA, we have implemented a prototype[3] that is also a ProM[4] plug-in. The prototype includes a parser for our language and a program for automatically processing the specification as well as building the corresponding prediction model based on the approach explained in Sects. 3.1 and 3.2. We also provide several feature encoding functions to be selected such as one hot encoding of attributes, time since the previous event, time since midnight, attribute values encoding, etc. We can also choose the desired machine learning model to be built.

Our experiments aim at showing the applicability of our approach in automatically constructing reliable prediction models based on the given specification. The experiments were conducted using the real life event log from BPI Challenge 2013 (BPIC 13) [29]. For the experiment, we use the first 2/3 of the log for the training and the last 1/3 of the log for the testing. In BPIC 13, the ping-pong behaviour among support teams is one of the problems to be analyzed. Ideally a customer problem should be solved without involving too many

[2] Note that timestamp can be represented as milliseconds since epoch (hence, it is a number).

[3] More information about the implementation architecture, the code, the tool, and the screencast can be found at http://bit.ly/predictive-analysis.

[4] ProM is an extendable framework for process mining (http://www.promtools.org).

support teams. Here we specify a prediction task for predicting the ping-pong behaviour by first characterizing a ping-pong behaviour among support teams as follows:

$$\begin{aligned}\mathsf{Cond}_{\text{ppteam}} = \exists i.(\quad & i > \mathsf{curr} \quad \wedge \ \mathsf{e}[i]\text{. org:group} \neq \mathsf{e}[i+1]\text{. org:group} \ \wedge \\ & i+1 \leq \mathsf{last} \ \wedge \ \mathsf{e}[i]\text{. concept:name} \neq \text{“Queued”})\end{aligned}$$

Roughly, $\mathsf{Cond}_{\text{ppteam}}$ says that *there is a change in the support team while the problem is not being "Queued"*. We then specify the following analytic rule:

$$R_{ex1} = \langle \mathsf{Cond}_{\text{ppteam}} \implies \text{“Ping-Pong”}, \ \text{“Not Ping-Pong”} \rangle$$

that can be fed into our tool for obtaining the prediction model. For this case, we automatically generate Decision Tree and Random Forest models from that specification. We also predict the time until the next event by specifying the following analytic rule:

$$R_{ex2} = \langle \mathsf{curr} + 1 \leq \mathsf{last} \implies \mathsf{e}[\mathsf{curr}+1]\text{. time:timestamp} - \mathsf{e}[\mathsf{curr}]\text{. time:timestamp}, 0 \rangle$$

For this case, we automatically generate Linear Regression and Random Forest models.

We evaluate the prediction performance of each k-length prefix τ^k of each trace τ in the testing set (for $2 \leq k < |\tau|$). We use accuracy and AUC (Area Under the ROC Curve) [12] values as the metrics to evaluate the ping-pong prediction. For the prediction of the time until the next event, we use MAE (Mean Absolute Error) [12], and RMSE (Root Mean Square Error) [12] values as the metrics, and we also provide the MAE and RMSE values for the mean-based prediction (i.e., the basic approach where the prediction is based on the mean of the target values in the training data). The results are summarized in Tables 1 and 2. We highlight the evaluation for several prediction points, namely *(i)* early prediction (at the 1/4 of the trace length), *(ii)* intermediate prediction (at the 1/2 of the trace length), and *(iii)* late prediction (at the 3/4 of the trace length). The column "All" presents the aggregate evaluation for all k-length prefix where $2 \leq k < |\tau|$.

Table 1. The evaluation of predicting ping-pong behaviour among support teams

	Accuracy				AUC value			
	Early	Mid	Late	All	Early	Mid	Late	All
Decision Tree	0.82	0.67	0.87	0.77	0.76	0.69	0.63	0.75
Random Forest	0.83	0.73	0.91	0.83	0.89	0.73	0.78	0.87

The AUC values in Table 1 show that our approach is able to automatically produce reasonable prediction models (The AUC values > 0.5). Table 2 shows that all of the automatically generated models perform better than the mean-based prediction (the baseline). The experiment also exhibits that the performance of

Table 2. The evaluation of predicting the time until the next event

	MAE (in days)				RMSE (in days)			
	Early	Mid	Late	All	Early	Mid	Late	All
Linear Regression	0.70	1.42	2.64	2.07	1.04	1.87	2.99	2.77
Random Forest	0.34	1.07	1.81	1.51	1.03	2.33	2.89	2.61
Mean-based Prediction	2.42	2.33	2.87	2.70	2.44	2.40	3.16	2.90

our approach depends on the machine learning model that is generated (e.g., in Table 1, random forest performs better than decision tree). Since our approach does not rely on a particular machine learning model, it justifies that we can simply plug in different supervised machine learning techniques in order to get different/better performance. In the future we plan to experiment with deep learning approach in order to get a better accuracy. As reported by [30], the usage of LSTM neural networks could improve the accuracy of some prediction tasks. More experiments can be seen in our supplementary materials (cf. [26]).

5 Related Work

This work is related to the area of predictive analysis in business process management. In the literature, there have been several works focusing on predicting time-related properties of running processes. For instance, the works in [2,23–25] focus on predicting the remaining processing time. The works by [18,22,27] focus on predicting delays in process execution. The authors of [30] present a deep learning approach for predicting the timestamp of the next event and use it to predict the remaining cycle time. Looking at another perspective, the works by [9,15,31] focus on predicting the outcomes of a running process. The work by [15] introduces a framework for predicting the business constraints compliance of a running process. In [15], the business constraints are formulated in propositional Linear Temporal Logic (LTL), where the atomic propositions are all possible events during the process executions. Another work on outcomes prediction is presented by [21], which proposes an approach for predicting aggregate process outcomes by also taking into account the evaluation of process risk. Related to process risks, [8] proposes an approach for risks prediction. Another stream of works tackle the problem of predicting the future events of a running process (cf. [5,10,11,24,30]).

A key difference between those works and ours is that, instead of focusing on a specific prediction task, this work enables us to specify and focus on various prediction tasks. To deal with these various desired prediction tasks, we also present a mechanism that can automatically build the corresponding prediction models based on the given specification of prediction tasks.

This work is also related to the works on devising specification language. Unlike the propositional LTL, which is the basis of Declare language [20] and typically used for specifying business constraints over sequence of events (cf.

[15]), our FOE language (which is part of our rule-based specification language) allows us not only to specify properties over sequence of events but also to specify properties over the data (attribute values) of the events. Concerning data-aware specification language, the work by [3] introduces a data-aware specification language by combining data querying mechanisms and temporal logic. Such language has been used in verification of data-aware processes systems (cf. [4,6,7]). The works by [16] enrich the Declare language with data conditions based on First-Order LTL (LTL-FO). Although those languages are data-aware, they do not support arithmetic expressions/operations over the data which is absolutely needed, e.g., for expressing the time difference between the timestamp of the first and the last event. Another interesting data-aware language is S-FEEL, which is part of the Decision Model and Notation (DMN) standard [19] by OMG. Though S-FEEL supports arithmetic expressions over the data, it does not allow us to (universally/existentially) quantify different event time points and to compare different event attribute values at different event time points, which is needed, e.g., in the ping-pong behaviour.

6 Conclusion

We have introduced a mechanism for specifying the desired prediction tasks by using a rule-based language, and for automatically creating the corresponding prediction models based on the given specification. A prototype of ProM plug-in that implements our approach has been developed and several experiments using a real life event log confirmed the applicability of our approach.

Future work includes the extension of the tool and the language. One possible extension would be to incorporate aggregate functions such as `SUM` and `CONCAT`. These functions enable us to specify more tasks such as the prediction of total cost that is based on the sum of the cost attributes in all events. The `CONCAT` function could allow us to specify the prediction of the next sequence of activities by concatenating all next activities. Experimenting with other supervised machine learning techniques would be the next step as well, e.g., using deep learning approach in order to improve accuracy.

Acknowledgement. This research has been supported by the Euregio IPN12 "*KAOS: Knowledge-Aware Operational Support*" project, which is funded by the "European Region Tyrol-South Tyrol-Trentino" (EGTC) under the first call for basic research projects. The author thanks Tri Kurniawan Wijaya for various suggestions related to this work, and Yasmin Khairina for the implementation of some prototype components.

References

1. van der Aalst, W.M.P.: Process Mining. Data Science in Action, 2nd edn. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
2. van der Aalst, W.M.P., Schonenberg, M., Song, M.: Time prediction based on process mining. Inf. Syst. **36**(2), 450–475 (2011)

3. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., Deutsch, A., Montali, M.: Verification of relational data-centric dynamic systems with external services. In: the 32nd ACM SIGACT SIGMOD SIGAI Symposium on Principles of Database Systems (PODS), pp. 163–174 (2013)
4. Bagheri Hariri, B., Calvanese, D., Montali, M., Santoso, A., Solomakhin, D.: Verification of semantically-enhanced artifact systems. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 600–607. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45005-1_51
5. Breuker, D., Matzner, M., Delfmann, P., Becker, J.: Comprehensible predictive models for business processes. MIS Q. **40**(4), 1009–1034 (2016)
6. Calvanese, D., Ceylan, İİ., Montali, M., Santoso, A.: Verification of context-sensitive knowledge and action bases. In: Fermé, E., Leite, J. (eds.) JELIA 2014. LNCS (LNAI), vol. 8761, pp. 514–528. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11558-0_36
7. Calvanese, D., Montali, M., Santoso, A.: Verification of generalized inconsistency-aware knowledge and action bases (extended version). CoRR Technical report arXiv:1504.08108, arXiv.org e-Print archive (2015). http://arxiv.org/abs/1504.08108
8. Conforti, R., de Leoni, M., La Rosa, M., van der Aalst, W.M., ter Hofstede, A.H.: A recommendation system for predicting risks across multiple business process instances. Decis. Support Syst. **69**, 1–19 (2015)
9. Di Francescomarino, C., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. IEEE Trans. Serv. Comput. **PP**(99), 1–18 (2016)
10. Di Francescomarino, C., Ghidini, C., Maggi, F.M., Petrucci, G., Yeshchenko, A.: An eye into the future: leveraging a-priori knowledge in predictive business process monitoring. In: Carmona, J., Engels, G., Kumar, A. (eds.) BPM 2017. LNCS, vol. 10445, pp. 252–268. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65000-5_15
11. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decis. Support Syst. **100**, 129–140 (2017)
12. Friedman, J., Hastie, T., Tibshirani, R.: The Elements of Statistical Learning. Springer, New York (2001). https://doi.org/10.1007/978-0-387-21606-5
13. IEEE Comp. Intelligence Society: IEEE Standard for eXtensible Event Stream (XES) for achieving interoperability in event logs and event streams. IEEE Std 1849–2016 (2016)
14. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 297–313. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_21
15. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 457–472. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07881-6_31
16. Maggi, F.M., Dumas, M., García-Bañuelos, L., Montali, M.: Discovering data-aware declarative process models from event logs. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 81–96. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40176-3_8

17. Metzger, A., Leitner, P., Ivanović, D., Schmieders, E., Franklin, R., Carro, M., Dustdar, S., Pohl, K.: Comparing and combining predictive business process monitoring techniques. IEEE Trans. Syst. Man Cybern. Syst. **45**(2), 276–290 (2015)
18. Metzger, A., Franklin, R., Engel, Y.: Predictive monitoring of heterogeneous service-oriented business networks: the transport and logistics case. In: Annual SRII Global Conference (2012)
19. Object Management Group: Decision Model and Notation (DMN) 1.0 (2015). http://www.omg.org/spec/DMN/1.0/
20. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: Eder, J., Dustdar, S. (eds.) BPM 2006. LNCS, vol. 4103, pp. 169–180. Springer, Heidelberg (2006). https://doi.org/10.1007/11837862_18
21. Pika, A., van der Aalst, W., Wynn, M., Fidge, C., ter Hofstede, A.: Evaluating and predicting overall process risk using event logs. Inf. Sci. **352–353**, 98–120 (2016)
22. Pika, A., van der Aalst, W.M.P., Fidge, C.J., ter Hofstede, A.H.M., Wynn, M.T.: Predicting deadline transgressions using event logs. In: La Rosa, M., Soffer, P. (eds.) BPM 2012. LNBIP, vol. 132, pp. 211–216. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36285-9_22
23. Polato, M., Sperduti, A., Burattin, A., de Leoni, M.: Data-aware remaining time prediction of business process instances. In: 2014 International Joint Conference on Neural Networks (IJCNN) (2014)
24. Polato, M., Sperduti, A., Burattin, A., de Leoni, M.: Time and Activity Sequence Prediction of Business Process Instances. CoRR abs/1602.07566 (2016)
25. Rogge-Solti, A., Weske, M.: Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 389–403. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45005-1_27
26. Santoso, A.: Specification-driven multi-perspective predictive business process monitoring (extended version). CoRR Technical Report arXiv:1804.00617, arXiv.org e-Print archive (2018). https://arxiv.org/abs/1804.00617
27. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining – predicting delays in service processes. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 42–57. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07881-6_4
28. Smullyan, R.M.: First Order Logic. Springer, Heidelberg (1968). https://doi.org/10.1007/978-3-642-86718-7
29. Steeman, W.: BPI challenge 2013 (2013). https://doi.org/10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07
30. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 477–492. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_30
31. Verenich, I., Dumas, M., La Rosa, M., Maggi, F.M., Di Francescomarino, C.: Complex symbolic sequence clustering and multiple classifiers for predictive process monitoring. In: Reichert, M., Reijers, H.A. (eds.) BPM 2015. LNBIP, vol. 256, pp. 218–229. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42887-1_18