Software and Systems Modeling (2020) 19:1307–1343
https://doi.org/10.1007/s10270-019-00761-w

SPECIAL SECTION PAPER

Specification-driven predictive business process monitoring

Ario Santoso[1,2] · Michael Felderer[1,3]

Received: 30 November 2018 / Accepted: 5 October 2019 / Published online: 2 November 2019

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. Differently from previous studies, instead of focusing on a particular prediction task, we present an approach to deal with various prediction tasks based on the given specification of the desired prediction tasks. We also provide an implementation of the approach which is used to conduct experiments using real-life event logs.

Keywords Predictive business process monitoring · Prediction task specification language · Automatic prediction model creation · Machine learning-based prediction

1 Introduction

Process mining [66,67] 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 and service-level agreement (SLA) violations. Many studies have been conducted in order to deal with various prediction tasks such as predicting the remaining processing time [52–54,63,69], predicting the outcomes of a process [18,35,50,72], and predicting future events [19,24,63] (cf. [11,15,41,42,49,58]). An overview of various works in the area of predictive business process monitoring can be found in [20,36].

In practice, different business areas might need different kinds of prediction tasks. For instance, an online retail company might be interested in predicting the processing time until an order can be delivered to the customer, while for an insurance company, predicting the outcome 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 less than a certain amount of time).

When it comes to predicting the outcome of a process, business constraint satisfaction, and the existence of an unexpected behaviour, it is important to specify the desired outcomes, the business constraint, and the unexpected behaviour precisely. For instance, in the area of customer problem man-

Communicated by Dr. Rainer Schmidt and Jens Gulden.

This research has been supported by the Euregio Interregional Project Network IPN12 "*Knowledge-Aware Operational Support*" (KAOS), which is funded by the "European Region Tyrol-South Tyrol-Trentino" (EGTC) under the first call for basic research projects.

✉ Ario Santoso
ario.santoso@uibk.ac.at; santoso.ario@gmail.com

Michael Felderer
michael.felderer@uibk.ac.at

1 Department of Computer Science, University of Innsbruck, Innsbruck, Austria

2 Faculty of Computer Science, Free University of Bozen-Bolzano, Bozen-Bolzano, Italy

3 Department of Software Engineering, Blekinge Institute of Technology, Karlskrona, Sweden

agement, to increase the 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 to another CS officer who belongs into 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 to consider a ping-pong behaviour as a situation when a CS officer transfers a problem to another CS officer who has the same expertise, and the problem is transferred back to the original CS officer.

To have a suitable prediction service for our domain, we need to be able to 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 unambiguous and machine processable. Such specification mechanism should also allow us to specify constraints over the data, and compare 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 (who handles the problem) is different from 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 other cases, we might even need to involve arithmetic expressions. For instance, consider a business constraint that requires that the length of customer order processing time to be less than 3 hours, where the length of the processing time is the time difference between the timestamp of the first activity and the last activity within the process. To express this constraint, we need to be able to specify that "*the time difference between the timestamp of the first activity and the last activity within the process is less than 3 hours*".

The language should also enable us to specify the target information to be predicted. For instance, in the prediction of remaining processing time, we need to be able to define that the remaining processing time is *the time difference between timestamp of the last activity and the current activity*. We might also need to aggregate some data values, for instance, in the prediction of the total processing cost where the total cost is *the sum over the cost of all activities/events*. In other cases, we might even need to specify an expression that counts the number of a certain activity. For example, in the prediction of the amount of work to be done (workload), we might be interested in predicting the *number of the remaining validation activities* that are necessary to be done for processing a client application.

In this work, we tackle those problems by proposing an approach for obtaining the desired prediction services based on the specification of the desired prediction tasks. This work answers the following questions:

RQ1: How can a specification-driven mechanism for building prediction models for predictive process monitoring look like?

RQ2: How can an expressive specification language that allows us to express various desired prediction tasks, and at the same time enables us to automatically create the corresponding prediction model from the given specification, look like? Additionally, can that language allow us to specify complex expressions involving data, arithmetic operations and aggregate functions?

RQ3: Once we are able to specify the desired prediction tasks properly, how can a mechanism to automatically build the corresponding prediction model based on the given specification look like?

To answer these questions, we aim at introducing a rich language for expressing the desired prediction tasks, as well as a mechanism to process the specification in order to build the corresponding prediction model. Specifically, in this work, we provide the following contributions:

1. We introduce a rich language that allows us to specify various desired prediction tasks. In some sense, this language allows us to specify how to create the desired prediction models based on the event logs. We also provide a formal semantics for the language in order to ensure a uniform understanding and avoid ambiguity.
2. We devise a mechanism for building the corresponding prediction model based on the given specification. This includes the mechanism for automatically processing the specification. Once created, the prediction model can be used to provide predictive analysis services in business process monitoring.
3. To provide a general idea on the capability of our language, we exhibit how our proposal can be used for specifying various prediction tasks (cf.Sect. 5).
4. We provide an implementation of our approach which enables the automatic creation of prediction models based on the specified prediction objective.
5. To demonstrate the applicability of our approach, we carry out experiments using real-life event logs that were provided for the Business Process Intelligence Challenge (BPIC) 2012, 2013, and 2015.

Figure 1 illustrates our approach for obtaining prediction services. Essentially, it consists of the following main steps: *(i)* First, we specify the desired prediction tasks, *(ii)* sec-

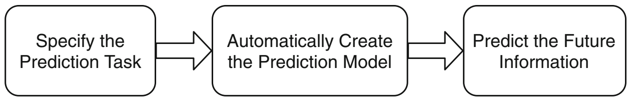
Specify the Prediction Task
Automatically Create the Prediction Model
Predict the Future Information

Fig. 1 Illustration of our approach for obtaining the prediction services

ond, we automatically create the prediction models based on the given specification, *(iii)* once created, we can use the constructed prediction models for predicting the future information of a running process.

Roughly speaking, we specify the desired prediction task by specifying how we want to map each (partial) business processes execution information into the expected predicted information. Based on this specification, we train either a classification or regression model that will serve as the prediction model. By specifying a set of desired prediction tasks, we could obtain *multi-perspective prediction services* that enable us to focus on different aspects and predict various information of interest. 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 (in order to allow some sort of feature engineering).

This paper extends [57] in several ways. First, we extend the specification language so as to incorporate various aggregate functions such as Max, Min, Average, Sum, Count, and Concat. Importantly, our aggregate functions allow us not only to perform aggregation over some values but also to choose the values to be aggregated. Obviously, this extension increases the expressivity of the language and allows us to specify many more interesting prediction tasks. Next, we add various new showcases that exhibit the capabilities of our language in specifying prediction tasks. We also extend the implementation of our prototype in order to incorporate those extensions. To demonstrate the applicability of our approach, more experiments on different prediction tasks are also conducted and presented. Apart from using the real-life event log that was provided for BPIC 2013 [62], we also use another real-life event logs, namely the event logs that were provided for BPIC 2012 [70] and BPIC 2015 [71]. Notably, our experiments also exhibit the usage of a deep Learning approach [30] in predictive process monitoring. In particular, we use deep feed-forward neural network. Though there have been some works that exhibit the usage of deep learning approach in predictive process monitoring (cf. [19,23,24,38,63]), here we consider the prediction tasks that are different from the tasks that have been studied in those works. We also add more thorough explanation on several concepts and ideas of our approach so as to provide a better understanding. The discussion on the related work is also extended. Last but not least, several examples are added in order to support the explanation of various technical concepts as well as to ease the understanding of the ideas.

The remainder of this paper is structured as follows: In Sect. 2, we provide the required background on the concepts that are needed for the rest of the paper. Having laid the foundation, in Sect. 3, we present the language that we introduce for specifying the desired prediction tasks. In Sect. 4, we present a mechanism for building the corresponding prediction model based on the given specification. In Sect. 5, we continue the explanation by providing numerous showcases that exhibit the capability of our language in specifying various prediction tasks. In Sect. 6, we present the implementation of our approach as well as the experiments that we have conducted. Related work is presented in Sect. 7. In Sect. 8, we present various discussions concerning this work, including a discussion on some potential limitations, which pave the way towards our future direction. Finally, Sect. 9 concludes this work.

2 Preliminaries

We will see later that we build the prediction models by using machine learning classification/regression techniques and based on the data in event logs. To provide some background concepts, this section briefly explains the typical structure of event logs as well as the notion of classification and regression in machine learning.

2.1 Trace, event, and event log

We follow the usual notion of event logs as in process mining [67]. Essentially, an event log captures historical information of business process executions. Within an event log, an execution of a business process instance (a case) is represented as a trace. In the following, we may use the terms *trace* and *case* interchangeably. Each trace has several events, and each event in a trace captures the information about a particular event/activity that happens during the process execution. Events are characterized by various attributes, e.g., *timestamp* (the time when the event occurred).

We now proceed to formally define the notion of event logs as well as their components. 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 $n \in \mathcal{A}$, $\#_{\text{n}}(e)$ denotes the *value of attribute n* of e. For example, $\#_{\text{timestamp}}(e)$ denotes the timestamp of the event e. If an event e does not have an attribute named n, then $\#_{\text{n}}(e) = \perp$ (where $\perp$ is undefined value). A *finite sequence over* $\mathcal{E}$ *of length* n is a mapping $\sigma : \{1, \ldots, n\} \rightarrow \mathcal{E}$, and we represent such a sequence 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 trace prefix* of τ (for $1 \leq k < n$).

Example 1 For example, let $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7\} \subset \mathcal{E}$ be some event identifiers, then the sequence $\tau = \langle e_3, e_7, e_6, e_4, e_5 \rangle \in \mathcal{E}^*$ is an example of a trace. In this case, we have that $|\tau| = 5$, and $\tau(3)$ refers to the event of the trace τ at the index 3, i.e., $\tau(3) = e_6$. Moreover, τ^2 is the prefix of length 2 of the trace τ, i.e., $\tau^2 = \langle e_3, e_7 \rangle$. ■

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 [32]. 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:

1. *concept:name*, which stores the name of event/trace;
2. *org:resource*, which stores the name/identifier of the resource that triggered the event (e.g., a person name);
3. *org:group*, which stores the group name of the resource that triggered the event.

Event logs that obey this IEEE standard for event logs are often called XES event logs.

2.2 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 continuous values (quantitative outputs) [28,31]. Both of them are supervised machine learning techniques where the models are trained with labelled data. That is, the inputs for the training are pairs of input variables $\vec{x}$ and (expected) target value y. This way, the models learn how to map certain inputs $\vec{x}$ into the expected target value y.

3 Specifying the desired prediction tasks

This section elaborates our mechanism for specifying the desired prediction tasks. In predictive business process monitoring, we are interested in predicting the future information of a running process. Thus, the input of the prediction is information about the process that is currently running, e.g., what is the sequence of activities that have been executed so far, who executes a certain activity, etc. This information is often referred to as a (partial) business process execution information or a (partial) trace, which consists of a sequence of events that have occurred during the process execution. On the other hand, the output of the prediction is the future information of the process that is currently running, e.g., how long is the remaining processing time, whether the process will comply to a certain constraint, whether an unexpected behaviour will occur, etc. Based on these facts, here we introduce a language that can capture the desired prediction task in terms of the specification for mapping each (partial) trace in the event log into the desired prediction results. In some sense, this specification language enables us to specify the desired prediction function that maps each input (partial) trace into an output that gives the corresponding predicted information. Such specification can be used to train a classification/regression model that will be used as the prediction model.

To express the specification of a prediction task, we introduce the notion of *analytic rule*. An *analytic rule* R is an expression of the form:

$$\begin{aligned} R = \langle\ & \mathsf{Cond}_1 \Longrightarrow \mathsf{Target}_1, \\ & \mathsf{Cond}_2 \Longrightarrow \mathsf{Target}_2, \\ & \quad \vdots \\ & \mathsf{Cond}_n \Longrightarrow \mathsf{Target}_n, \\ & \mathsf{DefaultTarget}\ \rangle, \end{aligned}$$

where *(i)* Cond_i (for $i \in \{1, \ldots, n\}$) is called *condition expression*; *(ii)* Target_i (for $i \in \{1, \ldots, n\}$) is called *target expression*. *(iii)* $\mathsf{DefaultTarget}$ is a special target expression called *default target expression*. *(iv)* The expression $\mathsf{Cond}_i \Longrightarrow \mathsf{Target}_i$ is called *conditional-target expression*.

Section 3.1 provides an informal intuition on our language for specifying prediction tasks. It also gives some intuitive examples that illustrate the motivation of some of our language requirements. In Sect. 3.2, we elaborate several requirements that guide the development of our specification language. Throughout Sects. 3.3 and 3.4, we introduce the language for specifying the condition and target expressions in analytic rules. Specifically, Sect. 3.4 introduces a language called First-Order Event Expression (FOE), while Sect. 3.3 elaborates several components that are needed to define such language. We will see later that FOE can be used to formally specify condition expressions and a fragment of FOE can be used to specify target expressions. Finally, the formalization of analytic rules is provided in Sect. 3.5.

3.1 Overview: prediction task specification language

An analytic rule R is interpreted as a mapping that maps each (partial) trace into a value that is obtained by evaluating the target expression in which the corresponding condition is satisfied by the corresponding trace. Let τ be a (partial) trace, such mapping R can be illustrated as follows:

$$R(\tau) = \begin{cases} eval(\mathsf{Target}_1) & \text{if } \tau \text{ satisfies } \mathsf{Cond}_1, \\ eval(\mathsf{Target}_2) & \text{if } \tau \text{ satisfies } \mathsf{Cond}_2, \\ \vdots & \vdots \\ eval(\mathsf{Target}_n) & \text{if } \tau \text{ satisfies } \mathsf{Cond}_n, \\ eval(\mathsf{DefaultTarget}) & \text{otherwise} \end{cases}$$

where $eval(\mathsf{DefaultTarget})$ and $eval(\mathsf{Target}_i)$ consecutively denote the results of evaluating the target expression $\mathsf{DefaultTarget}$ and Target_i, for $i \in \{1, \ldots, n\}$. (The formal definition of this evaluation operation is given later.)

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

To specify condition expressions 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 [61] where the atoms are expressions over some event attribute values and some comparison operators, e.g., $==$, $\neq$, $>$, $\leq$. 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 in Sect. 3.4, consider the ping-pong behaviour that can be specified as follows:

$$\begin{aligned} \mathsf{Cond}_1 = \exists i.(&i > \mathsf{curr} \ \wedge \ i+1 \leq \mathsf{last} \ \wedge \\ &\mathsf{e}[i].\text{org:resource} \neq \mathsf{e}[i+1].\text{org:resource} \ \wedge \\ &\mathsf{e}[i].\text{org:group} == \mathsf{e}[i+1].\text{org:group}) \end{aligned}$$

where *(i)* $\mathsf{e}[i+1].\text{org:group}$ is an expression for getting the org:group attribute value of the event at index $i+1$ (similarly for $\mathsf{e}[i].\text{org:resource}$, $\mathsf{e}[i+1].\text{org:resource}$, and $\mathsf{e}[i].\text{org:group}$), *(ii)* curr refers to the current time point, and *(iii)* last refers to the last time point.

The formula Cond_1 basically says that *there exists a time point i that is greater than the current time point (i.e., in the future), in which the resource (the person in charge) is different from 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 an analytic rule R_1 as follows:

$$R_1 = \langle \mathsf{Cond}_1 \Longrightarrow \text{“ping-pong”}, \ \text{“not ping-pong”} \rangle,$$

where Cond_1 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 Cond_1 or not (i.e., a trace will be classified into "ping-pong" if it satisfies Cond_1, otherwise it will be classified into "not ping-pong"). During the prediction phase, such classifier can be used to predict whether a given (partial) trace will lead to 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[1]

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

where $\mathsf{e}[\mathsf{last}].\text{time:timestamp}$ refers to the timestamp of the last event and $\mathsf{e}[\mathsf{curr}].\text{time:timestamp}$ refers to the timestamp of the current event. Essentially, the expression $\mathsf{Target}_{\text{remainingTime}}$ 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 processing time, because R_2 maps each (partial) trace into its remaining processing time. In this case, during the prediction model creation phase, we will create a regression model for predicting the remaining processing time of a given (partial) trace. Section 5 provides more examples of prediction tasks specification using our language.

3.2 Specification language requirements

Driven by various typical prediction tasks that are studied in the area of predictive process monitoring (see [20,36] for an extensive overview), in the following, we elaborate several requirements for our language as well as the motivation

[1] Note that, as usual, a timestamp can be represented as milliseconds since Unix epoch (i.e., the number of milliseconds that have elapsed since Jan 1, 1970 00:00:00 UTC).

for each requirement. These requirements guide the development of our language.

As can be seen from the illustrative examples in Sect. 3.1, the output of the prediction could be either numerical or non-numerical (categorical) values. This immediately gives the first requirement of our language as follows:

Req. 1: The language should support the specification of prediction tasks in which the outputs could be either numerical or non-numerical values/information.

One of our goals is to have a mechanism for automatically processing the prediction task specification so as to build the corresponding prediction model. Consequently, our language should fulfil the following requirement:

Req. 2: The language should allow us to automatically build the corresponding prediction model from the given specification.

When it comes to predicting the outcome of a process, the existence of an unexpected behaviour, as well as the satisfaction of business constraints, we need to be able to precisely specify the desired outcomes, the corresponding unexpected behaviour, and the corresponding business constraint. As can be seen from the example in Sect. 3.1, these kinds of tasks often require us to express complex properties involving events data (attribute values). Additionally, we might also need to be able to universally or existentially quantify different event time points (e.g., saying that there exist a time point where a certain condition holds) and to compare different event attribute values at different time points (e.g., saying that the person in charge at time point i is different from the person in charge at time point $i + 1$). As a consequence, this gives us the following requirement:

Req. 3: The language should allow us to express complex constraints/properties over sequence of events by also involving the corresponding events data (attribute values). This includes the capability to universally/existentially quantify different event time points and to compare different event attribute values at different time points.

In Sect. 1, we have seen that we might even need to specify a business constraint that requires that *the length of customer order processing to be less than 3 hours*. Obviously, this requires us to specify an arithmetic expression involving the data. On the other hand, we might also need to specify a constraint saying that *the total cost of all validation activities should be less than 100 Eur*. In order to be able to specify this constraint, we need to be able to aggregate (sum up) the cost of each validation activity, and *only* to sum up those validation activities. Thus, we need to be able to aggregate some particular attribute values. All of these give us the following requirement:

Req. 4: The language should allow us to specify arithmetic expressions as well as aggregate functions involving the data. Additionally, the language should allow us to do selective aggregation operations (i.e., selecting the values to be aggregated).

In the example of predicting the remaining processing time in Sect. 3.1, we also need to be able to specify the target information to be predicted. This often requires us to specify the way to obtain a certain value. For instance, to obtain the remaining processing time, we need to take the difference between the timestamp of the last event and the current event. Hence, we need to be able to specify that the remaining processing time is the difference between the timestamp of the last event and the current event. This gives us the following requirement:

Req. 5: The language should allow us to specify the target information to be predicted, and this might include the way to obtain a certain value (which might involve some arithmetic expressions).

3.3 Towards formalizing the condition and target expressions

This section is devoted to introduce several components that are needed to define the language for specifying condition and target expressions in Sect. 3.4.

As we have seen in Sect. 3.1, we often need to refer to a particular index of an event within a trace. Recall the expression $\mathsf{e}[i+1]$**.** org:group that refers to the org:group attribute value of the event at the index $i+1$, and also the expression $\mathsf{e}[\mathsf{last}]$**.** time:timestamp that refers to the timestamp of the last event. The former requires us to refer to the event at the index $i + 1$, while the latter requires us to refer to the last event in the trace. To capture this, we introduce the notion of *index expression* idx defined as follows:

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

where *(i)* i is an *index variable*. *(ii)* $pint$ is a positive integer (i.e., $pint \in \mathbb{Z}^+$). *(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 trace 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 operations over indices.

The semantics of index expression is defined over traces and considered trace prefix length. Since an index expression can be a variable, given a trace τ and a considered trace prefix length k, we first introduce a *variable valuation* ν, i.e., a mapping from index variables into $\mathbb{Z}^+$. We assign meaning to index expression by associating with 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:

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

The definition above says that the interpretation function $(\cdot)_\nu^{\tau,k}$ interprets index expressions as follows: *(i)* Each variable is interpreted based on how the variable valuation ν maps the corresponding variable into a positive integer in $\mathbb{Z}^+$; *(ii)* each positive integer is interpreted as itself, e.g., $(2603)_\nu^{\tau,k} = 2603$; *(iii)* curr is interpreted into k; *(iv)* last is interpreted into $|\tau|$; and *(v)* the arithmetic addition/subtraction operators are interpreted as usual.

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

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

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:

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

Note that the above definition also says that if the event attribute accessor refers to an index that is beyond the valid event indices in the corresponding trace, then we will get undefined value (i.e., $\bot$).

As an example of event attribute accessor, the expression $\mathsf{e}[i].\,$org:resource refers to the value of the attribute org:resource of the event at the position i.

Example 2 Consider the trace $\tau = \langle e_1, e_2, e_3, e_4, e_5\rangle$, let "Bob" be the value of the attribute org:resource of the event e_3 in τ, i.e., $\#_{\text{org:resource}}(e_3) = \text{“Bob”}$, and e_3 does not have any attributes named org:group, i.e., $\#_{\text{org:group}}(e_3) = \bot$. In this example, we have that $(\mathsf{e}[3].\,\text{org:resource})_\nu^{\tau,k} = \text{“Bob”}$, and $(\mathsf{e}[3].\,\text{org:group})_\nu^{\tau,k} = \bot$. ■

The value of an event attribute within a trace 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 follows:

$$\begin{aligned}
\mathsf{nonNumExp} \;::=\;& \mathsf{true} \mid \mathsf{false} \mid \mathsf{String} \mid\\
& \mathsf{e}[\mathsf{idx}].\,\mathrm{NonNumericAttribute}\\[1em]
\mathsf{numExp} \;::=\;& \mathsf{number} \mid \mathsf{idx} \mid\\
& \mathsf{e}[\mathsf{idx}].\,\mathrm{NumericAttribute} \mid\\
& \mathsf{numExp}_1 + \mathsf{numExp}_2 \mid\\
& \mathsf{numExp}_1 - \mathsf{numExp}_2
\end{aligned}$$

where *(i)* true and false are the usual boolean values, *(ii)* String is the usual string (i.e., a sequence of characters), *(iii)* number is a real number, *(iv)* $\mathsf{e}[\mathsf{idx}].\,$NonNumericAttribute is an event attribute accessor for accessing an attribute with non-numeric values, and $\mathsf{e}[\mathsf{idx}].\,$NumericAttribute is an event attribute accessor for accessing an attribute with 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.,

$$\begin{aligned}
(3)_\nu^{\tau,k} &= 3,\\
(\text{“sendOrder”})_\nu^{\tau,k} &= \text{“sendOrder”},
\end{aligned}$$

and by interpreting the arithmetic operations as usual, e.g.,

$$\begin{aligned}
(26 + 3)_\nu^{\tau,k} &= (26)_\nu^{\tau,k} + (3)_\nu^{\tau,k} = 26 + 3 = 29,\\
(86 - 3)_\nu^{\tau,k} &= (86)_\nu^{\tau,k} - (3)_\nu^{\tau,k} = 86 - 3 = 83.
\end{aligned}$$

Formally, we extend our interpretation function as follows:

$$\begin{aligned}
(\mathsf{true})_\nu^{\tau,k} &= \mathsf{true}\\
(\mathsf{false})_\nu^{\tau,k} &= \mathsf{false}\\
(\mathsf{String})_\nu^{\tau,k} &= \mathsf{String}\\
(\mathsf{number})_\nu^{\tau,k} &= \mathsf{number}\\
(\mathsf{numExp}_1 + \mathsf{numExp}_2)_\nu^{\tau,k} &= (\mathsf{numExp}_1)_\nu^{\tau,k} + (\mathsf{numExp}_2)_\nu^{\tau,k}\\
(\mathsf{numExp}_1 - \mathsf{numExp}_2)_\nu^{\tau,k} &= (\mathsf{numExp}_1)_\nu^{\tau,k} - (\mathsf{numExp}_2)_\nu^{\tau,k}
\end{aligned}$$

Note that the value of an event attribute might be undefined, i.e., it is equal to $\bot$. In this case, we define that the arithmetic operations involving $\bot$ give $\bot$, e.g., $26 + \bot = \bot$.

We now define the notion of *event expression* as a comparison between either numeric expressions or non-numeric expressions. Formally, it is defined as follows:

$$\begin{array}{lcl} \mathsf{eventExp} & ::= & \mathsf{true} \mid \mathsf{false} \mid \\ & & \mathsf{numExp}_1 == \mathsf{numExp}_2 \quad \mid \\ & & \mathsf{numExp}_1 \neq \mathsf{numExp}_2 \quad \mid \\ & & \mathsf{numExp}_1 < \mathsf{numExp}_2 \quad \mid \\ & & \mathsf{numExp}_1 > \mathsf{numExp}_2 \quad \mid \\ & & \mathsf{numExp}_1 \leq \mathsf{numExp}_2 \quad \mid \\ & & \mathsf{numExp}_1 \geq \mathsf{numExp}_2 \quad \mid \\ & & \mathsf{nonNumExp}_1 == \mathsf{nonNumExp}_2 \quad \mid \\ & & \mathsf{nonNumExp}_1 \neq \mathsf{nonNumExp}_2 \end{array}$$

where *(i)* numExp is a numeric expression; *(ii)* nonNumExp is a non-numeric expression; *(iii)* the operators $==$ and $\neq$ are the usual logical comparison operators, namely *equality* and *inequality*; *(iv)* the operators $<, >, \leq$, and $\geq$ are the usual arithmetic comparison operators, namely *less than*, *greater than*, *less than or equal*, and *greater than or equal*.

Example 3 The expression

$$\mathbb{e}[i].\text{org:resource} \neq \mathbb{e}[i+1].\text{org:resource}$$

is an example of an event expression which says that the resource at the time point i is different from the resource at the time point $i+1$. As another example, the expression

$$\mathbb{e}[i].\text{concept:name} == \text{“OrderCreated”}$$

is an event expression saying that the value of the attribute concept:name of the event at the index i is equal to “OrderCreated”. ■

We interpret each logical/arithmetic comparison operator (i.e., $==$, $\neq$, $<$, $>$, etc) in the event expressions as usual. For instance, the expression $26 \geq 3$ is interpreted as true, while the expression “receivedOrder” $==$ “sendOrder” is interpreted as false. Additionally, any comparison involving undefined value ($\bot$) is interpreted as false. It is easy to see how to extend the formal definition of our interpretation function $(\cdot)^{\tau,k}_{\nu}$ towards interpreting event expressions; therefore, we omit the details.

3.3.1 Adding aggregate functions

We now extend the notion of *numeric expression* and *non-numeric expression* by adding several numeric and non-numeric aggregate functions. A numeric (resp. non-numeric) aggregate function is a function that performs an aggregation operation over some values and return a numeric (resp. non-numeric) value. Before providing the formal syntax and semantics of our aggregate functions, in the following we illustrate the needs of having aggregate functions and we provide some intuition on the shape of our aggregate functions.

Suppose that each event in each trace has an attribute named cost. Consider the situation where we want to specify a task for predicting the total cost of all activities (from the first until the last event) within a trace. In this case, we need to sum up all values of the cost attribute in all events. To express this need, we introduce the aggregate function **sum** and we can specify the notion of total cost as follows:

$$\textbf{sum}(\mathbb{e}[x].\text{cost};\ \textbf{where}\ x = 1 : \mathsf{last}).$$

The expression above computes the sum of the values of $\mathbb{e}[x].\text{cost}$ for all $x \in \{1, \dots, \mathsf{last}\}$. In this case x is called *aggregation variable*, the expression $\mathbb{e}[x].\text{cost}$ specifies the *aggregation source*, i.e., the source of the values to be aggregated, and the expression $x = 1 : \mathsf{last}$ specifies the *aggregation range* by defining the range of the aggregation variable x.

In some situation, we might only be interested to compute the total cost of a certain activity, e.g., the total cost of all validation activities within a trace. To do this, we introduce the notion of *aggregation condition*, which allows us to select only some values that we want to aggregate. For example, the expression

$$\begin{array}{l}\textbf{sum}(\mathbb{e}[x].\text{cost};\ \textbf{where}\ x = 1 : \mathsf{last}; \quad . \\ \quad \textbf{and}\ \mathbb{e}[x].\text{concept:name} == \text{“Validation”})\end{array}$$

computes the sum of the values of the attribute $\mathbb{e}[x].\text{cost}$ for all $x \in \{1, \dots, \mathsf{last}\}$ in which the expression

$$\mathbb{e}[x].\text{concept:name} == \text{“Validation”}$$

is evaluated to true. Therefore, the summation only considers the values of x in which the activity name is “Validation”, and we only compute the total cost of all validation activities. As before, $\mathbb{e}[x].\text{cost}$ specifies the source of the values to be aggregated, the expression $x = 1 : \mathsf{last}$ specifies the *aggregation range* by defining the range of the aggregation variable x, and the expression $\mathbb{e}[x].\text{concept:name} ==$ “Validation” provides the *aggregation condition*.

The expression for specifying the source of the values to be aggregated can be more complex, for example when we want to compute the average activity running time within a trace. In this case, the running time of an activity is specified as the time difference between the timestamp of that activity and the next activity, i.e.,

$\mathrm{e}[x+1].\mathrm{time{:}timestamp} - \mathrm{e}[x].\mathrm{time{:}timestamp}$.

Then, the average activity running time can be specified as follows:

$$\textbf{avg}(\mathrm{e}[x+1].\mathrm{time{:}timestamp} - \mathrm{e}[x].\mathrm{time{:}timestamp};\ .\ \textbf{where}\ x = 1 : \mathsf{last})$$

Essentially, the expression above computes the average of the time difference between the activity at the time point $x+1$ and x, where $x \in \{1, \ldots, \mathsf{last}\}$.

In other cases, we might not be interested in aggregating the data values, but we are interested in counting the number of a certain activity/event. To do this, we introduce the aggregate function **count**. As an example, we can specify an expression to count the number of validation activities within a trace as follows:

$$\textbf{count}(\mathrm{e}[x].\mathrm{concept{:}name} == \text{“validation”};\ \textbf{where}\ x = 1 : \mathsf{last})$$

where $\mathrm{e}[x].\mathrm{concept{:}name} == $ “validation” is an *aggregation condition*. The expression above counts how many times the specified aggregation condition is true within the specified range. Thus, in this case, it counts the number of the events between the first and the last event, in which the activity name is “validation”.

We might also be interested in counting the number of different values of a certain attribute within a trace. For example, we might be interested in counting the number of different resources that are involved within a trace. To capture this, we introduce the aggregate function **countVal**. We can then specify the expression to count the number of different resources between the first and the last event as follows:

$$\textbf{countVal}(\mathrm{org{:}resource};\ \textbf{within}\ 1 : \mathsf{last})$$

where *(i)* org:resource is the name of the attribute in which we want to count its number of different values; and *(ii)* the expression “**within** $1 : \mathsf{last}$” is the *aggregation range*.

We will see later in Sect. 5 that the presence of aggregate functions allows us to express numerous interesting prediction tasks. Towards formalizing the aggregate functions, we first formalize the notion of *aggregation conditions*. An aggregation condition is an unquantified First-Order Logic (FOL) [61] formula where the atoms are event expressions and may use only a single unquantified variable, namely the aggregation variable. The values of the unquantified/free variable in aggregation conditions are ranging over the specified aggregation range in the corresponding aggregate function. Formally, *aggregation conditions* are defined as follows:

$$\mathsf{aggCond} ::= \mathsf{eventExp} \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \psi_1 \vee \psi_2$$

where eventExp is an event expression, and the semantics of aggCond is based on the usual FOL semantics. Formally, we extend the definition of our interpretation function $(\cdot)_\nu^{\tau,k}$ as follows:

$$\begin{array}{ll}(\neg\psi)_\nu^{\tau,k} & = \mathsf{true},\ \text{if}\ (\psi)_\nu^{\tau,k} = \mathsf{false}\\ (\psi_1 \wedge \psi_2)_\nu^{\tau,k} & = \mathsf{true},\ \text{if}\ (\psi_1)_\nu^{\tau,k} = \mathsf{true},\ \text{and}\ (\psi_2)_\nu^{\tau,k} = \mathsf{true}\\ (\psi_1 \vee \psi_2)_\nu^{\tau,k} & = \mathsf{true},\ \text{if}\ (\psi_1)_\nu^{\tau,k} = \mathsf{true},\ \text{or}\ (\psi_2)_\nu^{\tau,k} = \mathsf{true}\end{array}$$

With this machinery in hand, we are ready to define the syntax and the semantics of numeric and non-numeric aggregate functions. We first extend the syntax of the numeric and non-numeric expressions by adding the *numeric and non-numeric aggregate functions* as follows:

$$\begin{array}{l}\mathsf{nonNumExp} ::=\\ \quad \mathsf{true} \mid \mathsf{false} \mid \mathsf{String} \mid \mathrm{e}[\mathsf{idx}].\mathrm{NonNumericAttribute} \mid\\ \quad \textbf{concat}(\mathsf{nonNumSrc};\ \textbf{where}\ x = \mathsf{st} : \mathsf{ed};\ \textbf{and}\ \mathsf{aggCond})\end{array}$$

$$\begin{array}{l}\mathsf{numExp} ::= \mathsf{number} \mid \mathsf{idx} \mid \mathrm{e}[\mathsf{idx}].\mathrm{NumericAttribute} \mid\\ \quad \mathsf{numExp}_1 + \mathsf{numExp}_2 \mid \mathsf{numExp}_1 - \mathsf{numExp}_2 \mid\\ \quad \textbf{sum}(\mathsf{numSrc};\ \textbf{where}\ x = \mathsf{st} : \mathsf{ed};\ \textbf{and}\ \mathsf{aggCond}) \mid\\ \quad \textbf{avg}(\mathsf{numSrc};\ \textbf{where}\ x = \mathsf{st} : \mathsf{ed};\ \textbf{and}\ \mathsf{aggCond}) \mid\\ \quad \textbf{min}(\mathsf{numSrc};\ \textbf{where}\ x = \mathsf{st} : \mathsf{ed};\ \textbf{and}\ \mathsf{aggCond}) \mid\\ \quad \textbf{max}(\mathsf{numSrc};\ \textbf{where}\ x = \mathsf{st} : \mathsf{ed};\ \textbf{and}\ \mathsf{aggCond}) \mid\\ \quad \textbf{min}(\mathsf{numExp}_1, \mathsf{numExp}_2) \mid \textbf{max}(\mathsf{numExp}_1, \mathsf{numExp}_2) \mid\\ \quad \textbf{count}(\mathsf{aggCond};\ \textbf{where}\ x = \mathsf{st} : \mathsf{ed}) \mid\\ \quad \textbf{countVal}(\mathsf{attName};\ \textbf{within}\ \mathsf{st} : \mathsf{ed})\end{array}$$

where *(i)* number, idx, $\mathrm{e}[\mathsf{idx}].\mathrm{NumericAttribute}$, $\mathrm{e}[\mathsf{idx}].\mathrm{NonNumericAttribute}$, $\mathsf{numExp}_1 + \mathsf{numExp}_2$, and $\mathsf{numExp}_1 - \mathsf{numExp}_2$ are as before; *(ii)* st and ed are either positive integers (i.e., $\mathsf{st} \in \mathbb{Z}^+$ and $\mathsf{ed} \in \mathbb{Z}^+$) or special indices (i.e., last or curr), and $\mathsf{st} \leq \mathsf{ed}$; *(iii)* x is a variable called *aggregation variable*, and the range of its value is between st and ed (i.e., $\mathsf{st} \leq x \leq \mathsf{ed}$). The expressions **where** $x = \mathsf{st} : \mathsf{ed}$ and **within** $\mathsf{st} : \mathsf{ed}$ are called *aggregation variable range*; *(iv)* numSrc and nonNumSrc specify the source of the values to be aggregated. The numSrc is specified as numeric expression, while nonNumSrc is specified as non-numeric expression. Both of them may and can only use the corresponding aggregation variable x, and they cannot contain any aggregate functions; *(v)* aggCond is an *aggregation condition* over the corresponding aggregation variable x and no other variables are allowed to occur in aggCond; *(vi)* attName is an attribute name; *(vii)* for the aggregate functions, as the names describe, **sum** stands for summation, **avg** stands for average, **min** stands for minimum, **max** stands for maximum, **count** stands for counting, **countVal** stands for counting values, and **concat** stands for concatenation. The behaviour of these aggregate functions is quite intuitive. Some intuition has been given previously, and we explain their details behaviour while providing their formal semantics below. The aggregate functions **sum**, **avg**, **min**,

max, **concat** that have aggregation conditions aggCond are also called *conditional aggregate functions*.

Notice that a numeric aggregate function is also a numeric expression and a numeric expression is also a component of a numeric aggregate function (either in the source value or in the aggregation condition). Hence, it may create some sort of nested aggregate function. However, to simplify the presentation, in this work we do not allow nested aggregation functions of this form, but technically it is possible to do that under a certain care on the usage of the variables (similarly for the non-numeric aggregate function).

To formalize the semantics of aggregate functions, we first introduce some notations. Given a variable valuation ν, we write $\nu[x \mapsto d]$ to denote a new variable valuation obtained from the variable valuation ν as follows:

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

Intuitively, $\nu[x \mapsto d]$ substitutes each variable x with d, while the other variables (apart from x) are substituted the same way as ν is defined. Given a conditional summation aggregate function

$$\textbf{sum}(\mathsf{numSrc};\ \textbf{where}\ x = \mathsf{st} : \mathsf{ed};\ \textbf{and}\ \mathsf{aggCond}),$$

a trace τ, a considered trace prefix length k, and a variable valuation ν, we define its corresponding *set* Idx *of valid aggregation indices* as follows:

$$\mathsf{Idx} = \{d \in \mathbb{Z}^+ \mid \mathsf{st} \leq d \leq \mathsf{ed},\ (\mathsf{aggCond})^{\tau,k}_{\nu[x \mapsto d]} = \mathsf{true}, \text{ and } (\mathsf{numSrc})^{\tau,k}_{\nu[x \mapsto d]} \neq \bot\}.$$

basically, Idx collects the values within the given aggregation range (i.e., between st and ed), in which, by substituting the aggregation variable x with those values, the aggregation condition aggCond is evaluated to true and numSrc is not evaluated to undefined value $\bot$. For the other conditional aggregate functions **avg**, **max**, **min**, and **concat**, the corresponding set of valid aggregation indices can be defined similarly.

Example 4 Consider the trace $\tau = \langle e_1, e_2, e_3, e_4\rangle$, let "validation" be the value of the attribute concept:name of the event e_2 and e_4 in τ, i.e.,

$$\#_{\text{concept:name}}(e_2) = \#_{\text{concept:name}}(e_4) = \text{"validation"}.$$

Moreover, let $\#_{\text{concept:name}}(e_1) = $ "initialization" and $\#_{\text{concept:name}}(e_3) = $ "assembling". Suppose that the cost of each activity is the same, let say it is equal to 3, i.e.,

$$\#_{\text{cost}}(e_1) = \#_{\text{cost}}(e_2) = \#_{\text{cost}}(e_3) = \#_{\text{cost}}(e_4) = 3,$$

and we have the following aggregate function specification:

$$\textbf{sum}(\mathsf{e}[x].\text{cost};\ \textbf{where}\ x = 1 : \mathsf{last};\ \textbf{and}\ \mathsf{true}),$$

and

$$\textbf{sum}(\mathsf{e}[x].\text{cost};\ \textbf{where}\ x = 1 : \mathsf{last};\ \textbf{and}\ \mathsf{e}[x].\text{cost} == \text{"validation"})$$

The former computes the total cost of all activities, while the latter computes the total cost of validation activities. In this case, the corresponding set of the valid aggregation indices (with respect to the given trace τ) for the first aggregate function is $\mathsf{Idx}_1 = \{1, 2, 3, 4\}$, while for the second aggregate function we have $\mathsf{Idx}_2 = \{2, 4\}$ because the second aggregate function requires that the activity name (i.e., the value of the attribute concept:name) to be equal to "validation" and it is only true when x is equal to either 2 or 4. ■

Having this machinery in hand, we are now ready to formally define the semantics of aggregate functions. The formal semantics of the conditional aggregate functions **sum**, **avg**, **max**, **min** is provided in Fig. 2. Intuitively, the aggregate function **sum** computes the sum of the values that are obtained from the evaluation of the specified numeric expression numSrc over the specified aggregation range (i.e., between st and ed). Additionally, the computation of the summation ignores undefined values and it only considers those indices within the specified aggregation range in which the aggregation condition is evaluated to true. The intuition for the aggregate functions **avg**, **max**, **min** is similar, except that **avg** computes the average, **max** computes the maximum values, and **min** computes the minimum values.

Example 5 Continuing Example 4, the first aggregate function is evaluated to 12 because we have that $\mathsf{Idx}_1 = \{1, 2, 3, 4\}$, and

$$\begin{aligned}\sum_{d \in \mathsf{Idx}_1} (\mathsf{e}[x].\text{cost})^{\tau,k}_{\nu[x \mapsto d]} &= (\mathsf{e}[x].\text{cost})^{\tau,k}_{\nu[x \mapsto 1]} + \\ &\quad (\mathsf{e}[x].\text{cost})^{\tau,k}_{\nu[x \mapsto 2]} + \\ &\quad (\mathsf{e}[x].\text{cost})^{\tau,k}_{\nu[x \mapsto 3]} + \\ &\quad (\mathsf{e}[x].\text{cost})^{\tau,k}_{\nu[x \mapsto 4]} \\ &= 12.\end{aligned}$$

On the other hand, the second aggregate function is evaluated to 6 because we have that $\mathsf{Idx}_2 = \{2, 4\}$, and

$$\begin{aligned}\sum_{d \in \mathsf{Idx}_2} (\mathsf{e}[x].\text{cost})^{\tau,k}_{\nu[x \mapsto d]} &= (\mathsf{e}[x].\text{cost})^{\tau,k}_{\nu[x \mapsto 2]} + \\ &\quad (\mathsf{e}[x].\text{cost})^{\tau,k}_{\nu[x \mapsto 4]} \\ &= 6\end{aligned}$$

■

$$(\mathbf{sum}(\mathsf{numSrc};\ \mathbf{where}\ x = \mathsf{st} : \mathsf{ed};\ \mathbf{and}\ \mathsf{aggCond}))^{\tau,k}_{\nu} = \begin{cases} \sum_{d \in \mathsf{Idx}} (\mathsf{numSrc})^{\tau,k}_{\nu[x \mapsto d]} & \text{if } \mathsf{Idx} \neq \emptyset \\ \bot & \text{otherwise} \end{cases}$$

$$(\mathbf{avg}(\mathsf{numSrc};\ \mathbf{where}\ x = \mathsf{st} : \mathsf{ed};\ \mathbf{and}\ \mathsf{aggCond}))^{\tau,k}_{\nu} = \begin{cases} \left(\sum_{d \in \mathsf{Idx}} (\mathsf{numSrc})^{\tau,k}_{\nu[x \mapsto d]}\right) \ / \ |\mathsf{Idx}| & \text{if } \mathsf{Idx} \neq \emptyset \\ \bot & \text{otherwise} \end{cases}$$

$$(\mathbf{max}(\mathsf{numSrc};\ \mathbf{where}\ x = \mathsf{st} : \mathsf{ed};\ \mathbf{and}\ \mathsf{aggCond}))^{\tau,k}_{\nu} = \begin{cases} (\mathsf{numSrc})^{\tau,k}_{\nu[x \mapsto i]} & \text{if there exists } i \in \mathsf{Idx} \text{ such that} \\ & \text{for each } j \in \mathsf{Idx} \text{ we have that} \\ & (\mathsf{numSrc})^{\tau,k}_{\nu[x \mapsto i]} \geq (\mathsf{numSrc})^{\tau,k}_{\nu[x \mapsto j]} \\ \bot & \text{otherwise} \end{cases}$$

$$(\mathbf{min}(\mathsf{numSrc};\ \mathbf{where}\ x = \mathsf{st} : \mathsf{ed};\ \mathbf{and}\ \mathsf{aggCond}))^{\tau,k}_{\nu} = \begin{cases} (\mathsf{numSrc})^{\tau,k}_{\nu[x \mapsto i]} & \text{if there exists } i \in \mathsf{Idx} \text{ such that} \\ & \text{for each } j \in \mathsf{Idx} \text{ we have that} \\ & (\mathsf{numSrc})^{\tau,k}_{\nu[x \mapsto i]} \leq (\mathsf{numSrc})^{\tau,k}_{\nu[x \mapsto j]} \\ \bot & \text{otherwise} \end{cases}$$

$$\text{where } \mathsf{Idx} = \{d \in \mathbb{Z}^+ \mid \mathsf{st} \leq d \leq \mathsf{ed},\ (\mathsf{aggCond})^{\tau,k}_{\nu[x \mapsto d]} = \mathsf{true},\ \text{ and }\ (\mathsf{numSrc})^{\tau,k}_{\nu[x \mapsto d]} \neq \bot\}.$$

Fig. 2 Formal semantics of aggregate functions **sum**, **avg**, **max**, **min** in the presence of aggregation conditions

The aggregate function **max**(numExp_1, numExp_2) computes the maximum value between the two values that are obtained by evaluating the specified two numeric expressions numExp_1 and numExp_2. It gives undefined value $\bot$ if one of them is evaluated to undefined value $\bot$ (similarly for the aggregate function **min**(numExp_1, numExp_2) except that it computes the minimum value). The detailed formal semantics of these functions is provided in Appendix A.

The formal semantics of the aggregate function **count** is provided below

$$(\mathbf{count}(\mathsf{aggCond};\ \mathbf{where}\ x = \mathsf{st} : \mathsf{ed}))^{\tau,k}_{\nu} = \\ |\{d \in \mathbb{Z}^+ \mid \mathsf{st} \leq d \leq \mathsf{ed}, \text{and } (\mathsf{aggCond})^{\tau,k}_{\nu[x \mapsto d]} = \mathsf{true}\}|$$

Intuitively, it counts how many times the aggCond is evaluated to true within the given range, i.e., between st and ed. This aggregate function is useful to count the number of events/activities within a certain range that satisfy a certain condition, for example to count the number of the activity named "*modifying delivery appointment*" within a certain range in a trace.

Notice that **count** is a syntactic variant of **sum**. It is easy to see that we could express

$$\mathbf{count}(\mathsf{aggCond};\ \mathbf{where}\ x = \mathsf{st} : \mathsf{ed})$$

as

$$\mathbf{sum}(1;\ \mathbf{where}\ x = \mathsf{st} : \mathsf{ed};\ \mathbf{and}\ \mathsf{aggCond}).$$

Both of the expressions above count how many times the aggCond is evaluated to true within the given range. However, since expressing the counting aggregation using sum is a bit less intuitive, we choose to explicitly introduce the aggregate function **count**.

The semantics of the aggregate function **countVal** is formally defined as follows:

$$(\mathbf{countVal}(\text{attName};\ \mathbf{within}\ \mathsf{st} : \mathsf{ed}))^{\tau,k}_{\nu} = \\ |\{v \mid (\mathsf{e}[x].\text{attName})^{\tau,k}_{\nu[x \mapsto d]} = v, \text{and } \mathsf{st} \leq d \leq \mathsf{ed}\}|.$$

Intuitively, it counts the number of all possible values of the attribute attName within all events between the given start and end time points (i.e., between st and ed).

The aggregate function **concat** concatenates the values that are obtained from the evaluation of the given non-numeric expression under the valid aggregation range (i.e., we only consider the value within the given aggregation range in which the aggregation condition is satisfied). Moreover, the concatenation ignores undefined values and treats them as empty string. The detailed formal semantics of the aggregate function **concat** is provided in Appendix A.

Notice that, for convenience, we could easily extend our language with unconditional aggregate functions by adding the following:

$$\begin{array}{l} \mathbf{sum}(\mathsf{numSrc};\ \mathbf{where}\ x = \mathsf{st} : \mathsf{ed}) \\ \mathbf{avg}(\mathsf{numSrc};\ \mathbf{where}\ x = \mathsf{st} : \mathsf{ed}) \\ \mathbf{min}(\mathsf{numSrc};\ \mathbf{where}\ x = \mathsf{st} : \mathsf{ed}) \\ \mathbf{max}(\mathsf{numSrc};\ \mathbf{where}\ x = \mathsf{st} : \mathsf{ed}) \\ \mathbf{concat}(\mathsf{nonNumSrc};\ \mathbf{where}\ x = \mathsf{st} : \mathsf{ed}) \end{array}$$

In this case, they simply perform an aggregation computation over the values that are obtained by evaluating the specified numeric/non-numeric expression over the specified aggregation range. However, they do not give additional expressive power since they are only syntactic variant of the current conditional aggregate functions. This is the case because we can simply put "true" as the aggregation condition, e.g., **sum**(numSrc; **where** x = st : ed; **and** true). Based on their semantics, we get the aggregate functions that behave as unconditional aggregate functions. That is, they ignore the aggregation condition since it will always be true for every values within the specified aggregation range. In the following, for the brevity of presentation, when aggregation condition is not important we often simply use the unconditional version of aggregate functions.

3.4 First-Order Event Expression (FOE)

Finally, we are ready to define the language for specifying condition expression, namely First-Order Event Expression (FOE). A part of this language is also used to specify target expression.

An FOE formula is a First-Order Logic (FOL) [61] formula where the atoms are event expressions and the quantification is ranging over event indices. Syntactically, FOE is defined as follows:

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

where *(i)* eventExp is an event expression; *(ii)* $\neg\varphi$ is negated FOE formula; *(iii)* $\forall i.\varphi$ is an FOE formula where the variable i is universally quantified; *(iv)* $\exists i.\varphi$ is an FOE formula where the variable i is existentially quantified; *(v)* $\varphi_1 \wedge \varphi_2$ is a conjunction of FOE formulas; *(vi)* $\varphi_1 \vee \varphi_2$ is a disjunction of FOE formulas; *(vii)* $\varphi_1 \rightarrow \varphi_2$ is an FOE implication formula saying that φ_1 implies φ_2; *(viii)* The notion of free and bound variables is as usual in FOL, except that the variables inside aggregate functions, i.e., aggregation variables, are not considered as free variables; *(ix)* the aggregation variables cannot be existentially/universally quantified.

The semantics of FOE constructs is based on the usual FOL semantics. Formally, we extend the definition of our interpretation function $(\cdot)^{\tau,k}_{\nu}$ as follows[2]:

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

[2] 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 [61].

As before, $\nu[i \mapsto c]$ substitutes each variable i with c, while the other variables are substituted the same way as ν is defined. When φ is a closed formula, its truth value does not depend on the valuation of the variables, and we denote the interpretation of φ simply by $(\varphi)^{\tau,k}$. We also say that the trace τ and the prefix length k satisfy φ, written $\tau, k \models \varphi$, if $(\varphi)^{\tau,k} = \mathsf{true}$. With a little abuse of notation, sometimes we also say that the *k-length trace prefix τ^k of the trace τ satisfies* φ, written $\tau^k \models \varphi$, if $\tau, k \models \varphi$.

Example 6 An example of a closed FOE formula is as follows:

$$\begin{aligned}
&\forall i.(\mathsf{e}[i].\text{concept:name} == \text{"OrderCreated"} \rightarrow\\
&\quad \exists j.(j > i \;\wedge\; \mathsf{e}[i].\text{orderID} == \mathsf{e}[j].\text{orderID} \;\wedge\\
&\qquad \mathsf{e}[j].\text{concept:name} == \text{"OrderDelivered"} \;\wedge\\
&\qquad (\mathsf{e}[j].\text{time:timestamp} - \mathsf{e}[i].\text{time:timestamp}) \leq\\
&\qquad 10.800.000\\
&\quad)\\
&)
\end{aligned}$$

which essentially says that *whenever there is an event where an order is created, eventually there will be an event where the corresponding order is delivered and the time difference between the two events (the processing time) is less than or equal to 10.800.000 milliseconds (3 hours).* ■

In general, FOE has the following main features: *(i)* It allows us to specify constraints over the data (attribute values); *(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 allows us to specify arithmetic expressions/operations involving the data as well as aggregate functions; *(iv)* it allows us to do selective aggregation operations (i.e., selecting the values to be aggregated). *(v)* The fragments of FOE, namely the numeric and non-numeric expressions, allow us to specify the way to compute a certain value.

Notice that here we opt for FOL-style syntax for providing the mechanism to refer to a certain event attribute at a particular time point as well as to compare the event attribute values at two different time points. Another possible alternative for expressing properties that involve time points would be to adopt the LTL-style syntax [51] by using the temporal operators. For instance, we could roughly express the following property:

$$\forall i.(\mathsf{e}[i].\text{org:resource} \neq \text{"Bob"})$$

which says that, *at every time points, it is always be the case that the value of the attribute 'org:resource' is not equal to 'Bob'*, as

$$\mathbf{G}(\text{org:resource} \neq \text{“Bob”})$$

where **G** is the usual LTL temporal operator that is typically called 'Global'. However, when we want to compare the values of two attributes at two different time points, e.g.,

$$\exists i.(\mathrm{e}[i].\,\text{org:resource} == \mathrm{e}[i+3].\,\text{org:resource})$$

we cannot easily do it using the usual LTL-style syntax, and the expression might become less intuitive since we cannot easily refer to a particular time point (by explicitly saying the corresponding time points). Similarly, we also cannot easily specify an arithmetic expression that involves event attributes at two different time points, e.g.,

$$\mathrm{e}[i+1].\,\text{time:timestamp} - \mathrm{e}[i].\,\text{time:timestamp}$$

Therefore, in order to have a more intuitive language, we opt for FOL-style syntax.

3.4.1 Checking whether a closed FOE formula is satisfied

We now proceed to introduce several properties of FOE formulas that are useful for checking whether a trace τ and a prefix length k satisfy a closed FOE formula φ, i.e., to check whether $\tau, k \models \varphi$. This check is needed when we create the prediction model based on the specification of prediction task provided by an analytic rule.

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. In the following, Theorems 1 and 2 show that while checking whether a trace τ and a prefix length k satisfy a closed FOE formula φ, we can eliminate the presence of existential and universal quantifiers.

Theorem 1 *Given a closed FOE formula $\exists i.\varphi$, a trace τ, and a prefix length k,*

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

Proof By the definition of the semantics of FOE, we have that τ and k satisfy $\exists i.\varphi$ (i.e., $\tau, k \models \exists i.\varphi$) iff there exists an index $c \in \{1,\ldots,|\tau|\}$, such that τ and k satisfy the formula ψ that is obtained from φ by substituting each variable i in φ with c (i.e., $\tau, k \models \psi$ where ψ is $\varphi[i \mapsto c]$) Thus, it is the same as satisfying the disjunctions of formulas that is obtained by considering all possible substitutions of the variable i in φ by all possible values of c (i.e., $\bigvee_{c \in \{1,\ldots|\tau|\}} \varphi[i \mapsto c]$). This is the case because such disjunctions of formulas can be satisfied by τ and k if and only if there exists at least one formula in that disjunctions of formulas that is satisfied by τ and k. $\square$

Theorem 2 *Given a closed FOE formula $\forall i.\varphi$, a trace τ, and a prefix length k,*

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

Proof The proof is quite similar to Theorem 1, except that we use the conjunctions of formulas. Basically, we have that τ and k satisfy $\forall i.\varphi$ (i.e., $\tau, k \models \forall i.\varphi$) iff for every $c \in \{1,\ldots,|\tau|\}$, we have that $\tau, k \models \psi$, where ψ is obtained from φ by substituting each variable i in φ with c. In other words, τ and k satisfy each formula that is obtained from φ by considering all possible substitutions of variable i with all possible values of c. Hence, it is the same as satisfying the conjunctions of those formulas (i.e., $\bigwedge_{c \in \{1,\ldots|\tau|\}} \varphi[i \mapsto c]$). This is the case because such conjunctions of formulas can be satisfied by τ and k if and only if each formula in that conjunctions of formulas is satisfied by τ and k. $\square$

To check whether a trace τ and a prefix length k satisfy a closed FOE formula φ, i.e., $\tau, k \models \varphi$, we could perform the following steps:

1. First, we eliminate all quantifiers. This can be done easily by applying Theorems 1 and 2. As a result, each quantified variable will be instantiated with a concrete value;
2. Evaluate all aggregate functions as well as all event attribute accessor expressions based on the event attributes in τ so as to get the actual values of the corresponding event attributes. After this step, we have a formula that is constituted by only concrete values composed by either arithmetic operators (i.e., $+$ or $-$), logical comparison operators (i.e., $==$ or $\neq$), or arithmetic comparison operators (i.e., $<$, $>$, $\leq$, $\geq$, $==$ or $\neq$);
3. Last, we evaluate all arithmetic expressions as well as all expressions involving logical and arithmetic comparison operators. If the whole evaluation gives us true (i.e., $(\varphi)^{\tau,k} = \mathsf{true}$), then we have that $\tau, k \models \varphi$, otherwise $\tau, k \not\models \varphi$ (i.e., τ and k do not satisfy φ).

The existence of this procedure gives us the following theorem:

Theorem 3 *Given a closed FOE formula φ, a trace τ, and a prefix length k, checking whether $\tau, k \models \varphi$ is decidable.*

This procedure has been implemented in our prototype as a part of the mechanism for processing the specification of prediction task while constructing the prediction model.

3.5 Formalizing the analytic rule

With this machinery in hand, we can formally say how to specify condition and target expressions in analytic rules, namely that condition expressions are specified as closed FOE formulas, while target expressions are 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. We require 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*.

We can now formalize the semantics of analytic rules as illustrated in Sect. 3.1. Formally, given a trace τ, a considered prefix length k, and an analytic rule R of the form

$$\begin{aligned} R = \langle\, & \mathsf{Cond}_1 \Longrightarrow \mathsf{Target}_1, \\ & \mathsf{Cond}_2 \Longrightarrow \mathsf{Target}_2, \\ & \quad\vdots \\ & \mathsf{Cond}_n \Longrightarrow \mathsf{Target}_n, \\ & \mathsf{DefaultTarget}\,\rangle, \end{aligned}$$

R maps τ and k into a value obtained from evaluating the corresponding target expression as follows:

$$R(\tau, k) = \begin{cases} (\mathsf{Target}_1)^{\tau,k} & \text{if } \tau, k \models \mathsf{Cond}_1, \\ (\mathsf{Target}_2)^{\tau,k} & \text{if } \tau, k \models \mathsf{Cond}_2, \\ \quad\vdots & \quad\vdots \\ (\mathsf{Target}_n)^{\tau,k} & \text{if } \tau, k \models \mathsf{Cond}_n, \\ (\mathsf{DefaultTarget})^{\tau,k} & \text{otherwise} \end{cases}$$

where $(\mathsf{Target}_i)^{\tau,k}$ is the application of our interpretation function $(\cdot)^{\tau,k}$ to the target expression Target_i in order to evaluate the expression and get the value. Checking whether the given trace τ and the given prefix length k satisfy Cond_i, i.e., $\tau, k \models \mathsf{Cond}_i$, can be done as explained in Sect. 3.4.1. We also require an analytic rule to be well defined, i.e., given a trace τ, a prefix length k, and an analytic rule R, we say that *R is well defined for τ and k* if R maps τ and 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 $(\mathsf{Target}_i)^{\tau,k} = (\mathsf{Target}_j)^{\tau,k}$. This notion of well-definedness can be easily generalized to event logs as follows: Given an event log L and an analytic rule R, we say that *R is well defined for L* if for every possible trace τ in L and every possible prefix length k, we have that R is well defined for τ and k. Note that such condition can be easily checked for the given event log L and an analytic rule R since the event log is finite. This notion of well defined is required in order to guarantee that the given analytic rule R behaves as a function with respect to the given event log L, i.e., R maps every pair of trace τ and prefix length k into a unique value.

Compared to enforcing that each condition in analytic rules must not be overlapped, our notion of well defined gives us more flexibility in making a specification using our language while also guaranteeing reasonable behaviour. For instance, one can specify several characteristics of ping-pong behaviour in a more convenient way by specifying several conditional-target expressions, i.e.,

$$\begin{aligned} & \mathsf{Cond}_1 \Longrightarrow \text{“Ping-Pong”}, \\ & \mathsf{Cond}_2 \Longrightarrow \text{“Ping-Pong”}, \\ & \qquad\quad\vdots \\ & \mathsf{Cond}_m \Longrightarrow \text{“Ping-Pong”} \end{aligned}$$

(where each condition expression Cond_i captures a particular characteristic of a ping-pong behaviour), instead of using disjunctions of these several condition expressions, i.e.,

$$\mathsf{Cond}_1 \vee \mathsf{Cond}_2 \vee \ldots \vee \mathsf{Cond}_m \Longrightarrow \text{“Ping-Pong”}$$

which could end up into a very long specification of a condition expression.

4 Building the prediction model

Once we are able to specify the desired prediction tasks properly, how can we build the corresponding prediction model based on the given specification? In this case, we need a mechanism that fulfils the following requirement: For the given specification of the desired prediction task provided in our language, it should create the corresponding reliable prediction function, which maps each input partial trace into the most probable predicted output.

Given an analytic rule R and an event log L, our aim is to create a prediction function that takes (partial) trace as the input and predict the most probable output value for the given input. To this aim, we train a classification/regression model in which the input is the features that are obtained from the encoding of all possible trace prefixes in the event log L (the training data). 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. There are several ways to encode (partial) traces into input features for training a machine learning model. For instance, [33,60] study various encoding techniques such as index-based encoding and boolean encoding. In [63], the authors use the so-called *one-hot encoding* of event names, and also add some time-related features (e.g., the time increase with respect to the previous event). Some works consider the feature encodings that incorporate the information of the last n-events. There are also several choices on the information to be incorporated. One can incorporate only the name of the events/activities, or one can also incorporate other information (provided by the available event attributes) such as the (human) resource who is in charged in the activity.

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., $\mathsf{enc}(\tau) = \{x_1, \ldots, x_m\}$. Furthermore, since a trace τ might have arbitrary length (i.e., arbitrary number of events), the encoding function must be able to transform these arbitrary number of trace information into a fix number of features. This can be done, for example, by considering the last n-events of the given trace τ or by aggregating the information within the trace itself. In the encoding that incorporates the last n-events, if the number of the events within the trace τ is less than n, then typically we can add 0 for all missing information in order to get a fix number of features.

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, which 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. 6), we provide some encoding functions that can be selected in order to encode a trace.

Algorithm 1 - Procedure for building the prediction model

```
Input:   an analytic rule $R$,
         an event log $L$, and
         a set $\mathsf{Enc} = \{\mathsf{enc}_1, \ldots, \mathsf{enc}_n\}$ of encoding functions
Output:  a prediction function $\mathcal{P}$
1: for each trace $\tau \in L$ do
2:     for each $k$ where $1 < k < |\tau|$ do
3:         $\tau^k_{\text{encoded}} = \mathsf{enc}_1(\tau^k) \cup \ldots \cup \mathsf{enc}_n(\tau^k)$
4:         targetValue = $R(\tau^k)$
5:         add a new training instance for $\mathcal{P}$, where
             $\mathcal{P}(\tau^k_{\text{encoded}})$ = targetValue
6:     end for
7: end for
8: Train the prediction function $\mathcal{P}$ (either classification or regression
   model)
```

Algorithm 1 illustrates our procedure for building the prediction model based on the given inputs, namely: *(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 algorithm works as follows: for each k-length trace prefix τ^k of each trace τ in the event log L (where $1 < k < |\tau|$), we do the following: In line 3, 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. In line 4, we compute the expected prediction result (target value) by applying the analytical rule R to τ^k. In line 5, 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. Finally, we train the prediction function $\mathcal{P}$ and get the desired prediction function.

Observe that the procedure above is independent with respect to the classification/regression model and trace encoding technique that are used. One can plug in different machine learning classification/regression technique as well as use different trace encoding technique in order to get the desired quality of prediction.

We elaborate why our approach fulfils the requirement that is described in the beginning of this section as follows: In our approach, we mainly rely on supervised machine learning technique in order to learn the desired prediction function from the historical business process execution log (event log). Recall that a supervised machine learning approach learns and approximates a function that maps each input to an output based on training data containing examples of input and output pairs. Additionally, recall that a prediction task specification essentially specifies the way to map a partial trace (input) into the corresponding desired prediction result (output). As can be seen from our approach, by using the given prediction task specification and the given event log, we can create a training data containing various examples of inputs and outputs. Therefore, each example of input and output in this training data is generated based on the given prediction task specification. Since the prediction function is built based on this training data (which is created based on the prediction task specification), the way the learned prediction function maps each input to output is influenced by the given specification as well. Thus, the generated prediction function is basically built based on the given specification of prediction task. As for the reliability of the generated prediction functions, when we apply our whole approach in our experiments (cf. Sect. 6), the reliability of the created prediction functions is measured by several metrics, e.g., accuracy, AUC, precision, etc.

5 Showcases and multi-perspective prediction 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., $R_1, R_2, \ldots, R_n$. Given a set $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$ of analytic rules, our approach allows us to construct a prediction model for each analytic rule $R_i \in \mathcal{R}$. By having all of the constructed prediction models where each of them focuses on a particular prediction objective, we can obtain a *multi-perspective prediction analysis service*.

In Sect. 3, we have seen some examples of prediction task specification for predicting the ping-pong behaviour and the remaining processing time. In this section, we present numerous other showcases of prediction task specification using our language. More showcases can be found in Appendix B.

5.1 Predicting unexpected behaviour/situation

We can specify the task for predicting unexpected behaviour by first expressing the characteristics of the unexpected behaviour.

Ping-pong Behaviour. The condition expression Cond_1 (in Sect. 3.1) expresses a possible characteristic of ping-pong behaviour. Another possible characterization of ping-pong behaviour is shown below:

$$\begin{aligned}\mathsf{Cond}_2 = \exists i.\,(&i > \mathsf{curr} \;\wedge\; i+2 \leq \mathsf{last} \;\wedge\\ &\mathbb{e}[i].\text{org:resource} \neq \mathbb{e}[i+1].\text{org:resource} \;\wedge\\ &\mathbb{e}[i].\text{org:resource} == \mathbb{e}[i+2].\text{org:resource} \;\wedge\\ &\mathbb{e}[i].\text{org:group} == \mathbb{e}[i+1].\text{org:group} \;\wedge\\ &\mathbb{e}[i].\text{org:group} == \mathbb{e}[i+2].\text{org:group})\end{aligned}$$

In other word, Cond_2 characterizes the condition where "*an officer transfers a task into another officer of the same group, and then the task is transferred back to 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 create an analytic rule to specify the task for predicting ping-pong behaviour as follows:

$$\begin{aligned}R_3 = \langle\; &\mathsf{Cond}_1 \Longrightarrow \text{“ping-pong”},\\ &\mathsf{Cond}_2 \Longrightarrow \text{“ping-pong”},\\ &\qquad\quad \text{“not ping-pong”}\rangle,\end{aligned}$$

where Cond_1 is the same as specified in Sect. 3.1. During the construction of the prediction model, in the training phase, R_3 maps each trace prefix τ^k that satisfies either Cond_1 or Cond_2 into the target value "ping-pong", and those prefixes that neither satisfy Cond_1 nor Cond_2 into "not ping-pong". After training the model based on this rule, we get a classifier that is trained for distinguishing between (partial) traces that most likely and unlikely lead to ping-pong behaviour. This example also exhibits the ability of our language to specify a behaviour that has multiple characteristics.

Abnormal Activity Duration. The following expression specifies the *existence of abnormal waiting duration* by stating that *there exists a waiting activity in which the duration is more than 2 hours (7.200.000 milliseconds)*:

$$\begin{aligned}\mathsf{Cond}_3 = \exists i.(&i < \mathsf{last}) \;\wedge\\ &\mathbb{e}[i].\text{concept:name} == \text{“Waiting”} \;\wedge\\ &(\mathbb{e}[i+1].\text{time:timestamp} \;-\\ &\qquad \mathbb{e}[i].\text{time:timestamp}) > 7.200.000\end{aligned}$$

As before, we can then specify an analytic rule for predicting whether a (partial) trace is likely to have an abnormal waiting duration or not as follows:

$$R_4 = \langle \mathsf{Cond}_3 \Longrightarrow \text{“Abnormal”},\; \text{“Normal”}\rangle.$$

Applying the approach for constructing the prediction model in Sect. 4, we obtain a classifier that is trained to predict whether a (partial) trace is most likely or unlikely to have an abnormal waiting duration.

5.2 Predicting SLA/business constraints compliance

Using FOE, we can easily specify numerous expressive SLA conditions as well as business constraints. Furthermore, using the approach presented in Sect. 4, we can create the corresponding prediction model, which predicts the compliance of the corresponding SLA/business constraints.

Time-related SLA. Let Cond_4 be the FOE formula in Example 6. Roughly speaking, Cond_4 expresses an SLA stating that *each order that is created will be eventually delivered within 3 hours*. We can then specify an analytic rule for predicting the compliance of this SLA as follows:

$$R_5 = \langle \mathsf{Cond}_4 \Longrightarrow \text{“Comply”},\; \text{“Not Comply”}\rangle.$$

Using R_5, our procedure for constructing the prediction model in Sect. 4 generates a classifier that is trained to predict whether a (partial) trace is likely or unlikely to comply with the given SLA.

Separation of Duties (SoD). We could also specify a constraint concerning *Separation of Duties (SoD)*. For instance, we require that the person who assembles the product is different from the person who checks the product (i.e., quality assurance). This can be expressed as follows:

$$\begin{aligned}\mathsf{Cond}_5 = \forall i.\forall j.((&i < \mathsf{last}) \;\wedge\; (j < \mathsf{last}) \;\wedge\\ &\mathbb{e}[i].\text{concept:name} == \text{“assembling”} \;\wedge\\ &\mathbb{e}[j].\text{concept:name} == \text{“checking”}) \;\rightarrow\\ &\quad(\mathbb{e}[i].\text{org:resource} \;\neq\; \mathbb{e}[j].\text{org:resource}).\end{aligned}$$

Intuitively, Cond_5 states that for every two activities, if they are assembling and checking activities, then the resources who are in charge of those activities must be different. Similar to previous examples, we can specify an analytic rule for predicting the compliance of this constraint as follows:

$$R_6 = \langle \mathsf{Cond}_5 \Longrightarrow \text{“Comply”},\ \text{“Not Comply”} \rangle.$$

Applying our procedure for building the prediction model, we obtain a classifier that is trained to predict whether or not a trace is likely to fulfil this constraint.

Constraint on Activity Duration. Another example would be a constraint on the activity duration, e.g., a requirement which states that *each activity must be finished within 2 hours*. This can be expressed as follows:

$$\begin{aligned}\mathsf{Cond}_6 = \forall i.(i < \mathsf{last}) \rightarrow \\ (\mathsf{e}[i+1].\,\text{time:timestamp} - \\ \mathsf{e}[i].\,\text{time:timestamp}) < 7.200.000.\end{aligned}$$

Cond_6 basically says that *the time difference between two activities is always less than 2 hours (7.200.000 milliseconds)*. An analytic rule to predict the compliance of this SLA can be specified as follows:

$$R_7 = \langle \mathsf{Cond}_6 \Longrightarrow \text{“Comply”},\ \text{“Not Comply”} \rangle.$$

Notice that we can express the same specification in a different way, for instance

$$R_8 = \langle \mathsf{Cond}_7 \Longrightarrow \text{“Not Comply”},\ \text{“Comply”} \rangle$$

where

$$\begin{aligned}\mathsf{Cond}_7 = \exists i.(i < \mathsf{last}) \wedge \\ (\mathsf{e}[i+1].\,\text{time:timestamp} - \\ \mathsf{e}[i].\,\text{time:timestamp}) > 7.200.000.\end{aligned}$$

Essentially, Cond_7 expresses a specification on the *existence of abnormal activity duration*. It states that *there exists an activity in which the time difference between that activity and the next activity is greater than 7.200.000 milliseconds (2 hours)*. Using either R_7 or R_8, our procedure for building the prediction model (cf. Algorithm 1) gives us a classifier that is trained to distinguish between the partial traces that most likely will and will not satisfy this activity duration constraint.

We could even specify a more fine-grained constraint by focusing into a particular activity. For instance, the following expression specifies that *each validation activity must be done within 2 hours (7.200.000 milliseconds)*:

$$\begin{aligned}\mathsf{Cond}_8 = \forall i.((i < \mathsf{last}) \wedge \\ \mathsf{e}[i].\,\text{concept:name} == \text{“Validation”}) \rightarrow \\ (\mathsf{e}[i+1].\,\text{time:timestamp} - \\ \mathsf{e}[i].\,\text{time:timestamp}) < 7.200.000.\end{aligned}$$

Cond_8 basically says that *for each validation activity, the time difference between that activity and its next activity is always less than 2 hours (7.200.000 milliseconds)*. Similar to the previous examples, it is easy to see that we could specify an analytic rule for predicting the compliance of this SLA and create a prediction model that is trained to predict whether a (partial) trace is likely or unlikely fulfilling this SLA.

5.3 Predicting time-related information

In Sect. 3.1, we have seen how we can specify the task for predicting the *remaining processing time* (by specifying a target expression that computes the time difference between the timestamp of the last and the current events). In the following, we provide another examples on predicting time-related information.

Predicting Delay. Delay can be defined as a condition when the actual processing time is longer than the expected processing time. Suppose we have the information about the expected processing time, e.g., provided by an attribute “expectedDuration” of the first event, we can specify an analytic rule for predicting the occurrence of delay as follows:

$$R_9 = \langle \mathsf{Cond}_9 \Longrightarrow \text{“Delay”},\ \text{“Normal”} \rangle.$$

where Cond_9 is specified as follows:

$$\begin{aligned}(\mathsf{e}[\mathsf{last}].\,\text{time:timestamp} - \\ \mathsf{e}[1].\,\text{time:timestamp}) > \mathsf{e}[1].\,\text{expectedDuration}.\end{aligned}$$

Cond_9 states that the difference between the last event timestamp and the first event timestamp (i.e., the processing time) is greater than the expected duration (provided by the value of the event attribute “expectedDuration”). While training the classification model, R_9 maps each trace prefix τ^k into either “Delay” or “Normal” depending on whether the processing time of the whole trace τ is greater than the expected processing time or not.

Predicting the Overhead of Running Time. The *overhead of running time* is the amount of time that exceeds the expected running time. If the actual running time does not go beyond the expected running time, then the overhead is 0. Suppose that the expected running time is 3 hours (10.800.000 milliseconds), the task for predicting the overhead of running time can then be specified as follows:

$$R_{10} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathbf{max}(\mathsf{Overhead}, 0),\ 0 \rangle.$$

where $\mathsf{Overhead} = \mathsf{TotalRunTime} - 10.800.000$, and

$$\begin{aligned}\mathsf{TotalRunTime} = \\ \mathsf{e}[\mathsf{last}].\,\text{time:timestamp} - \mathsf{e}[1].\,\text{time:timestamp}.\end{aligned}$$

In this case, R_{10} computes the difference between the actual total running time and the expected total running time. Moreover, it outputs 0 if the actual total running time is less than the expected total running time, since it takes the maximum value between the computed time difference and 0. Applying our procedure for creating the prediction model, we obtain a regression model that predicts the overhead of running time.

Predicting the Remaining Duration of a Certain Event. Let the duration of an event be the time difference between the timestamp of that event and its succeeding event. The task for predicting the *total duration of all remaining "waiting" events* can be specified as follows:

$$R_{11} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{RemWaitingDur},\ 0\rangle.$$

where $\mathsf{RemWaitingDur}$ is defined as the *sum of the duration of all remaining waiting events*, formally as follows:

$$\begin{aligned}&\textbf{sum}(\mathbb{e}[x+1].\text{time:timestamp} - \mathbb{e}[x].\text{time:timestamp};\\ &\quad \textbf{where}\ x = \mathsf{curr} : \mathsf{last};\\ &\quad \textbf{and}\ \mathbb{e}[x].\text{concept:name} == \text{"waiting"})\end{aligned}$$

As before, based on this rule, we can create a regression model that predicts the total duration of all remaining waiting events.

Predicting Average Activity Duration. We can specify the way to compute the *average of activity duration* as follows:

$$\begin{aligned}&\mathsf{AvgActDur} =\\ &\quad \textbf{avg}(\mathbb{e}[x+1].\text{time:timestamp} - \mathbb{e}[x].\text{time:timestamp};\\ &\qquad \textbf{where}\ x = 1 : \mathsf{last})\end{aligned}$$

where the activity duration is defined as the time difference between the timestamp of that activity and its next activity. We can then specify an analytic rule that expresses the task for predicting the average activity duration as follows:

$$R_{12} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{AvgActDur},\ 0\rangle.$$

Similar to previous examples, applying our procedure for creating the prediction model, we get a regression model that computes the approximation of the average activity duration of a process.

5.4 Predicting workload-related information

Knowing the information about the amount of work to be done (i.e., workload) would be beneficial. Predicting the activity frequency is one of the ways to get an overview of workload. The following task specifies how to predict *the number of the remaining activities* that are necessary to be performed:

$$\begin{aligned}&R_{13} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow\\ &\qquad \textbf{count}(\text{true};\ \textbf{where}\ x = \mathsf{curr} : \mathsf{last}),\ 0\rangle\end{aligned}$$

In this case, R_{13} counts the number of remaining activities. We could also provide a more fine-grained specification by focusing on a certain activity. For instance, in the following we specify the task for predicting *the number of the remaining validation activities* that need to be done:

$$R_{14} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{NumOfRemValidation},\ 0\rangle.$$

where $\mathsf{NumOfRemValidation}$ is specified as follows:

$$\begin{aligned}&\textbf{count}(\mathbb{e}[x].\text{concept:name} == \text{"validation"};\\ &\quad \textbf{where}\ x = \mathsf{curr} : \mathsf{last})\end{aligned}$$

$\mathsf{NumOfRemValidation}$ counts the occurrence of validation activities between the current event and the last event (the occurrence of validation activity is reflected by the fact that the value of the attribute concept:name is equal to "validation"). Applying our procedure for creating the prediction model over R_{13} and R_{14}, consecutively we get regression models that predict the number of remaining activities as well as the number of the remaining validation activities.

We could also classify a process into complex or normal based on the frequency of a certain activity. For instance, we could consider a process that requires more than 25 validation activities as complex (otherwise it is normal). The following analytic rule specifies this task:

$$R_{15} = \langle \mathsf{Cond}_{10} > 25 \Longrightarrow \text{"complex"},\ \text{"normal"}\rangle$$

where Cond_{10} is specified as follows:

$$\begin{aligned}&\textbf{count}(\mathbb{e}[x].\text{concept:name} == \text{"validation"};\\ &\quad \textbf{where}\ x = 1 : \mathsf{last})\end{aligned}$$

Based on R_{15}, we could train a model to classify whether a (partial) trace is likely to be a complex or a normal process.

5.5 Predicting resource-related information

Human resources could be a crucial factor in the process execution. Knowing the number of different resources that are needed for handling a process could be beneficial. The following analytic rule specifies the task for predicting the number of different resources that are required:

$$\begin{aligned}&R_{16} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow\\ &\qquad \textbf{countVal}(\text{org:resource};\ \textbf{within}\ 1 : \mathsf{last}),\ 0\rangle.\end{aligned}$$

During the training phase, since

$$\textbf{countVal}(\text{org:resource};\ \textbf{within}\ 1 : \mathsf{last})$$

is evaluated to the number of different values of the attribute org:resource within the corresponding trace, R_{16} maps each trace prefix τ^k into the number of different resources.

To predict the number of *task handovers* among resources, we can specify the following prediction task:

$$R_{17} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{NumHandovers},\ 0\rangle.$$

where NumHandovers is defined as follows:

$$\textbf{count}(\mathsf{e}[x].\text{org:resource} \neq \mathsf{e}[x+1].\text{org:resource};\ ,\ \textbf{where}\ x = 1 : \mathsf{last})$$

i.e., NumHandovers counts the number of changes on the value of the attribute org:resource and the changes of resources reflect the task handovers among resources. Thus, in this case, R_{17} maps each trace prefix τ^k into the number of task handovers.

A process can be considered as *labour intensive* if it involves at least a certain number of different resources, e.g., three different number of resources. This kind of task can be specified as follows:

$$R_{18} = \langle \mathsf{Cond}_{11} \Longrightarrow \text{“LaborIntensive”},\ \text{“normal”}\rangle$$

where Cond_{11} is as follows:

$$\begin{aligned}\exists i.\exists j.\exists k.\ (i \neq j\ \wedge\ i \neq k\ \wedge\ j \neq k\ \wedge\\ \mathsf{e}[i].\text{org:resource} \neq \mathsf{e}[j].\text{org:resource}\ \wedge\\ \mathsf{e}[i].\text{org:resource} \neq \mathsf{e}[k].\text{org:resource}\ \wedge\\ \mathsf{e}[j].\text{org:resource} \neq \mathsf{e}[k].\text{org:resource})\end{aligned}$$

Essentially, Cond_{11} states that there are at least three different events in which the values of the attribute org:resource in those events are different.

5.6 Predicting value-added related information

Value-added analysis in BPM aims at identifying *unnecessary steps* within a process with the purpose of eliminating those steps [22]. The steps within a process can be classified into either *value-added*, *business value-added*, or *non-value added*. Value-added steps are the steps that produce/add values to the customer or to the final outcome/product of the process. Business value-added steps are those steps that might not directly add value to the customer but they might be necessary or useful for the business. Non-value-added steps are those that are neither value-added nor business value-added. Based on this analysis, typically we would like to retain those value-added steps, and eliminate (or reduce) those non-value-added steps. These non-value-added steps are often also associated with wastes. An example of waste is *overprocessing waste*. An overprocessing waste occurs when an activity is performed unnecessarily with respect to the outcome of a process (i.e., it is performed in a way that *does not add value* to the final outcome/product of the process). It includes tasks that are performed but later found to be unnecessary. In this light, the work by [73] proposes an approach for minimizing the overprocessing waste by employing prediction techniques.

Using our language, we can characterize prediction tasks that could assist in minimizing overprocessing waste. For instance, consider the process of handing personal loan applications. Suppose that there are several checking activities that need to be performed on each application, and these checks can be performed in any order (each check is independent of each other). Failure in any of these checks would make the application rejected and the process stops immediately. Only cases that successfully pass all checks are accepted. This kind of process is often called a *knockout process* [65]. For instance, let these checking activities be *(i) Applicant's Identity Check* (*IDC*); *(ii) Loan Worthiness Assessment* (*LWA*); and *(iii) Applicant's Documents Verification* (*DV*). In this scenario, if we first perform the *Identity Check* activity and then the *Documents Verification* activity, but the *Document Verification* gives negative outcome (i.e., the application is rejected), then the efforts that we have spent on performing the *Identity Check* activity would be a waste. Thus, it might be desirable to perform the *Document Verification* step first so that we could reject the application immediately before spending any efforts on any other activities.

In the situation above, it might be desirable to predict the checking activity that most likely would lead to a negative outcome for a certain case of a loan application. By doing this, we can prioritize that activity and reduce the waste of efforts that we spend on unnecessary activities. The analytic rule below specifies the prediction task that *predicts the activity that most likely would lead to the negative outcome* (i.e., the rejection of the application).

$$\begin{aligned}R_{32} = \langle\ &\mathsf{Cond}_{16} \Longrightarrow \text{“Identity Check (IDC)”},\\ &\mathsf{Cond}_{17} \Longrightarrow \text{“Loan Worthiness Assesment (LWA)”},\\ &\mathsf{Cond}_{18} \Longrightarrow \text{“Document Verification (DV)”},\\ &\qquad\quad \text{“None”}\rangle,\end{aligned}$$

where

$$\begin{aligned}\mathsf{Cond}_{16} = \exists i.(\ i > \mathsf{curr}\ \wedge\ \mathsf{e}[i].\text{concept:name} == \text{“IDC”}\ \wedge\\ \mathsf{e}[i+1].\text{concept:name} == \text{“Reject App.”}),\end{aligned}$$

$$\begin{aligned}\mathsf{Cond}_{17} = \exists i.(\ i > \mathsf{curr}\ \wedge\ \mathsf{e}[i].\text{concept:name} == \text{“LWA”}\ \wedge\\ \mathsf{e}[i+1].\text{concept:name} == \text{“Reject App.”}),\end{aligned}$$

$$\begin{aligned}\mathsf{Cond}_{18} = \exists i.(\ i > \mathsf{curr}\ \wedge\ \mathsf{e}[i].\text{concept:name} == \text{“DV”}\ \wedge\\ \mathsf{e}[i+1].\text{concept:name} == \text{“Reject App.”}).\end{aligned}$$

As observed by [65], another aspect that can be considered in prioritizing the activities in this kind of process is the effort that needs to be spent on each activity. This effort could be either in terms of the *processing time* that is needed to perform the activity, the *cost* that needs to be spent, or the *amount of resources* that needs to be allocated. An activity is expensive if it requires a lot of efforts to perform (e.g., an activity that requires a large amount of time). By knowing this information, we could give a priority to the activity that is less expensive, and leave those expensive activities to the last so that they are only performed when they are really necessary. For each checking activity, we can specify the corresponding prediction task that predicts the required effort for performing that activity. For instance, we can specify the task for predicting *the duration of the identity checking (IDC) activity* as follows:

$$R_{33} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{DurationIDC},\ 0\rangle.$$

where DurationIDC is defined as follows:

$$\begin{aligned}&\textbf{sum}(\mathrm{e}[x+1].\text{time:timestamp} - \mathrm{e}[x].\text{time:timestamp};\\ &\quad \textbf{where } x = \mathsf{curr} : \mathsf{last};\\ &\quad \textbf{and } \mathrm{e}[x].\text{concept:name} == \text{“IDC”})\end{aligned}$$

Similarly, we can specify the tasks for predicting *the duration of the Loan Worthiness Assessment (LWA) activity* and *the duration of the Document Verification (DV) activity* as follows:

$$R_{34} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{DurationLWA},\ 0\rangle.$$

$$R_{35} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{DurationDV},\ 0\rangle.$$

where DurationLWA and DurationDV are defined as follows:

$$\begin{aligned}&\textbf{sum}(\mathrm{e}[x+1].\text{time:timestamp} - \mathrm{e}[x].\text{time:timestamp};\\ &\quad \textbf{where } x = \mathsf{curr} : \mathsf{last};\\ &\quad \textbf{and } \mathrm{e}[x].\text{concept:name} == \text{“LWA”})\end{aligned}$$

$$\begin{aligned}&\textbf{sum}(\mathrm{e}[x+1].\text{time:timestamp} - \mathrm{e}[x].\text{time:timestamp};\\ &\quad \textbf{where } x = \mathsf{curr} : \mathsf{last};\\ &\quad \textbf{and } \mathrm{e}[x].\text{concept:name} == \text{“DV”})\end{aligned}$$

As we have seen, all information from all of these simple predictions could, in some sense, be combined/used in order to help to minimize the unnecessary steps in a process, which is one of the goals of value-added analysis.

6 Implementation and experiments

As a proof of concept, we develop a prototype that implements our approach [3] This prototype includes a parser for our language and a program for automatically processing the given prediction task specification as well as for building the corresponding prediction model based on our approach explained in Sects. 3 and 4. We also build a ProM[4] plug-in that wraps these functionalities. Several feature encoding functions to be selected are also provided, e.g., one-hot encoding of event attributes, time since the previous event, plain attribute values encoding, etc. We can also choose the desired machine learning model to be built. Our implementation uses Java and Python. For the interaction between Java and Python, we use Jep (Java Embedded Python).[5] In general, we use Java for implementing the program for processing the specification and we use Python for dealing with the machine learning models.

The main goal of our experiments is to demonstrate the applicability of our whole approach in a real-world setting by applying them into some case studies over real-life data, and to demonstrate the applicability of our approach in automatically constructing reliable prediction models based on the given specification. This includes the demonstration of the capability of our language in specifying relevant interesting prediction tasks based on the case study over real-life data. Basically, our experiments answer the following questions:

1. Is the whole proposed approach applicable in practice (in a real-life setting)?
2. In practice, can we generate reliable prediction models by following our approach (starting from specifying the desired prediction task)?
3. What are the factors that influence the quality of the generated prediction models?

The experiments were conducted by applying our approach into several case studies/problems that are based on real-life event logs. Particularly, we use the publicly available event logs that were provided for Business Process Intelligence Challenge (BPIC) 2012, BPIC 2013, and BPIC 2015. For each event log, several relevant prediction tasks are formulated based on the corresponding domain of the corresponding event log, and also by considering the available information. For instance, predicting the occurrence of ping-pong behaviour among support groups might be suitable for the BPIC 13 event log, but not for BPIC 12 event log since

[3] More information about the implementation architecture, the code, the tool, and the screencast can be found at http://bit.ly/sdprom2.

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

[5] Jep—https://pypi.org/project/jep/.

there is no information about groups in BPIC 12 event log (in fact, they are event logs from two different domains). For each prediction task, we provide the corresponding formal specification that can be fed into our tool in order to create the corresponding prediction model.

For the experiment, we follow the standard *holdout method* [31]. Specifically, we partition the data into two sets as follows: We use the first two-thirds of the log for the training data and the last one-third of the log for the testing data. For each prediction task specification, we apply our approach in order to generate the corresponding prediction model, and then we evaluate the prediction quality of the generated prediction model by considering each k-length trace prefix τ^k of each trace τ in the testing set (for $1 < k < |\tau|$). In order to provide a baseline, we use a statistical-based prediction technique, which is often called Zero Rule (ZeroR). Specifically, for the classification task, the prediction by ZeroR is performed based on the most common target value in the training set, while for the regression task, the prediction is based on the mean value of the target values in the training data. Although ZeroR seems to be quite naive, depending on the data, in some cases it can outperform some advanced machine learning models. The datasets and our source code are available online so as to enable the replication of the experiments [3].

Recall that our approach for building the prediction model presented in Sect. 4 is independent with respect to the supervised machine learning model that is used. Within these experiments, we consider several machine learning models, namely *(i)* Logistic Regression, *(ii)* Linear Regression, *(iii)* Naive Bayes Classifier, *(iv)* Decision Tree [10], *(v)* Random Forest [9], *(vi)* AdaBoost [27] with Decision Tree as the base estimator, *(vii)* Extra Trees [29], *(viii)* Voting Classifier that is composed of Decision Tree, Random Forest, AdaBoost, and Extra Trees. Among these, logistic regression, naive Bayes, and voting classifier are only used for classification tasks, and linear regression is only used for regression tasks. The rest are used for both. Notably, we also use a deep learning approach [30]. In particular, we use the deep feed-forward neural network and we consider various sizes of the network by taking into account several different depth and width of the network. We consider different numbers of hidden layers ranging from 2 to 6 and three variants of the number of neurons, namely 75, 100, and 150. For compactness of the presentation, we do not show the evaluation results on all of these network variations but we only show the best result. The detail results are available at http://bit.ly/sdprom2 as a supplementary material. In the implementation, we use the machine learning libraries provided by *scikit-learn* [46]. For the implementation of neural network, we use *Keras*[6] with *Theano* [64] backend.

[6] https://keras.io.

To assess the prediction quality, we use the standard metrics for evaluating classification and regression models that are generally used in the machine learning literature. These metrics are also widely used in many works in this research area (e.g., [33,35,37,63,69,72]). For the classification task, we use accuracy, area under the ROC curve (AUC), precision, recall, and F-measure. For the regression task, we use mean absolute error (MAE) and root mean square error (RMSE). In the following, we briefly explain these metrics. A more elaborate explanation on these metrics can be found in the typical literature on machine learning and data mining, e.g., [28,31,43].

Accuracy is the fraction of predictions that are correct. It is computed by dividing the number of correct predictions by the number of all predictions. The range of accuracy value is between 0 and 1. The value 1 indicates the best model, while 0 indicates the worst model. An *ROC (receiver operating characteristic) curve* allows us to visualize the prediction quality of a classifier. If the classifier is good, the curve should be as closer to the top left corner as possible. A random guessing is depicted as a straight diagonal line. Thus, the closer the curve to the straight diagonal line, the worse the classifier. The value of the *area under the ROC curve (AUC)* allows us to assess a classifier as follows: The AUC value equal to 1 shows a perfect classifier, while the AUC value equal to 0.5 shows the worst classifier that is not better than random guessing. Thus, the closer the value to 1, the better it is, and the closer the value to 0.5, the worse it is. *Precision* measures the *exactness* of the prediction. When a classifier predicts a certain output for a certain case, the precision value intuitively indicates how much is the chance that such prediction is correct. Specifically, among all cases that are classified into a particular class, precision measures the fraction of those cases that are correctly classified. On the other hand, *recall* measures the *completeness* of the prediction. Specifically, among all cases that should be classified as a particular class, recall measures the fraction of those cases that can be classified correctly. Intuitively, given a particular class, the recall value indicates the ability of the model to correctly classify all cases that should be classified into that particular class. The best precision and recall value is 1. *F-Measure* is harmonic mean of precision and recall. It provides a measurement that combines both precision and recall values by also giving equal weight to them. Formally, it is computed as follows: F-Measure $= (2 \times P \times R)/(P + R)$, where P is precision and R is recall. The best F-measure value is 1. Thus, the closer the value to 1, the better it is.

MAE computes the average of the absolute error of all predictions over the whole testing data, where each error is computed as the difference between the expected and the predicted values. Formally, given n testing data, $MAE = \left(\sum_{i=1}^{n} |y_i - \hat{y}_i|\right)/n$, where $\hat{y}_i$ (resp. y_i) is the predicted value (resp. the expected/actual value) for the testing instance i. *RMSE* can be computed as follows: $RMSE = \sqrt{\left(\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right)/n}$, where $\hat{y}_i$ (resp. y_i) is the predicted value (resp. the expected/actual value) for the testing instance i. Compared to MAE, RMSE is more sensitive to errors since it gives larger penalty to larger errors by using the 'square' operation. For both MAE and RMSE, the lower the score, the better the model.

In our experiments, we use the trace encoding that incorporates the information of the last n-events, where n is the maximal length of the traces in the event log under consideration. Furthermore, for each experiment we consider two types of encoding, where each of them considers different available event attributes (one encoding incorporates more event attributes than the others). The detail of event attributes that are considered is explained in each experiment below.

6.1 Experiment on BPIC 2013 event log

The event log from BPIC 2013[7] [62] contains the data from the Volvo IT incident management system called VINST. It stores information concerning the incidents handling process. For each incident, a solution should be found as quickly as possible so as to bring back the service with minimum interruption to the business. It contains 7554 traces (process instances) and 65533 events. There are also several attributes in each event containing various information such as the problem status, the support team (group) that is involved in handling the problem, the person who works on the problem, etc.

In BPIC 2013, ping-pong behaviour is one of the interesting problems to be analyzed. Ideally, an incident should be solved quickly without involving too many support teams. To specify the tasks for predicting whether a process would probably exhibit a ping-pong behaviour, we first identify and express the possible characteristics of ping-pong behaviour as follows:

$$\begin{aligned}\mathsf{Cond}_{\mathrm{E1}} = \exists i.(&i > \mathsf{curr} \ \wedge \ i < \mathsf{last} \ \wedge \\ &\mathsf{e}[i].\text{org:group} \neq \mathsf{e}[i+1].\text{org:group} \ \wedge \\ &\mathsf{e}[i].\text{concept:name} \neq \text{“Queued”})\end{aligned}$$

$$\begin{aligned}\mathsf{Cond}_{\mathrm{E2}} = \exists i.(&\mathsf{e}[i].\text{org:resource} \neq \mathsf{e}[i+1].\text{org:resource} \ \wedge \\ &\mathsf{e}[i].\text{org:resource} == \mathsf{e}[i+2].\text{org:resource})\end{aligned}$$

$$\begin{aligned}\mathsf{Cond}_{\mathrm{E3}} = \exists i.(&\mathsf{e}[i].\text{org:resource} \neq \mathsf{e}[i+1].\text{org:resource} \ \wedge \\ &\mathsf{e}[i].\text{org:resource} == \mathsf{e}[i+3].\text{org:resource})\end{aligned}$$

$$\begin{aligned}\mathsf{Cond}_{\mathrm{E4}} = \exists i.(&\mathsf{e}[i].\text{org:group} \neq \mathsf{e}[i+1].\text{org:group} \ \wedge \\ &\mathsf{e}[i].\text{org:group} == \mathsf{e}[i+2].\text{org:group})\end{aligned}$$

$$\begin{aligned}\mathsf{Cond}_{\mathrm{E5}} = \exists i.(&\mathsf{e}[i].\text{org:group} \neq \mathsf{e}[i+1].\text{org:group} \ \wedge \\ &\mathsf{e}[i].\text{org:group} == \mathsf{e}[i+3].\text{org:group})\end{aligned}$$

$$\begin{aligned}\mathsf{Cond}_{\mathrm{E6}} = \exists i.\exists j.\exists k.(&i < j \ \wedge \ j < k \ \wedge \\ &\mathsf{e}[i].\text{org:group} \neq \mathsf{e}[j].\text{org:group} \ \wedge \\ &\mathsf{e}[i].\text{org:group} \neq \mathsf{e}[k].\text{org:group} \ \wedge \\ &\mathsf{e}[j].\text{org:group} \neq \mathsf{e}[k].\text{org:group})\end{aligned}$$

Roughly speaking, $\mathsf{Cond}_{\mathrm{E1}}$ says that *there is a change in the support team while the problem is not being "Queued"*. $\mathsf{Cond}_{\mathrm{E2}}$ and $\mathsf{Cond}_{\mathrm{E3}}$ state that *there is a change in the person who handles the problem, but then at some point it changes back into the original person*. $\mathsf{Cond}_{\mathrm{E4}}$ and $\mathsf{Cond}_{\mathrm{E5}}$ say that *there is a change in the support team (group) who handles the problem, but then at some point it changes back into the original support team*. $\mathsf{Cond}_{\mathrm{E6}}$ states that *the process of handling the incident involves at least three different groups*.

We then specify three different analytic rules below in order to specify three different tasks for predicting ping-pong behaviour based on various characteristics of this unexpected behaviour.

$$R_{\mathrm{E1}} = \langle \mathsf{Cond}_{\mathrm{E1}} \Longrightarrow \text{“Ping-Pong”}, \ \text{“Not Ping-Pong”} \rangle$$

$$\begin{aligned}R_{\mathrm{E2}} = \langle \ &\mathsf{Cond}_{\mathrm{E2}} \Longrightarrow \text{“Ping-Pong”}, \\ &\mathsf{Cond}_{\mathrm{E3}} \Longrightarrow \text{“Ping-Pong”}, \\ &\mathsf{Cond}_{\mathrm{E4}} \Longrightarrow \text{“Ping-Pong”}, \\ &\mathsf{Cond}_{\mathrm{E5}} \Longrightarrow \text{“Ping-Pong”}, \\ &\qquad\qquad \text{“Not Ping-Pong”} \ \rangle,\end{aligned}$$

$$R_{\mathrm{E3}} = \langle \mathsf{Cond}_{\mathrm{E6}} \Longrightarrow \text{“Ping-Pong”}, \ \text{“Not Ping-Pong”} \rangle$$

In this case, R_{E1} specifies the task for predicting ping-pong behaviour based on the characteristic provided by $\mathsf{Cond}_{\mathrm{E1}}$ (similarly for R_{E2} and R_{E3}). These analytic rules can be fed into our tool in order to obtain the prediction model, and for these cases we create classification models.

In BPIC 2013 event log, an incident can have several statuses. One of them is waiting. In this experiment, we predict the *remaining duration of all waiting-related events* by specifying the following analytic rule:

$$R_{\mathrm{E4}} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{RemWaitingTime}, 0 \rangle$$

where $\mathsf{RemWaitingTime}$ is as follows:

$$\begin{aligned}&\textbf{sum}(\mathsf{e}[x+1].\text{time:timestamp} - \mathsf{e}[x].\text{time:timestamp}; \\ &\quad \textbf{where } x = \mathsf{curr} : \mathsf{last}; \\ &\quad \textbf{and } \ \mathsf{e}[x].\text{lifecycle:transition} == \text{“Await. Assign.”} \ \vee \\ &\qquad\quad \mathsf{e}[x].\text{lifecycle:transition} == \text{“Wait”} \ \vee \\ &\qquad\quad \mathsf{e}[x].\text{lifecycle:transition} == \text{“Wait - Impl.”} \ \vee \\ &\qquad\quad \mathsf{e}[x].\text{lifecycle:transition} == \text{“Wait - User”} \ \vee \\ &\qquad\quad \mathsf{e}[x].\text{lifecycle:transition} == \text{“Wait - Cust.”} \ \vee \\ &\qquad\quad \mathsf{e}[x].\text{lifecycle:transition} == \text{“Wait - Vendor”})\end{aligned}$$

[7] More information on BPIC 2013 can be found in http://www.win.tue.nl/bpi/doku.php?id=2013:challenge.

i.e., RemWaitingTime is the sum of all event duration in which the status is related to waiting (e.g., Awaiting Assignment, Wait, Wait-User, etc). Similarly, we predict the *remaining duration of all (exactly) waiting events* by specifying the following:

$$R_{E5} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{RemWaitDur}, 0\rangle$$

where RemWaitDur is as follows:

$$\begin{aligned}&\mathbf{sum}(\mathrm{e}[x+1].\,\text{time:timestamp} - \mathrm{e}[x].\,\text{time:timestamp};\\ &\quad \mathbf{where}\ x = \mathsf{curr} : \mathsf{last};\\ &\quad \mathbf{and}\ \mathrm{e}[x].\,\text{lifecycle:transition} == \text{“Wait”})\end{aligned}$$

i.e., RemWaitDur is the sum of all event duration in which the status is “wait”. Both R_{E4} and R_{E5} can be fed into our tool, and in this case we generate regression models.

For all of these tasks, we consider two different trace encodings. First, we use the trace encoding that incorporates several available event attributes, namely concept:name, org:resource, org:group, lifecycle:transition, organization involved, impact, product, resource country, organization country, org:role. Second, we use the trace encoding that only incorporates the event names, i.e., the values of the attribute concept:name. Intuitively, the first encoding considers more information than the second encoding. Thus, the prediction models that are obtained by using the first encoding use more input information for doing the prediction. The evaluation on the generated prediction models from all prediction tasks specified above is reported in Tables 1 and 2.

6.2 Experiment on BPIC 2012 event log

The event log for BPIC 2012[8] [70] comes from a Dutch financial institute. It stores the information concerning the process of handling either personal loan or overdraft application. It contains 13.087 traces (process instances) and 262.200 events. Generally, the process of handling an application is as follows: Once an application is submitted, some checks are performed. After that, the application is augmented with necessary additional information that is obtained by contacting the client by phone. An offer will be send to the client, if the applicant is eligible. After this offer is received back, it is assessed. The customer will be contacted again if there is a missing information. After that, a final assessment is performed. In this experiment, we consider two prediction task as follows:

1. One type of activity within this process is named *W_Completeren aanvraag*, which stands for “Filling in information for the application”. The task for predicting the *total duration of all remaining activities of this type* is formulated as follows:

   $$R_{E6} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{RemTimeFillingInfo}, 0\rangle$$

   where RemTimeFillingInfo is as follows:

   $$\begin{aligned}&\mathbf{sum}(\mathrm{e}[x+1].\,\text{time:timestamp}-\mathrm{e}[x].\,\text{time:timestamp};\\ &\quad \mathbf{where}\ x = \mathsf{curr} : \mathsf{last};\\ &\quad \mathbf{and}\ \mathrm{e}[x].\,\text{concept:name} ==\\ &\qquad\qquad \text{“W\_Completeren aanvraag”}),\end{aligned}$$

   i.e., it computes the sum of the duration of all remaining *W_Completeren aanvraag* activities.
2. At the end of the process, an application can be declined. The task to predict whether an application will eventually be declined is specified as follows:

   $$R_{E7} = \langle \mathsf{Cond}_{E8} \Longrightarrow \text{“Declined, “Not\_Declined}\rangle$$

   where Cond_{E8} is as follows:

   $$\begin{aligned}\mathsf{Cond}_{E8} = \exists i.(i > \mathsf{curr}\ \wedge\\ \mathrm{e}[i].\,\text{concept:name} == \text{“A\_DECLINED”}),\end{aligned}$$

   i.e., Cond_{E8} says that eventually there will be an event in which the application is declined.

Both R_{E6} and R_{E7} can be fed into our tool. For R_{E6}, we generate a regression model, while for R_{E7}, we generate a classification model. Different from the BPIC 2013 and BPIC 2015 event logs, there are not so many event attributes in this log. For all of these tasks, we consider two different trace encodings. First, we use the trace encoding that incorporates several available event attributes, namely concept:name and lifecycle:transition. Second, we use the trace encoding that only incorporates the event names, i.e., the values of the attribute concept:name. Thus, intuitively the first encoding considers more information than the second encoding. The evaluation on the generated prediction models from the prediction tasks specified above is shown in Tables 3 and 4.

6.3 Experiment on BPIC 2015 event log

In BPIC 2015[9] [71], 5 event logs from 5 Dutch Municipalities are provided. They contain the data of the processes for handling the building permit application. In general, the processes in these 5 municipalities are similar. Thus, in this experiment we only consider one of these logs.

[8] More information on BPIC 2012 can be found in http://www.win.tue.nl/bpi/doku.php?id=2012:challenge.

[9] More information on BPIC 2015 can be found in http://www.win.tue.nl/bpi/doku.php?id=2015:challenge.

Table 1 The results from the experiments on BPIC 2013 event log using prediction tasks R_{E1}, R_{E2}, and R_{E3}

Model	First encoding (more features)					Second encoding (less features)				
	AUC	Accuracy	W. Prec	W. Rec	F-Measure	AUC	Accuracy	W. Prec	W. Rec	F-Measure
Experiments with the analytic rule R_{E1} (change of group while the concept:name is not 'queued')										
R_{E1}										
ZeroR	0.50	0.82	0.68	0.82	0.75	0.50	0.82	0.68	0.82	0.75
Logistic Reg.	0.64	0.81	0.75	0.81	0.76	0.55	0.82	0.68	0.82	0.75
Naive Bayes	0.51	0.21	0.80	0.21	0.12	0.54	0.19	**0.79**	0.19	0.09
Decision Tree	0.67	0.78	0.80	0.78	0.79	**0.68**	0.82	0.76	0.82	**0.77**
Random Forest	**0.83**	**0.84**	**0.83**	**0.84**	**0.83**	**0.68**	0.82	0.76	0.82	**0.77**
AdaBoost	0.73	0.81	0.77	0.81	0.78	0.66	0.82	0.75	0.82	0.75
Extra Trees	0.81	0.83	0.81	0.83	0.82	**0.68**	0.82	0.76	0.82	**0.77**
Voting	0.81	0.81	0.81	0.81	0.81	**0.68**	0.82	0.76	0.82	**0.77**
Deep Neural Net.	0.73	0.83	0.81	0.83	0.81	**0.68**	**0.83**	0.78	**0.83**	0.75
Experiments with the analytic rule R_{E2} (change of people/group and change back to the original person/group)										
R_{E2}										
ZeroR	0.50	0.79	0.63	0.79	0.70	0.50	0.79	0.63	0.79	0.70
Logistic Reg.	0.77	0.82	0.80	0.82	0.80	0.62	0.81	0.78	0.81	0.76
Naive Bayes	0.69	0.79	0.75	0.79	0.75	0.63	0.80	0.77	0.80	0.76
Decision Tree	0.73	0.82	0.82	0.82	0.82	0.76	0.82	0.80	0.82	0.80
Random Forest	**0.85**	**0.86**	0.85	**0.86**	0.85	**0.78**	0.82	0.80	0.82	0.80
AdaBoost	0.81	0.84	0.83	0.84	0.83	0.68	0.81	0.79	0.81	0.77
Extra Trees	**0.85**	**0.86**	0.85	**0.86**	**0.86**	**0.78**	0.82	0.80	0.82	0.80
Voting	**0.85**	**0.86**	0.85	**0.86**	0.85	0.77	0.82	0.81	0.82	**0.81**
Deep Neural Net.	0.77	**0.86**	**0.86**	**0.86**	0.85	**0.78**	**0.83**	**0.82**	**0.83**	0.80
Experiments with the analytic rule R_{E3} (involves at least three different groups)										
R_{E3}										
ZeroR	0.50	0.74	0.54	0.74	0.63	0.50	0.74	0.54	0.74	0.63
Logistic Reg.	0.78	0.78	0.76	0.78	0.76	0.77	0.79	0.77	0.79	0.77
Naive Bayes	0.75	0.76	0.73	0.76	0.70	0.76	0.77	0.75	0.77	0.73
Decision Tree	0.79	0.82	0.83	0.82	0.83	0.81	0.82	**0.82**	0.82	**0.82**
Random Forest	**0.92**	**0.87**	**0.87**	**0.87**	**0.87**	**0.83**	0.82	**0.82**	0.82	**0.82**
AdaBoost	0.89	0.86	0.86	0.86	0.86	**0.83**	0.81	0.80	0.81	0.80
Extra Trees	0.91	**0.87**	**0.87**	**0.87**	**0.87**	0.82	0.82	**0.82**	0.82	**0.82**
Voting	0.91	0.85	0.85	0.85	0.85	0.82	0.82	0.81	0.82	**0.82**
Deep Neural Net.	0.85	0.85	0.84	0.85	0.84	**0.83**	**0.83**	**0.82**	**0.83**	**0.82**

The bold values indicate the best values that we obtained with respect to the corresponding metric. The best value is not always the highest value. For example, when the metric measures the error (e.g., MAE or RMSE), the smallest value is the best value. However, for the metrics such as accuracy, AUC, precision, recall, and F-measure, the highest value is the best value

There are several information available such as the activity name and the resource/person that carried out a certain task/activity. The statistic about the log that we consider is as follows: It has 1409 traces (process instances) and 59681 events.

For this event log, we consider several tasks related to predicting workload-related information (i.e., related to the amount of work/activities need to be done). First, we deal with the task for predicting whether a process of handling an application is complex or not based on the number of the remaining different activities that need to be done. Specifically, we consider a process is complex (or need more attention) if there are still more than 25 different activities need to be done. This task can be specified as follows:

$$R_{E8} = \langle \mathsf{NumDifRemAct} \geq 25 \Longrightarrow \text{“Complex”}, \text{“Normal”} \rangle$$

where NumDifRemAct is specified as follows:

$$\textbf{countVal}(\text{activityNameEN};\ \textbf{within}\ \mathsf{curr} : \mathsf{last}),$$

Table 2 The results of the experiments on BPIC 2013 event log using prediction tasks R_{E4} and R_{E5}

Model	First Encoding (more features)		Second Encoding (less features)	
	MAE (in days)	RMSE (in days)	MAE (in days)	RMSE (in days)
Experiments with the analytic rule R_{E4} (the remaining duration of all waiting-related events)				
R_{E4}				
ZeroR	5.977	6.173	5.977	6.173
Linear Reg.	5.946	6.901	6.16	6.462
Decision Tree	5.431	17.147	5.8	7.227
Random Forest	4.808	8.624	5.81	7.114
AdaBoost	14.011	18.349	14.181	15.164
Extra Trees	4.756	8.612	5.799	7.132
Deep Neural Net.	**2.205**	**4.702**	**4.064**	**4.596**
Experiments with the analytic rule R_{E5} (the remaining duration of all events in which the status is "wait")				
R_{E5}				
ZeroR	1.061	**1.164**	1.061	1.164
Linear Reg.	1.436	1.974	1.099	1.233
Decision Tree	0.685	5.165	1.003	1.66
Random Forest	0.713	3.396	1.016	1.683
AdaBoost	1.507	3.89	1.044	1.537
Extra Trees	0.843	3.719	1.005	1.649
Deep Neural Net.	**0.37**	2.037	**0.683**	**0.927**

The bold values indicate the best values that we obtained with respect to the corresponding metric. The best value is not always the highest value. For example, when the metric measures the error (e.g., MAE or RMSE), the smallest value is the best value. However, for the metrics such as accuracy, AUC, precision, recall, and F-measure, the highest value is the best value

Table 3 The results of the experiments on BPIC 2012 event log using the prediction task R_{E6}

Model	First Encoding (more features)		Second Encoding (less features)	
	MAE (in days)	RMSE (in days)	MAE (in days)	RMSE (in days)
Experiments with the analytic rule R_{E6} (total duration of all remaining activities named 'W_Completeren aanvraag')				
R_{E6}				
ZeroR	3.963	5.916	3.963	5.916
Linear Reg.	3.613	5.518	3.677	5.669
Decision Tree	2.865	5.221	2.876	5.228
Random Forest	2.863	5.198	2.877	5.213
AdaBoost	3.484	5.655	3.484	5.655
Extra Trees	2.857	**5.185**	2.868	**5.191**
Deep Neural Net.	**2.487**	5.683	**2.523**	5.667

The bold values indicate the best values that we obtained with respect to the corresponding metric. The best value is not always the highest value. For example, when the metric measures the error (e.g., MAE or RMSE), the smallest value is the best value. However, for the metrics such as accuracy, AUC, precision, recall, and F-measure, the highest value is the best value

i.e., NumDifRemAct counts the number of different values of the attribute 'activityNameEN' from the current time point until the end of the process. As the next workload-related prediction task, we specify the task for predicting the number of remaining events/activities as follows:

$$R_{E9} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{RemAct}, 0 \rangle$$

where $\mathsf{RemAct} = \mathbf{count}(\mathsf{true};\ \mathbf{where}\ x = \mathsf{curr} : \mathsf{last})$, i.e., RemAct counts the number of events/activities from the current time point until the end of the process.

Both R_{E8} and R_{E9} can be fed into our tool. For the former, we generate a classification model, and for the latter, we generate a regression model. For all of these tasks, we consider two different trace encodings. First, we use the trace encoding that incorporates several available event attributes, namely

Table 4 The results from the experiments on BPIC 2012 event log using the prediction task R_{E7}

Model	First encoding (more features)					Second encoding (less features)				
	AUC	Accuracy	W. Prec	W. Rec	F-Measure	AUC	Accuracy	W. Prec	W. Rec	F-Measure
Experiments with the analytic rule R_{E7} (predict whether an application will be eventually 'DECLINED')										
R_{E7}										
ZeroR	0.50	0.78	0.61	0.78	0.68	0.50	0.78	0.61	0.78	0.68
Logistic Reg.	0.69	0.78	0.75	0.78	0.76	0.69	0.77	0.71	0.77	0.71
Naive Bayes	0.67	0.33	0.74	0.33	0.30	0.67	0.33	0.73	0.33	0.30
Decision Tree	0.70	0.78	0.76	0.78	0.77	0.70	0.78	0.76	0.78	0.77
Random Forest	**0.71**	0.79	0.77	0.79	**0.78**	**0.71**	0.79	0.77	0.79	**0.78**
AdaBoost	**0.71**	**0.81**	**0.78**	**0.81**	**0.78**	**0.71**	**0.80**	**0.78**	**0.80**	**0.78**
Extra Trees	**0.71**	0.79	0.77	0.79	**0.78**	**0.71**	0.79	0.77	0.79	**0.78**
Voting	**0.71**	0.79	0.77	0.79	**0.78**	**0.71**	0.79	0.77	0.79	0.77
Deep Neural Net.	**0.71**	0.80	0.77	0.80	**0.78**	**0.71**	**0.80**	**0.78**	**0.80**	**0.78**

The bold values indicate the best values that we obtained with respect to the corresponding metric. The best value is not always the highest value. For example, when the metric measures the error (e.g., MAE or RMSE), the smallest value is the best value. However, for the metrics such as accuracy, AUC, precision, recall, and F-measure, the highest value is the best value

monitoringResource, org:resource, activityNameNL, activityNameEN, question, concept:name. Second, we use the trace encoding that only incorporates the event names, i.e., the values of the attribute concept:name. As before, the first encoding considers more information than the second encoding. The evaluation on the generated prediction models from the prediction tasks specified above is shown in Tables 5 and 6.

6.4 Discussion on the experiments

In total, our experiments involve 9 different prediction tasks over 3 different real-life event logs from 3 different domains (1 event log from BPIC 2015, 1 event log from BPIC 2012, and 1 event log from BPIC 2013).

Overall, these experiments show the capabilities of our language in capturing and specifying the desired prediction tasks that are based on the event logs coming from real-life situation. These experiments also exhibit the applicability of our approach in automatically constructing reliable prediction models based on the given specification. This is supported by the following facts: First, for all prediction tasks that we have considered, by considering different input features and machine learning models, we are able to obtain prediction models that beat the baseline. Moreover, for all prediction tasks that predict categorical values, in our experiments we are always able to get a prediction model that has AUC value greater than 0.5. Recall that AUC = 0.5 indicates the worst classifier that is not better than a random guess. Thus, since we have AUC > 0.5, the prediction models that we generate certainly take into account the given input and predict the most probable output based on the given input, instead of randomly guessing the output no matter what the input is. In fact, in many cases, we could even get very high AUC values which are ranging between 0.8 and 0.9 (see Tables 1, 5). This score is very close to the AUC value for the best predictor (recall that AUC = 1 indicates the best classifier).

As can be seen from the experiments, the choice of the input features and the machine learning models influence the quality of the prediction model. The result of our experiments also shows that there is no single machine learning model that always outperforms other models on every task. Since our approach does not rely on a particular machine learning model, it justifies that we can simply plug in different supervised machine learning models in order to get different/better performance. In fact, in our experiments, by considering different models we could get different/better prediction quality. Concerning the input features, for each task in our experiments, we intentionally consider two different input encodings. The first one includes many attributes (hence it incorporates many information), and the second one includes only a certain attribute (i.e., it incorporates less information). In general, our common sense would expect that the more information, the better the prediction quality would be. This is simply because we thought that, by having more information, we have a more holistic view over the situation. Although many of our experiment results show this fact, there are several cases where considering less features could give us a better result, e.g., the RMSE score in the experiment with several models on the tasks R_{E5}, and the scores of several metrics in the experiment R_{E8} show this fact (see Tables 2, 5). In fact, this aligns with the typical observation in machine learning. Irrelevant features could decrease the prediction performance because they might introduce noise in the prediction. Although in the learning process a

Table 5 The results from the experiments on BPIC 2015 event log using the prediction task R_{E8}

Model	First encoding (more features)					Second encoding (less features)				
	AUC	Accuracy	W. Prec	W. Rec	F-Measure	AUC	Accuracy	W. Prec	W. Rec	F-Measure
Experiments with the analytic rule R_{E8} (predicting whether a process is complex)										
R_{E8}										
ZeroR	0.50	0.57	0.32	0.57	0.41	0.50	0.57	0.32	0.57	0.41
Logistic Reg.	0.92	0.83	0.85	0.83	0.83	0.90	0.84	0.84	0.84	0.83
Naive Bayes	0.81	0.72	0.82	0.72	0.71	0.93	0.68	0.81	0.68	0.66
Decision Tree	0.80	0.79	0.80	0.79	0.80	0.84	0.85	0.85	0.85	0.85
Random Forest	**0.95**	**0.89**	**0.89**	**0.89**	**0.89**	**0.95**	**0.90**	**0.90**	**0.90**	**0.90**
AdaBoost	0.92	0.87	0.87	0.87	0.87	0.93	0.88	0.88	0.88	0.88
Extra Trees	**0.95**	0.88	0.88	0.88	0.88	**0.95**	0.88	0.89	0.88	0.88
Voting	0.94	0.85	0.86	0.85	0.86	**0.95**	0.88	0.88	0.88	0.88
Deep Neural Net.	0.89	0.84	0.84	0.84	0.84	0.92	0.84	0.84	0.84	0.84

The bold values indicate the best values that we obtained with respect to the corresponding metric. The best value is not always the highest value. For example, when the metric measures the error (e.g., MAE or RMSE), the smallest value is the best value. However, for the metrics such as accuracy, AUC, precision, recall, and F-measure, the highest value is the best value

Table 6 The results of the experiments on BPIC 2015 event log using the prediction task R_{E9}

Model	First Encoding (more features)		Second Encoding (less features)	
	MAE	RMSE	MAE	RMSE
Experiments with the analytic rule R_{E9} (the number of the remaining events/activities)				
R_{E9}				
ZeroR	11.21	13.274	11.21	13.274
Linear Reg.	6.003	7.748	14.143	18.447
Decision Tree	6.972	9.296	6.752	9.167
Random Forest	4.965	6.884	4.948	6.993
AdaBoost	4.971	6.737	4.879	6.714
Extra Trees	**4.684**	**6.567**	**4.703**	**6.627**
Deep Neural Net.	6.325	8.185	5.929	7.835

The bold values indicate the best values that we obtained with respect to the corresponding metric. The best value is not always the highest value. For example, when the metric measures the error (e.g., MAE or RMSE), the smallest value is the best value. However, for the metrics such as accuracy, AUC, precision, recall, and F-measure, the highest value is the best value

good model should (or will try to) give a very low weight into irrelevant features, the absence of these unrelated features might improve the quality of the prediction. Additionally, in some situation, too many features might cause overfitting, i.e., the model fits the training data very well, but it fails to generalize well while doing prediction with new input data.

Based on the experience from these experiments, time constraint would also be a crucial factor in choosing the model when we would like to apply this approach in practice. Some models require a lot of tuning in order to achieve a good performance (e.g., neural network), while other models do not need many adjustment and able to achieve relatively good performance (e.g., Extra Trees, Random Forest).

Looking at another perspective, our experiments complement various studies in the area of predictive process monitoring in several ways. First, instead of using machine learning models that are typically used in many studies within this area such as Random Forest and Decision Tree (cf. [17,18,35,72]), we also consider other machine learning models that, to the best of our knowledge, are not typically used. For instance, we use Extra Trees, AdaBoost, and Voting Classifier. Thus, we provide a fresh insight on the performance of these machine learning models in predictive process monitoring by using them in various different prediction tasks (e.g., predicting (fine-grained) time-related information, unexpected behaviour). Although this work is not aimed at comparing various machine learning models, as we see from the experiments, in several cases, Extra Trees exhibits similar performance (in terms of accuracy) as Random Forest. There are also some cases where it outperforms the Random Forest (e.g., see the experiment with the task R_{E9} in Table 6). In the experiment with the task R_{E7}, AdaBoost outperforms all other models. Regarding the type of the prediction tasks, we also look into the tasks that are not yet highly explored in the literature within the area of predictive process monitoring. For instance, while there are numerous works on predicting the remaining processing time, to the best of our knowledge, there is no literature exploring a more fine-grained task such as the prediction of the remaining duration of a particular type of event (e.g., predicting the duration of all remaining waiting events). We also consider several workload-related prediction tasks, which is rarely explored in the area of predictive process monitoring.

Concerning the deep learning approach, there have been several studies that explore the usage of deep neural network for predictive process monitoring (cf. [19,23,24,38,63]). However, they focus on predicting the name of the future activities/events, the next timestamp, and the remaining processing time. In this light, our experiments contribute new insights on exhibiting the usage of deep learning approach in dealing with different prediction tasks other than just those tasks. Although the deep neural network does not always give the best result in all tasks in our experiments, there are several interesting cases where it shows a very good performance. Specifically, in the experiments with the tasks R_{E4} and R_{E5} (cf. Table 2), where all other models cannot beat the RMSE score of the baseline, the deep neural network comes to the rescue and be the only model that could beat the RMSE score of our baseline.

To sum up, concerning the three questions regarding this experiment that are described in the beginning of this section, the first two questions are positively answered by our experiment. In our experiments of applying our approach over three different real-life event logs and nine different prediction tasks, we have successfully obtained (reliable) prediction models and positively demonstrate the applicability of our approach. Regarding the third question, our experiments show that the choice of the machine learning model and the information that is incorporated in the trace encoding would influence the quality of the prediction.

7 Related work

This work is tightly 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. The works by [52–55,68,69] focus on predicting the remaining processing time. In [68, 69], the authors present an approach for predicting the remaining processing time based on annotated transition system that contains time information extracted from event logs. The work by [54,55] proposes a technique for predicting the remaining processing time using stochastic Petri nets. The works by [41,49,58,59] focus on predicting delays in process execution. In [58,59], the authors use queueing theory to address the problem of delay prediction, while [41] explores the delay prediction in the domain of transport and logistics process. In [25], the authors present an ad hoc predictive clustering approach for predicting process performance. The authors of [63] present a deep learning approach (using LSTM neural network) for predicting the timestamp of the next event and use it to predict the remaining cycle time by repeatedly predicting the timestamp of the next event.

Looking at another perspective, the works by [18,35,72] focus on predicting the outcomes of a running process. The work by [35] introduces a framework for predicting the business constraints compliance of a running process. In [35], the business constraints are formulated in propositional Linear Temporal Logic (LTL), where the atomic propositions are all possible events during the process executions. The work by [18] improves the performance of [35] by using a clustering preprocessing step. Another work on outcomes prediction is presented by [50], which proposes an approach for predicting aggregate process outcomes by taking into account the information about overall process risk. Related to process risks, [14,15] propose an approach for risks prediction. The work by [37] presents an approach based on evolutionary algorithm for predicting business process indicators of a running process instance, where business process indicator is a quantifiable metric that can be measured by data that is generated by the processes. The authors of [40] present a work on predicting business constraint satisfaction. Particularly, [40] studies the impact of considering the estimation of prediction reliability on the costs of the processes.

Another major stream of works tackle the problem of predicting the future activities/events of a running process (cf. [11,19,23,24,38,53,63]). The works by [19,23,24,38,63] use deep learning approach for predicting the future events, e.g., the next event of the current running process. Specifically, [19,23,24,63] use LSTM neural network, while [38] uses deep feed-forward neural network. In [19,53,63] the authors also tackle the problem of predicting the whole sequence of future events (the suffix of the current running process).

A key difference between many of those works and ours is that, instead of focusing on dealing with a particular prediction task (e.g., predicting the remaining processing time or the next event), this work introduces a specification language that enables us to specify various desired prediction tasks for predicting various future information of a running business process. To deal with these various desired prediction tasks, we present a mechanism to automatically process the given specification of prediction task and to build the corresponding prediction model. From another point of view, several works in this area often describe the prediction tasks under study simply by using a (possibly ambiguous) natural language. In this light, the presence of our language complements this area by providing a means to formally and unambiguously specify/describe the desired prediction tasks. Consequently, it could ease the definition of the task and the comparison among different works that propose a particular prediction technique for a particular prediction task.

Regarding the specification language, unlike the propositional LTL [51], which is the basis of Declare language [47, 48] and often used for specifying business constraints over a sequence of events (cf. [35]), 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, i.e., it is data-aware. Concerning data-aware specification language, the work by [1] introduces a data-aware specification language by combining data querying mechanisms and temporal logic. Such language has been used in several works on verification of data-aware processes systems (cf. [2,12,13,56]). The works by [16,34] provide a data-aware extension of the Declare language based on the First-Order LTL (LTL-FO). Although those languages are data-aware, they do not support arithmetic expressions/operations over the data which is absolutely necessary for our purpose, 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 [45] 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 important for our needs, e.g., in specifying the ping-pong behaviour.

Concerning aggregation, there are several formal languages that incorporate such feature (cf. [5,8,21]) and many of them have been used in system monitoring. The work by [21] extends the temporal logic Past Time LTL with counting quantifier. Such extension allows us to express a constraint on the number of occurrences of events (similar to our **count** function). In [8] a language called SOLOIST is introduced and it supports several aggregate functions on the number of event occurrences within a certain time window. Differently from ours, both [21] and [8] do not consider aggregation over data (attribute values). The works by [5,6] extend the temporal logic that was introduced in [4,7] with several aggregate functions. Such language allows us to select the values to be aggregated. However, due to the interplay between the set and bag semantics in their language, as they have illustrated, some values might be lost while computing the aggregation because they first collect the set of tuples of values that satisfy the specified condition and then they collect the bag of values to be aggregated from that set of tuples of values. To avoid this situation, they need to make sure that each tuple of values has a sort of unique identifier. This situation does not happen in our aggregation because, in some sense, we directly use the bag semantics while collecting the values to be aggregated.

Importantly, unlike those languages above, apart from allowing us to specify a complex constraint/pattern, a fragment of our FOE language also allows us to specify the way to compute/obtain certain values from a trace, which is needed for specifying the desired information/values to be predicted, e.g., the remaining processing time, or the remaining number of a certain activity/event. Our language is also specifically tuned for expressing data-aware properties based on the typical structure of business process execution logs (cf. [32]), and the design is highly driven by the typical prediction tasks in business process management. From another point of view, our work complements the works on predicting SLA/business constraints compliance by providing an expressive language to specify complex data-aware constraints that may involve arithmetic expression and data aggregation.

8 Discussion

This work should be beneficial for the researchers in the area of predictive process monitoring. As we have seen, several works in this area often describe the prediction tasks under study simply by using a (possibly ambiguous) natural language. In this light, the presence of our language complements this area by providing a means to formally and unambiguously specify/describe the desired prediction tasks. Consequently, it could ease the definition of the task and the comparison among different works that propose a particular prediction technique for a particular prediction task. Similarly, as for the business process analyst, the presence of our language should help them in precisely specifying and communicating the desired prediction tasks so as to have suitable prediction services. The presence of our mechanism for creating the corresponding prediction model would also help practitioners in automatically obtaining the corresponding prediction model.

In the following, we provide a discussion concerning the research questions described in Sect. 1 as well as the language requirements in Sect. 3. Furthermore, we also provide a discussion regarding the usability aspect. Finally, this section also discusses potential limitations of this work, which might pave the way towards our future direction.

8.1 Discussion on the research questions

Concerning **RQ1**, i.e., "*How can a specification-driven mechanism for building prediction models for predictive process monitoring look like?*", as introduced in this work, our specification-driven approach for building prediction models for predictive process monitoring essentially consists of several steps: *(i)* First, we have to specify the desired prediction tasks by using the specification language that we have introduced in this work; *(ii)* second, we create the corresponding prediction models based on the given specification by using the approach that is explained in Sect. 4. *(iii)* Finally, once the prediction models are created, we can use the constructed prediction models for predicting the future information of a running process. Notice that this approach requires a mechanism to express various desired prediction tasks and also a mechanism to process the given specification so as to build the corresponding prediction models. In this work, we provide all of these and we have also applied this approach into some case studies based on real-life data (cf. Sect. 6).

Regarding **RQ2**, i.e., "*How can an expressive specification language that allows us to express various desired prediction tasks, and at the same time enables us to automatically create the corresponding prediction model from the given specification, look like? Additionally, can that language allow us to specify complex expressions involving data, arithmetic operations and aggregate functions?*", as explained in Sect. 3, the specification of a prediction task can be expressed by using an analytic rule, which is introduced in this work. Essentially, an analytic rule consists of several conditional-target expressions and it allows us to specify how we want to map each partial business processes execution information into the expected predicted information. As can be seen in Sects. 4 and 6, a specification provided in this language can be used to train a classification/regression model that can be used as the prediction model. Additionally, as a part of analytic rules, we introduce an FOL-based language called FOE. As can be seen from Sects. 3 and 5, FOE allows us to specify complex expression involving data, arithmetic operations and aggregate functions.

Concerning **RQ3**, i.e., "*how can a mechanism to automatically build the corresponding prediction model based on the given specification look like?*", as explained in Sect. 4, roughly speaking, our mechanism to build the corresponding prediction model from the given specification in our language consists of two core steps: *(i)* First, we build the training data based on the given specification in our language; *(ii)* second, we use supervised machine learning technique to learn the corresponding prediction model based on the training data that is generated in the first step.

8.2 Discussion on the language requirements

We now elaborate why our language fulfils all requirements in Sect. 3.2. For the first requirement, since the target expression in the analytic rules can be either numeric or non-numeric expressions, it is easy to see that the first requirement is fulfilled. Various examples of prediction tasks specification in Sect. 5 also confirm this fact. Our experiments in Sect. 6 also exhibit several case studies where the predicted information is either numeric or non-numeric values.

Concerning the second requirement, our procedure in Sect. 4 and our experiments in Sect. 6 confirm the fact that we can have a mechanism for automatically building the corresponding prediction model from the given specification in our language.

Finally, regarding the requirements 3, 4, and 5, our language formalism in Sect. 3 and our extensive showcases in Sect. 5 exhibit the fact that our language fulfils these requirements. Specifically, we are able to specify complex expressions over sequence of events by also involving the events data, arithmetic expressions as well as aggregate functions. Additionally, from our various showcases, we can also see that our language allows us to specify the target information to be predicted where we might also have to specify the way to obtain a certain value and might involve some arithmetic expression.

8.3 Discussion on usability

The usability of the language can only be measured by outcomes of user studies which are beyond the scope of this work. Obviously, it is interesting to do this; therefore, we consider doing a user study as one of our future directions. Although this is one of the limitations of our current work, some insights regarding this aspect can be drawn from previous studies. Taking the success story from the works on domain-specific languages (DSLs) [39], by providing a language in which the development is driven by or tailored to a particular domain/area, DSL offers substantial gains in expressiveness and ease of use compared to general purpose language in their domain of application [39]. In general, DSLs were developed because they can offer domain-specificity in better ways. In this light, our language has a similar situation in the sense that its development is highly driven by a particular area, namely predictive process monitoring. Due to this fact, we expect that similar benefits concerning this aspect could be obtained.

Another aspect of usability concerns learnability. According to [39], one way to design a DSL is to build it based on an existing language. A possible benefit of this approach is familiarity for the users. Due to this familiarity, it is expected that the users, who are familiar with the corresponding existing language, could easily learn and use it. In this light, our FOE language follows this approach. It is actually based on a well-known logic-based language namely First-Order Logic (FOL). Furthermore, choosing FOL as the basis of the language also gives us another advantage since it is more well known compared to other more advanced logic-based languages, e.g., Linear Temporal Logic (LTL). This is the case because typically FOL is a part of many foundational courses; hence, it is taught to a wider audience than other more advanced logics. In general, since this language is a logic-based language, a user with pre-trained knowledge in logic should be able to use the language. Looking at another perspective, the development of our language is also highly driven by the typical structure of XES event log. Since XES is a typical standard format for event logs and widely known in the area of process mining, we expect that this fact improves the familiarity for the users and eases the adoption/usage by the users who are familiar with the XES standard (which should be the typical users in this area). A possible direction to improve the usability would be to study advanced visual techniques and explore the possibility of having visual support (e.g., graphical notations) for creating the specification. Having visual support is a way that has been observed in the area of DSL as a way to improve usability [26,44]. This kind of approach also has been seen in the literature (cf. [47,48]).

According to [44], the effectiveness of communication is measured by the match between the intended message and the received message (information transmitted). In this light, the presence of the formal semantics of our language prevents ambiguous interpretation that could cause a mismatch between the intended and the understood meaning. In the end, the presence of this language could be useful for BP analysts to precisely specify the desired prediction tasks, and this is important in order to have suitable prediction services. This language could also be an effective means for BP analysts to communicate the desired prediction tasks unambiguously.

8.4 Limitation and future work

This work focuses on the problem of predicting the future information of a single running process based on the current information of that corresponding running process. In practice, there could be several processes running concurrently. Hence, it is absolutely interesting to extend the work further so as to consider the prediction problems on concurrently running processes. This extension would involve the extension of the language itself. For instance, the language should be able to specify some patterns over multiple running processes. Additionally, it should be able to express the desired predicted information or the way to compute the desired predicted information, and it might involve the aggregation of information over multiple running processes. Consequently, the mechanism for building the corresponding prediction model needs to be adjusted.

Our experiments (cf. Sect. 6) show a possible instantiation of our generic approach in creating prediction services. In this case we predict the future information of a running process by only considering the information from a single running process. However, in practice, other processes that are concurrently running might affect the execution of other processes. For instance, if there are so many processes running together and there are not enough employees for handling all processes simultaneously, some processes might need to wait. Hence, when we predict the remaining duration of waiting events, the current workload information might be a factor that need to be considered and ideally these information should be incorporated in the prediction. One possibility to overcome this limitation is to use the trace encoding function that incorporates the information related to the processes that are concurrently running. For instance, we can make an encoding function that extracts relevant information from all processes that are concurrently running, and use them as the input features. Such information could be the number of employees that are actively handling some processes, the number of available resources/employees, the number of processes of a certain type that are currently running, etc.

This kind of machine learning-based technique performs the prediction based on the observable information. Thus, if the information to be predicted depends on some unobservable factors, the quality of the prediction might be decreasing. Therefore, in practice, all factors that highly influence the information to be predicted should be incorporated as much as possible. Furthermore, the prediction model is only built based on the historical information about the previously running processes and neglects the possibility of the existence of the domain knowledge (e.g., some organizational rules) that might influence the prediction. In some sense, it (implicitly) assumes that the domain knowledge is already incorporated in those historical data that captures the processes execution in the past. Obviously, it is then interesting to develop further the technique so as to incorporate the existing domain knowledge in the creation of the prediction model with the aim of enhancing the prediction quality. Looking at another perspective, since the prediction model is only built based on the historical data of the past processes execution, this approach is absolutely suitable for the situation in which the (explicit) process model is unavailable or hard to obtain.

As also observed by other works in this area (e.g., [69]), in practice, by predicting the future information of a running process, we might affect the future of the process itself, and hence, we might reduce the preciseness of the prediction. For instance, when it is predicted that a particular process would exhibit an unexpected behaviour, we might be eager to prevent it by closely watching the process in order to prevent that unexpected behaviour. In the end, that unexpected behaviour might not be happened due to our preventive actions, and hence, the prediction is not happened. On the other hand, if we predict that a particular process will run normally, we might put less attention than expected into that process, and hence, the unexpected behaviour might occur. Therefore, knowing the (prediction of the) future might not always be good for this case. This also indicates that a certain care need to be done while using the predicted information.

9 Conclusion

We have introduced an approach for obtaining predictive process monitoring services based on the specification of the desired prediction tasks. Specifically, we proposed a novel rule-based language for specifying the desired prediction tasks, and we devise a mechanism for automatically building the corresponding prediction models based on the given specification. Establishing such language is a non-trivial task. The language should be able to capture various prediction tasks, while at the same time allowing us to have a procedure for building/deriving the corresponding prediction model. Our language is a logic-based language which is fully equipped with a well-defined formal semantics. Therefore, it allows us to do formal reasoning over the specification, and to have a machine processable language that enables us to automate the creation of the prediction model. The language allows us to express complex properties involving data and arithmetic expressions. It also allows us to specify the way how to compute certain values. Notably, our language supports several aggregate functions. A prototype that implements our approach has been developed, and several experiments using real-life event logs confirmed the applicability of our approach. Remarkably, our experiments involve the usage of a deep learning model. In particular, we use the deep feed-forward neural network.

Apart from those that are discussed in Sect. 8, the future work includes the extension of the tool and the language. One possible extension would be to incorporate *trace attribute accessor* that allows us to specify properties involving trace attribute values. As our FOE language is a logic-based language, there is a possibility to exploit existing logic-based tools such as satisfiability modulo theories (SMT) solver [3] for performing some reasoning tasks related to the language. Experimenting with other supervised machine learning techniques would be the next step as well, for instance by using another deep learning approach (i.e., another type of neural network such as recurrent neural network) with the aim of improving the prediction quality.

Acknowledgements Open access funding provided by University of Innsbruck and Medical University of Innsbruck. The authors thank Tri Kurniawan Wijaya for various suggestions related to this work, and Yasmin Khairina for the implementation of several prototype components.

A Extended details on the formal semantics

The aggregate function $\mathbf{max}(\mathsf{numExp}_1, \mathsf{numExp}_2)$ computes the maximum value between the two values that are obtained by evaluating the specified two numeric expressions numExp_1 and numExp_2. It gives undefined value $\perp$ if one of them is evaluated to undefined value $\perp$ (similarly for the aggregate function $\mathbf{min}(\mathsf{numExp}_1, \mathsf{numExp}_2)$ except that it computes the minimum value). Formally, the semantics of these functions is defined as follows:

$$(\mathbf{min}(\mathsf{numExp}_1, \mathsf{numExp}_2))_{\nu}^{\tau,k} = \begin{cases} (\mathsf{numExp}_1)_{\nu}^{\tau,k} & \text{if } (\mathsf{numExp}_1)_{\nu}^{\tau,k} \leq (\mathsf{numExp}_2)_{\nu}^{\tau,k} \\ (\mathsf{numExp}_2)_{\nu}^{\tau,k} & \text{if } (\mathsf{numExp}_1)_{\nu}^{\tau,k} > (\mathsf{numExp}_2)_{\nu}^{\tau,k} \\ \perp & \text{otherwise} \end{cases}$$

$$(\mathbf{max}(\mathsf{numExp}_1, \mathsf{numExp}_2))_{\nu}^{\tau,k} = \begin{cases} (\mathsf{numExp}_1)_{\nu}^{\tau,k} & \text{if } (\mathsf{numExp}_1)_{\nu}^{\tau,k} \geq (\mathsf{numExp}_2)_{\nu}^{\tau,k} \\ (\mathsf{numExp}_2)_{\nu}^{\tau,k} & \text{if } (\mathsf{numExp}_1)_{\nu}^{\tau,k} < (\mathsf{numExp}_2)_{\nu}^{\tau,k} \\ \perp & \text{otherwise} \end{cases}$$

The aggregate function **concat** concatenates the values that are obtained from the evaluation of the given non-numeric expression under the valid aggregation range (i.e., we only consider the value within the given aggregation range in which the aggregation condition is satisfied). Moreover, the concatenation ignores undefined values and treats them as empty string. The formal semantics of the aggregate function **concat** is provided in Fig. 3.

B More showcases

In the following, we present more showcases of prediction task specification using our language.

B.1 Cost-related prediction

Suppose that each activity within a process has its own cost and this information is stored in the attribute named *cost*. The task for predicting the total cost of a process can be specified as follows:

$$R_{19} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathbf{sum}(\mathsf{e}[x].\mathrm{cost};\ \textbf{where}\ x = 1 : \mathsf{last}),\ 0\rangle,$$

where R_{19} maps each trace prefix τ^k into the corresponding total cost that is computed by summing up the cost of all activities. We can also specify the task for predicting the maximal cost within a process as follows:

$$R_{20} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathbf{max}(\mathsf{e}[x].\mathrm{cost};\ \textbf{where}\ x = 1 : \mathsf{last}),\ 0\rangle.$$

$$(\textbf{concat}(\mathsf{nonNumSrc};\ \textbf{where}\ x = \mathsf{st} : \mathsf{ed};\ \textbf{and}\ \mathsf{aggCond}))_{\nu}^{\tau,k} = \begin{cases} \text{""} & \text{if } \mathsf{st} > \mathsf{ed} \\ (\mathsf{nonNumSrc})_{\nu[x \mapsto \mathsf{st}]}^{\tau,k} \odot (\textbf{concat}(\mathsf{nonNumSrc};\ \textbf{where}\ x = \mathsf{st}+1 : \mathsf{ed};\ \textbf{and}\ \mathsf{aggCond}))_{\nu}^{\tau,k} & \text{if } \mathsf{st} \leq \mathsf{ed}, \\ & \quad (\mathsf{nonNumSrc})_{\nu[x \mapsto \mathsf{st}]}^{\tau,k} \neq \bot \\ & \quad \text{and } (\mathsf{aggCond})_{\nu[x \mapsto \mathsf{st}]}^{\tau,k} = \mathsf{true} \\ \text{""} \odot (\textbf{concat}(\mathsf{nonNumSrc};\ \textbf{where}\ x = \mathsf{st}+1 : \mathsf{ed};\ \textbf{and}\ \mathsf{aggCond}))_{\nu}^{\tau,k} & \text{if } \mathsf{st} \leq \mathsf{ed} \text{ and either} \\ & \quad (\mathsf{nonNumSrc})_{\nu[x \mapsto \mathsf{st}]}^{\tau,k} = \bot \\ & \quad \text{or } (\mathsf{aggCond})_{\nu[x \mapsto \mathsf{st}]}^{\tau,k} = \mathsf{false} \\ \bot & \text{otherwise} \end{cases}$$

where $\odot$ is a concatenation operator that simply concatenates two non-numeric values.

Fig. 3 Formal Semantics of the aggregate function **concat**

In this case, R_{20} computes the maximal cost among the cost of all activities within the corresponding process. Similarly, we can specify the task for predicting the average activity cost as follows:

$$R_{21} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \textbf{avg}(\mathsf{e}[x].\mathrm{cost};\ \textbf{where}\ x = 1 : \mathsf{last}),\ 0\rangle.$$

We could also create a more detailed specification. For instance, we want to predict the *total cost of all validation activities*. This task can be specified as follows:

$$R_{22} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{TotalValidationCost},\ 0\rangle.$$

where TotalValidationCost is as follows:

$$\textbf{sum}(\mathsf{e}[x].\mathrm{cost};\ \textbf{where}\ x = 1 : \mathsf{last};\ \textbf{and}\ \mathsf{e}[x].\mathrm{concept{:}name} == \text{“Validation”})$$

In a certain situation, the cost of an activity can be broken down into several components such as human cost and material cost. Thus, the total cost of each activity is actually the sum of the human and material costs. To take these components into account, the prediction task can be specified as follows:

$$R_{23} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{TotalCost},\ 0\rangle.$$

where TotalCost is as follows:

$$\textbf{sum}(\mathsf{e}[x].\mathrm{humanCost} + \mathsf{e}[x].\mathrm{materialCost};\ \textbf{where}\ x = 1 : \mathsf{last})$$

One might consider a process as expensive if its total cost is greater than a certain amount (e.g., 550 Eur), otherwise it is normal. Based on this characteristic, we could specify a task for predicting whether a process would be expensive or not as follows:

$$R_{24} = \langle \mathsf{TotalCost} > 550 \Longrightarrow \text{“expensive”},\ \text{“normal”}\rangle$$

where $\mathsf{TotalCost} = \textbf{sum}(\mathsf{e}[x].\mathrm{cost};\ \textbf{where}\ x = 1 : \mathsf{last})$.

B.2 Predicting process performance

One could consider the process that runs longer than a certain amount of time as *slow*, otherwise it is *normal*. Given a (partial) process execution information, we might be interested to predict whether it will end up as a slow or a normal process. This prediction task can be specified as follows:

$$R_{25} = \langle \mathsf{Cond}_{12} \Longrightarrow \text{“Slow”},\ \text{“normal”}\rangle.$$

where

$$\mathsf{Cond}_{12} = (\mathsf{e}[\mathsf{last}].\mathrm{time{:}timestamp} - \mathsf{e}[1].\mathrm{time{:}timestamp}) > 18.000.000.$$

R_{25} states that if the total running time of a process is greater than 18.000.000 milliseconds (5 hours), then it is categorized as slow, otherwise it is normal. During the training, R_{25} maps each trace prefix τ^k into the corresponding performance category (i.e., slow or normal). In this manner, we get a prediction model that is trained to predict whether a certain (partial) trace will most likely be slow or normal.

Notice that we can specify a more fine-grained characteristic of process performance. For instance, we can add one more characteristic into R_{25} by saying that the processes that spend less than 3 hours (10.800.000 milliseconds) are considered as *fast*. This is specified by R_{26} as follows:

$$R_{26} = \langle \mathsf{Cond}_{12} \Longrightarrow \text{“Slow”},\ \mathsf{Cond}_{13} \Longrightarrow \text{“Fast”},\ \text{“normal”}\rangle$$

where

$$\mathsf{Cond}_{13} = (\mathsf{e}[\mathsf{last}].\mathrm{time{:}timestamp} - \mathsf{e}[1].\mathrm{time{:}timestamp}) < 10.800.000$$

One might consider that a process is performed efficiently if there are only small amount of *task handovers* between resources. On the other hand, one might consider a process is efficient if it involves only a certain number of different resources. Suppose that the processes that have more than 7 times of task handovers among the (human) resources are considered to be inefficient. We can then specify a task to predict whether a (partial) trace is most likely to be inefficient or not as follows:

$$R_{27} = \langle \mathsf{Cond}_{14} > 7 \Longrightarrow \text{“inefficient”}, \text{ “normal”}\rangle$$

where Cond_{14} is specified as follows:

$$\textbf{count}(\mathsf{e}[x].\text{org:resource} \neq \mathsf{e}[x+1].\text{org:resource}; , \\ \textbf{where } x = 1 : \mathsf{last})$$

i.e., Cond_{14} counts how many times the value of the attribute org:resource is changing from a one time point to another time point by checking whether the value of the attribute org:resource at a particular time point is different from the value of the attribute org:resource at the next time point. Now, suppose that the processes that involve more than 5 resources are considered to be inefficient. We can then specify a task to predict whether a (partial) trace is most likely to be inefficient or not as follows:

$$R_{28} = \langle \mathsf{Cond}_{15} > 5 \Longrightarrow \text{“inefficient”}, \text{ “normal”}\rangle$$

where $\mathsf{Cond}_{15} = \textbf{countVal}(\text{org:resource}; \textbf{ within } 1 : \mathsf{last})$, i.e., it counts the number of different values of the attribute org:resource. As before, using R_{27} and R_{28}, we could then train a classifier to predict whether a process will most likely perform inefficiently or normal.

B.3 Predicting future activities/events

The task for predicting the next activity/event can be specified as follows:

$$R_{29} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{e}[\mathsf{curr}+1].\text{concept:name}, \text{ “”}\rangle.$$

During the construction of the prediction model, R_{29} maps each trace prefix τ^k into its next activity name, because $\mathsf{e}[\mathsf{curr}+1].\text{concept:name}$ is evaluated to the name of the next activity.

Similarly, we can specify the task for predicting the next lifecycle as follows:

$$R_{30} = \langle \mathsf{curr} < \mathsf{last} \Longrightarrow \mathsf{e}[\mathsf{curr}+1].\text{lifecycle:transition}, \text{ “”}\rangle$$

In this case, since $\mathsf{e}[\mathsf{curr}+1].\text{lifecycle:transition}$ is evaluated to the lifecycle information of the next event, R_{30} maps each trace prefix τ^k into its next lifecycle.

Instead of just predicting the information about the next activity, we might be interested in predicting more information such as the information about the next three activities. This task can be specified as follows:

$$R_{31} = \langle \mathsf{curr} + 3 \leq \mathsf{last} \Longrightarrow \mathsf{Next3Activities}, \ \mathsf{RemEvents}\rangle.$$

where

$$\begin{aligned}\mathsf{Next3Activities} &= \textbf{concat}(\mathsf{e}[x].\text{concept:name}; \\ &\qquad \textbf{where } x = \mathsf{curr}+1 : \mathsf{curr}+3) \\ \mathsf{RemEvents} &= \textbf{concat}(\mathsf{e}[x].\text{concept:name}; \\ &\qquad \textbf{where } x = \mathsf{curr}+1 : \mathsf{last})\end{aligned}$$

During the construction of the prediction model, in the training phase, R_{31} maps each trace prefix τ^k into the information about the next three activities.

C Implementation

The implementation of our approach is visually illustrated in Fig. 4. Essentially, we have two main phases, namely the *preparation* and the *prediction* phases. In the preparation phase, we construct the prediction models based on the given event log, as well as based on: (i) the prediction tasks specification, (ii) the desired encoding mechanisms, and (iii) the desired classification/regression models. Once the prediction models are built, in the second phase, we can use the generated models to perform the prediction task in order to predict the future information of the given partial trace.

As a proof of concept, we have implemented two ProM plug-ins. One plug-in is for creating the prediction models based on the given specification, and another plug-in is for predicting the future information of a partial trace by using the generated prediction models. More information about the implementation can be found at http://bit.ly/sdprom2.

The use case diagram of our prototype is depicted in Fig. 5.

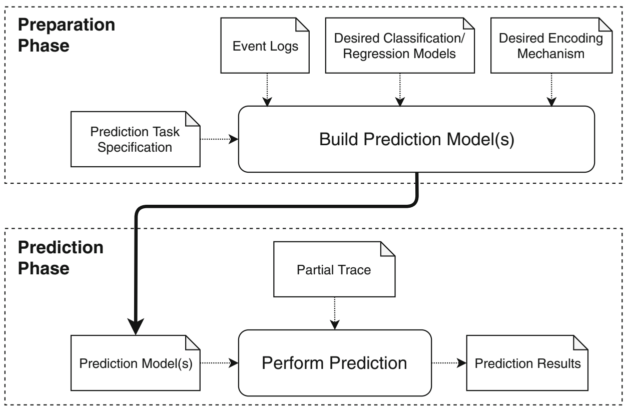
Preparation Phase
Event Logs
Desired Classification/ Regression Models
Desired Encoding Mechanism
Prediction Task Specification
Build Prediction Model(s)
Prediction Phase
Partial Trace
Prediction Model(s)
Perform Prediction
Prediction Results

Fig. 4 Illustration of the approach and the implementation

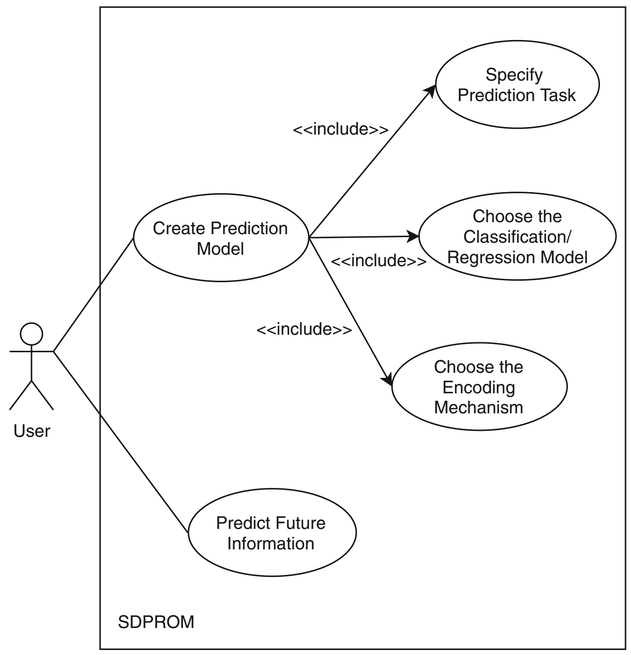
Specify Prediction Task
<<include>>
Create Prediction Model
Choose the Classification/ Regression Model
<<include>>
<<include>>
Choose the Encoding Mechanism
User
Predict Future Information
SDPROM

Fig. 5 Use case diagram of the prototype

References

1. 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)
2. Bagheri Hariri, B., Calvanese, D., Montali, M., Santoso, A., Solomakhin, D.: Verification of semantically-enhanced artifact systems. In: Proceedings of the 11th International Joint Conference on Service Oriented Computing (ICSOC), LNCS, vol. 8274, pp. 600–607. Springer (2013). https://doi.org/10.1007/978-3-642-45005-1_51
3. Barrett, C.W., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Biere, A., Heule, M., van Maaren, H. (eds.) Handbook of Satisfiability. IOS Press, Amsterdam (2009)
4. Basin, D., Klaedtke, F., Müller, S., Pfitzmann, B.: Runtime monitoring of metric first-order temporal properties. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, Leibniz International Proceedings in Informatics (LIPIcs), vol. 2, pp 49–60 (2008)
5. Basin, D., Klaedtke, F., Marinovic, S., Zălinescu, E.: Monitoring of temporal first-order properties with aggregations. In: Runtime Verification (RV) 2013, LNCS, vol. 8174, pp. 40–58. Springer (2013)
6. Basin, D., Klaedtke, F., Marinovic, S., Zălinescu, E.: Monitoring of temporal first-order properties with aggregations. Formal Methods Syst. Des. **46**(3), 262–285 (2015)
7. Basin, D., Klaedtke, F., Müller, S., Zălinescu, E.: Monitoring metric first-order temporal properties. J. ACM **62**(2), 15:1–15:45 (2015)
8. Bianculli, D., Ghezzi, C., San Pietro, P.: The tale of SOLOIST: a specification language for service compositions interactions. In: Formal Aspects of Component Software (FACS) 2012, LNCS, vol. 7684, pp. 55–72. Springer (2013)
9. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
10. Breiman, L., Friedman, J., Stone, C., Olshen, R.: Classification and Regression Trees The Wadsworth and Brooks-Cole Statistics-probability Series. Taylor & Francis, Milton Park (1984)
11. Breuker, D., Matzner, M., Delfmann, P., Becker, J.: Comprehensible predictive models for business processes. MIS Q. **40**(4), 1009–1034 (2016)
12. Calvanese, D., Ísmail Ílkan Ceylan, Montali, M., Santoso, A.: Verification of context-sensitive knowledge and action bases. In: Proceedings of the 14th European Conference on Logics in Artificial Intelligence (JELIA), LNCS, vol. 8761, pp. 514–528, Springer (2014). https://doi.org/10.1007/978-3-319-11558-0_36
13. Calvanese, D., Montali, M., Santoso, A.: Verification of generalized inconsistency-aware knowledge and action bases. In: Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI), pp. 2847–2853. AAAI Press (2015)
14. Conforti, R., de Leoni, M., La Rosa, M., van der Aalst, W.M.P.: Supporting Risk-Informed Decisions During Business Process Execution, pp. 116–132. Springer, Berlin (2013)
15. 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)
16. De Masellis, R., Maggi, F.M., Montali, M.: Monitoring data-aware business constraints with finite state automata. In: Proceedings of the 2014 International Conference on Software and System Process, pp. 134–143. ACM (2014)
17. Di Francescomarino, C., Dumas, M., Federici, M., Ghidini, C., Maggi, F.M., Rizzi, W.: Predictive business process monitoring framework with hyperparameter optimization. In: Proceedings of the 28th International Conference on Advanced Information Systems Engineering (CAiSE), LNCS, vol. 9694, pp. 361–376. Springer (2016)
18. Di Francescomarino, C., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. IEEE Trans. Serv. Comput. **PP**(99), 1–18 (2016)
19. 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: Proceedings of the 15th International Conference on Business Process Management (BPM), LNCS (2017)
20. Di Francescomarino, C., Ghidini, C., Maggi, F.M., Milani, F.: Predictive process monitoring methods: Which one suits me best? In: Proceedings of the 16th International Conference on Business Process Management (BPM), pp. 462–479. Springer (2018)
21. Du, X., Liu, Y., Tiu, A.: Trace-length independent runtime monitoring of quantitative policies in LTL. In: Formal Methods (FM) 2015, LNCS, vol. 9109, pp 231–247. Springer (2015)
22. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, 2nd edn. Springer, Berlin (2018)
23. Evermann, J., Rehse, J.R., Fettke, P.: A deep learning approach for predicting process behaviour at runtime. In: BPM Workshops 2016, pp. 327–338. Springer (2017)
24. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decis. Support Syst. **100**, 129–140 (2017)
25. Folino, F., Guarascio, M., Pontieri, L.: Discovering context-aware models for predicting business process performances. In: On the Move to Meaningful Internet Systems: OTM Conference 2012, pp. 287–304. Springer (2012)
26. Frank, U.: Domain-Specific Modeling Languages: Requirements Analysis and Design Guidelines, pp. 133–157. Springer, Berlin (2013)
27. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. J. Comput. Syst. Sci. **55**(1), 119–139 (1997)
28. Friedman, J., Hastie, T., Tibshirani, R.: The Elements of Statistical Learning. Springer, Berlin (2001)
29. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Mach. Learn. **63**(1), 3–42 (2006)

30. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
31. Han, J., Pei, J., Kamber, M.: Data Mining: Concepts and Techniques. Elsevier, Amsterdam (2011)
32. 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)
33. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: Proceedings of the 13th International Conference on Business Process Management (BPM), LNCS, Springer (2015)
34. Maggi, F.M., Dumas, M., García-Bañuelos, L., Montali, M.: Discovering data-aware declarative process models from event logs. In: Proceedings of the 11th International Conference on Business Process Management (BPM), pp. 81–96. Springer (2013)
35. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: Proceedings of the 26th International Conference on Advanced Information Systems Engineering (CAiSE), LNCS, vol. 8484, pp. 457–472. Springer (2014)
36. Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortés, A.: Predictive monitoring of business processes: a survey. IEEE Trans. Serv, Comput. (2017)
37. Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortés, A., Toro, M.: Run-time prediction of business process indicators using evolutionary decision rules. Expert Syst. Appl. **87**, 1–14 (2017)
38. Mehdiyev, N., Evermann, J., Fettke, P.: A multi-stage deep learning approach for business process event prediction. In: 2017 IEEE 19th Conference on Business Informatics (CBI), vol. 01, pp. 119–128 (2017)
39. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37**(4), 316–344 (2005)
40. Metzger, A., Föcker, F.: Predictive business process monitoring considering reliability estimates. In: Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CAiSE), pp 445–460. Springer (2017)
41. 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)
42. 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)
43. Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of Machine Learning. MIT Press, Cambridge (2012)
44. Moody, D.: The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. IEEE Trans. Softw. Eng. **35**(6), 756–779 (2009)
45. Object Management Group.: Decision Model and Notation (DMN) 1.0. http://www.omg.org/spec/DMN/1.0/ (2015)
46. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
47. Pesic, M., van der Aalst, W.M.P.: A declarative approach for flexible business processes management. In: BPM Workshops 2006, pp. 169–180. Springer (2006)
48. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), pp. 287–287 (2007)
49. 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: BPM Workshops 2012, LNBIP, Springer (2012)
50. 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)
51. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on the Foundations of Computer Science (FOCS), pp. 46–57 (1977)
52. 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)
53. Polato, M., Sperduti, A., Burattin, A., Leoni, Md: Time and activity sequence prediction of business process instances. Computing **100**(9), 1005–1031 (2018)
54. Rogge-Solti, A., Weske, M.: Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays. In: Proceedings of the 11th International Joint Conference on Service Oriented Computing (ICSOC), LNCS, vol. 8274, pp. 389–403. Springer (2013)
55. Rogge-Solti, A., Weske, M.: Prediction of business process durations using non-markovian stochastic petri nets. Inf. Syst. **54**, 1–14 (2015)
56. Santoso, A.: Verification of data-aware business processes in the presence of ontologies. Ph.D. thesis, Free University of Bozen-Bolzano, Technische Universität Dresden. http://nbn-resolving.de/urn:nbn:de:bsz:14-qucosa-213372 (2016)
57. Santoso, A.: Specification-driven multi-perspective predictive business process monitoring. In: Enterprise, Business-Process and Information Systems Modeling, BPMDS 2018, EMMSAD 2018, LNBIP, vol. 318, pp. 97–113. Springer (2018) https://doi.org/10.1007/978-3-319-91704-7_7
58. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining predicting delays in service processes. In: Proceedings of the 26th International Conference on Advanced Information Systems Engineering (CAiSE), LNCS, vol. 8484, pp. 42–57. Springer (2014)
59. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining for delay prediction in multi-class service processes. Inf. Syst. **53**, 278–295 (2015)
60. Senderovich, A., Di Francescomarino, C., Ghidini, C., Jorbina, K., Maggi, F. M.: Intra and inter-case features in predictive process monitoring: a tale of two dimensions. In: Proceedings of the 15th International Conference on Business Process Management (BPM), pp. 306–323. Springer, Cham (2017)
61. Smullyan, R.M.: First Order Logic. Springer, Berlin (1968)
62. Steeman, W.: BPI Challenge (2013). https://doi.org/10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07
63. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Proceedings of the 29th International Conference on Advanced Information Systems Engineering (CAiSE), LNCS, vol. 10253, pp. 477–492. Springer (2017)
64. Theano Development Team.: Theano: A Python Framework for Fast Computation of Mathematical Expressions. arXiv:1605.02688 (2016)
65. van der Aalst, W.: Re-engineering knock-out processes. Decis. Support Syst. **30**(4), 451–468 (2001)
66. van der Aalst, W., et al.: Process mining manifesto. In: BPM Workshops 2012, LNBIP, vol. 99, pp. 169–194. Springer (2012)
67. van der Aalst, W.M.P.: Process Mining-Data Science in Action, 2nd edn. Springer, Berlin (2016)
68. van der Aalst, W.M.P., Pesic, M., Song, M.: Beyond process mining: from the past to present and future. In: Proceedings of the 22nd

international conference on Advanced Information Systems Engineering (CAiSE), LNCS, vol. 6051, pp. 38–52. Springer (2010)
69. van der Aalst, W.M.P., Schonenberg, M., Song, M.: Time prediction based on process mining. Inf. Syst. **36**(2), 450–475 (2011)
70. Van Dongen, B.: BPI Challenge (2012). https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
71. Van Dongen, B.: BPI Challenge (2015). https://doi.org/10.4121/uuid:ed445cdd-27d5-4d77-a1f7-59fe7360cfbe
72. 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: BPM Workshops 2015, LNBIP, vol. 256, pp. 218–229. Springer (2015)
73. Verenich, I., Dumas, M., La Rosa, M., Maggi, F.M., Di Francescomarino, C.: Minimizing overprocessing waste in business processes via predictive activity ordering. In: Proceedings of the 28th International Conference on Advanced Information Systems Engineering (CAiSE), pp. 186–202. Springer (2016)

Ario Santoso received his bachelor degree in computer science from University of Indonesia in 2008. In 2011, he finished his master study within the European Master program in Computational Logic (EMCL), and he received his M.Sc. degree from TU Dresden as well as his Laurea magistrale in Informatica from Free University of Bozen-Bolzano. He obtained his joint Ph.D. degree from both Faculty of Computer Science, Free University of Bozen-Bolzano, and Faculty of Computer Science, TU Dresden, in 2016. He was a postdoctoral researcher at Free University of Bozen-Bolzano before joining the Department of Computer Science, University of Innsbruck, as a postdoctoral researcher in 2017. His research interest lies in the intersection between data and process management, as well as artificial intelligence (specifically, logic-based knowledge representation and reasoning, as well as machine learning). He has served as a reviewer of several international journals such as ACM Transactions on Database Systems (TODS), Journal of Artificial Intelligence Research (JAIR), and Expert Systems with Applications. He has also served as a reviewer of several international conferences such as IJCAI, ECAI, and BPM. For more information, visit his website at http://bit.ly/ariosantoso.

Michael Felderer is a professor at the Department of Computer Science at the University of Innsbruck, Austria, and a guest professor at the Department of Software Engineering at the Blekinge Institute of Technology, Sweden. His fields of expertise and interest include software quality, software and security processes, data-driven engineering, software analytics and measurement, model-based software engineering and empirical research methodology in software and security engineering. Michael Felderer holds a habilitation degree from the University of Innsbruck, co-authored more than 130 publications, and received 10 best paper awards. His research has a strong empirical focus also using methods of data science and is directed towards development and evaluation of efficient and effective methods to improve quality and value of industrial software systems and processes in close collaboration with companies. Michael Felderer is an internationally recognized member of the software engineering research community and supports it as an editorial board member of several journals (IST, IET Software, STTT), organizer of conferences (e.g., General Chair of PROFES 2017, Program Co-Chair of SEAA 2017, Industry Co-Chair of ESEC/FSE 2019), and regular PC member of premier conferences. For more information, visit his website at mfelderer.at.