Data-Aware Remaining Time Prediction of Business Process Instances

Mirko Polato Alessandro Sperduti, *Senior Member, IEEE* Andrea Burattin, *Member, IEEE* Massimiliano de Leoni

***Abstract*— Accurate prediction of the completion time of a business process instance would constitute a valuable tool when managing processes under service level agreement constraints. Such prediction, however, is a very challenging task. A wide variety of factors could influence the trend of a process instance, and hence just using time statistics of historical cases cannot be sufficient to get accurate predictions. Here we propose a new approach where, in order to improve the prediction quality, both the control and the data flow perspectives are jointly used. To achieve this goal, our approach builds a process model which is augmented by time and data information in order to enable remaining time prediction. The remaining time prediction of a running case is calculated combining two factors: *(a)* the likelihood of all the following activities, given the data collected so far; and *(b)* the remaining time estimation given by a regression model built upon the data.**

I. Introduction

The number of companies adopting *Process Aware Information Systems* to support their business processes is constantly growing. All these systems leave traces of each executed activity in the form of *event logs*, and these logs, nowadays, have been analyzed under several aspects. The two most important disciplines, in charge of such analysis, are *data* and *process mining*. Data mining analyzes the so called *data perspective*: set of values that are recorded and analyzed independently from the business process they are coming from. Process mining, on the other hand, considers the entire business process as a whole, and therefore allows *holistic analyses* on data that always are assumed coming from the execution of a business process. Nowadays, several new techniques are populating the family of *multi-perspective* approaches. The basic idea, in these cases, is to define new algorithms that are able to exploit and take advantage from the combination of techniques that belongs to the data and process mining fields.

It is possible to characterize process mining techniques according to their "application time" (i.e. when the analysis takes place). There are, basically, two possible categories: *(i)* a-posteriori; and *(ii)* at runtime. The first case considers as input a finite portion of historical data and tries to extract knowledge out of it. The latter approach, on the other hand, gives information as long as the business process is running. From a business perspective, it is important to get new information and new knowledge as soon as possible, in order to take the proper actions. For example, in a financial institute, handling several business processes instances, it is fundamental to identify frauds as soon as possible, in order to avoid loss of important resources.

All authors are with the Department of Mathematics, University of Padua, Italy. M. de Leoni is also affiliated with the Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands.
Corresponding author is M. Polato: *mpolato@math.unipd.it*.

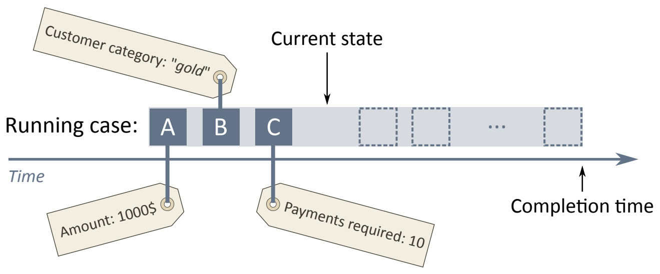

Fig. 1. Graphical representation of the application scenario of the described approach: given a running case we can predict its completion time. We assume that each executed activity (represented as filled boxes) records some additional attributes (represented as labels). Dashed boxes represent future activities.

In this paper we are going to present a new technique that can be used in operational settings. Specifically, we provide an approach that is able, for a running (i.e. not yet completed) case, to predict the completion time of such process instance. The described prediction relies on multiple perspectives: it is based on both the flow of the activities of the running case, and on the data that the current process instance is generating. A graphical representation of the typical application scenario of our approach is shown in Fig. 1. Specifically, given a partial trace containing the logs of already executed activities, it is possible to collect all the data attributes observed until that moment. Considering again the example in Fig. 1, after the execution of A, B, and C, the following attributes are available: *Amount* = 1000$; *Customer category = gold*; and *Payments required* = 10. With all this information (history of the trace and set of data attributes), we are able to predict the completion time of the running instance. The reported experimental results, both on artificial and real data, show that prediction accuracy strongly benefits from the usage of multiple perspectives, and that our approach outperforms previous ones.

The remainder of this paper is structured as follows: Section II reviews recent works concerning prediction tasks in the framework of process mining. Section III gives some essential definitions that are used throughout the paper, while Section V defines the core framework and describes the proposed approach in detail. Experimental results, on artificial and real logs, are described in Section VI, and conclusions are presented in Section VII.

II. Related Work

The problem of accurate prediction of the completion time of a business process instance has already been addressed by various authors and different approaches have been proposed. Here we summarize the most relevant ones.

One of the first tools capable of predicting the cycle time of running process instances is TIBCO Staffware iProcess Suite [1]. The prediction engine uses simulations to complete the running case, without taking into account historical data. This approach is based on a single run through the process model using parameters provides at build time, such as process routing and expected activity durations. In [2] and [3], B.F. van Dongen et al. exploit all the data available in the event log (not only the time information) and use non-parametric regression to predict the cycle time of a running case. Van der Aalst et al. [4] have shown how historical information can be used to build a recommendation system, which is able to predict the next activity of the running instance. In [5] the problem of predicting execution durations of process instances has been analyzed in a proper way, with particular emphasis on issues related to cross-trained resources, however no specific prediction algorithm is proposed. In [6], Song et al. have shown an approach which is not only based on event logs, but it also requires a process model in terms of a transition system (built w.r.t. a given abstraction level). The approach generates a new transition system augmented with time information, learned from historical cases, which can be used for prediction. In [7], the data perspective is taken into consideration to predict SLAs (Service Level Agreement) violations. In this work, besides the explicit data obtained from the event log (called *facts*), Leitner et al. estimate the amount of unknown data in order to improve the prediction quality. The prediction model is based on a regression technique, i.e., a multilayer perceptron trained with the Backpropagation algorithm. Folino et al. ([8], [9]) present an ad-hoc predictive clustering approach, which exploits the method proposed in [6]. Specifically, their approach partitions (i.e. clusters) the log traces according to their associated context features, and for each of these clusters a predictive model (see [6]) is built. The actual prediction considers the cluster that is closest to the partial trace of the running case, and the model associated to that cluster is used to predict the remaining time. Finally, an approach which employs the so called *instance-specific probabilistic process model* (PPM) to predict the likelihood of future tasks is described in [10]. The paper also demonstrates that, on the basis of certain assumptions, the obtained model is Markovian.

The novelty of the approach proposed in this paper, with respect to the ones described above, lies in considering the values of data attributes to predict the remaining time of a running case. The intuition is that the remaining time of a case depends on the event just observed, and on the data attribute values collected until that moment. Different data attribute values, typically, can lead to different paths through the process model, and each path can have a different completion time. Moreover, the same path can have different durations depending on the observed values of the data attributes. With this in mind, our approach uses a representative set of training event logs to build up a process model (i.e. transition system) enriched by regression models trained with the information extracted from the event log.

Once the enriched process model are built, remaining time predictions can be performed by *replaying* the activities of the running cases over the model, and using the regression models belonging to each reached state.

III. Preliminaries

This section reports the basic notations and definitions that are required to understand our approach. Specifically, how to build a transition system annotated with regression models.

Given a set $\mathcal{K}$, a finite sequence over $\mathcal{K}$ of length n is a mapping $s \in ([1,n] \subset \mathbb{N}) \to \mathcal{K}$, and it is represented by a string, i.e., $s = \langle s_1, s_2, \ldots, s_n \rangle$. Over a sequence s we define the following functions:

- *selection operator* $(\cdot)$: $s(i) = s_i$, $\forall\, 1 \le i \le n$;
- $hd^k(s) = \langle s_1, s_2, \ldots, s_{\min(k,n)} \rangle$;
- $tl^k(s) = \langle s_w, s_{w+1}, \ldots, s_n \rangle$ where $w = \max(n-k+1, 1)$;
- $|s| = n$ (i.e. the length of the sequence).

Let us now define the notions of *event*, *trace* and *event log*.

Definition 1 (Event): An *event* is a tuple $e = (a, c, t, d_1, \ldots, d_m)$, where:

- $a \in \mathcal{A}$ is the process activity associated to the event;
- $c \in \mathcal{C}$ is the *case id*;
- $t \in \mathbb{N}$ is the event timestamp (seconds since 1/1/1970[1]);
- $d_1, \ldots, d_m$ is a list of additional attributes, where $\forall\, 1 \le i \le m, d_i \in \mathcal{D}_i$, $\mathcal{D}_i$ being the domain of the i-th attribute.

We call $\mathcal{E} = \mathcal{A} \times \mathcal{C} \times \mathcal{T} \times \mathcal{D}_1 \times \cdots \times \mathcal{D}_m$ the event universe.

Over an event e we define the following *projection* functions: $\pi_a(e) = a$, $\pi_c(e) = c$, $\pi_t(e) = t$ and $\pi_{d_i}(e) = d_i, \forall\, 1 \le i \le m$. If e does not contain the attribute value d_i for some $i \in [1, m] \subset \mathbb{N}$, $\pi_{d_i}(e) = \perp$.

Definition 2 (Trace, Partial Trace): A *trace* is a finite sequence of events $\sigma_c = \langle e_1, e_2, \ldots, e_{|\sigma_c|} \rangle \in \mathcal{E}^*$ such that $\forall\, 1 \le i \le |\sigma|, \pi_c(e_i) = c \wedge \forall\, 1 \le j < |\sigma_c|$, $\pi_t(\sigma_c(j)) \le \pi_t(\sigma_c(j+1))$. We define a *partial trace* of length i as $\sigma_c^i = hd^i(\sigma_c)$, for some $i \in [1, |\sigma_c|] \subset \mathbb{N}$. We call $\mathcal{T}$ the set of all possible (partial) traces.

An *event log* L is a set of traces, $L = \{\sigma_c \mid c \in \mathcal{C}\}$.

Let us now define a transition system (TS) [6], [11], and how one TS can be constructed starting from an event log.

Definition 3 (Generic Transition System): A transition system is a triple (S, E, T), where S is the set of states (i.e. possible state of the process), E is the set of events

[1]We assume this representation according to the Unix epoch time.

(i.e. transition labels), and $T \subseteq S \times E \times S$ is the set of transitions which describe how a system can move from one state to another one. $S^{start} \subseteq S$ is the set of initial states, and $S^{end} \subseteq S$ is the set of final states.

A *walk*, in a Generic Transition System, is a sequence of transitions $\langle t_1, t_2, \ldots, t_n \rangle$ such that $t_1 = (s_1 \in S^{start}, e, s'_1)$, $t_n = (s_n, e, s'_n \in S^{end})$ and $\forall\, 1 < h < n,\ t_h = (s_h, e, s'_h)$ *s.t.* $s'_h \notin S^{end} \wedge s'_h = s_{h+1}$. We say that a trace is *compliant* with the transition system if it corresponds to a walk from $s_i \in S^{start}$ to $s_e \in S^{end}$. Given a state $s \in S$, it is possible to define the set of reachable states from a given state s as: $s\bullet = \{s' \in S \mid \exists t \in T, \exists e \in E \text{ s.t. } t = (s, e, s')\}$. Given a state $s \in S$ we define the set of transitions enabled by s as $T_s = \{(s, e, q) \in T \mid q \in s\bullet \ \wedge e \in E\}$.

According to Song et al. [6], to construct a transition system which maps each partial trace in the log to a state, we need the so called *state* and *event representation functions*.

Definition 4 (State representation function): Let $\mathcal{R}_s$ be the set of possible state representations, a state representation function $f^{state} \in \mathcal{T} \to \mathcal{R}_s$ is a function that, given a (partial) trace σ returns some representation of it (e.g., the set of included activities, the multiset of included activities, the sequence of included activities, ..).

Definition 5 (Event representation function): Let $\mathcal{R}_e$ be the set of possible event representations, an event representation function $f^{event} \in \mathcal{E} \to \mathcal{R}_e$ is a function that, given an event e produces some representation of it (e.g., $\pi_a(e)$).

Definition 6 (Transition System): Given a state representation function f^{state}, an event representation function f^{event} and an event log L, we define a Transition System as $TS = (S, E, T)$, where:

- $S = \{f^{state}(hd^k(\sigma)) \mid \sigma \in L \wedge 0 \le k \le |\sigma|\}$ is the state space;
- $E = \{f^{event}(\sigma(k)) \mid \sigma \in L \wedge 1 \le k \le |\sigma|\}$ is the set of event labels;
- $T(\subseteq S \times E \times S) = \{f^{state}(hd^k(\sigma)),\ f^{event}(\sigma(k+1)),\ f^{state}(hd^{k+1}(\sigma))) \mid \sigma \in L \wedge 0 \le k < |\sigma|\}$ is the transition relation.

$S^{start} = \{f^{state}(\langle\rangle)\}$ is the set of initial states, and $S^{end} = \{f^{state}(\sigma) \mid \sigma \in L\}$ is the set of final states.

Choosing the right functions f^{state} and f^{event}, also referred to as *abstractions*, is not a trivial task [12], [6]. A conservative choice (e.g., no abstraction: $f^{state}(\sigma_c) = \sigma_c$, $f^{event}(e) = e$) can lead to a transition system which does *overfit* the log L, because the state space becomes too large and specific. An aggressive choice (e.g., $f^{state}(\sigma_c) = \{\sigma_c(|\sigma_c|)\}$), instead, can lead to a transition system that *overgeneralizes* the log L, allowing too much behavior. In this latter case the transition system is *underfitting* L. Some possible good choices for f^{state} and f^{event} are described and discussed in [12] and [6]. A common event abstraction is $f^{event}(e) = \pi_a(e)$, which maps an event onto the name of the activity, while commons state abstractions are: the *set abstraction* (i.e., $f^{state}(\sigma_c) = \{\pi_a(e) \mid e \in \sigma_c\}$) and the *list abstraction* (i.e., $f^{state}(\sigma_c) = \langle \pi_a(\sigma_c(1)), \ldots, \pi_a(\sigma_c(|\sigma_c|)) \rangle$).

Algorithm 1 shows how to construct a transition system from an event log.

Algorithm 1: Construction of a Transition System

```
Input: $L$: event log; $f^{state}$: state representation function;
       $f^{event}$: event representation function
Output: $TS$: transition system
1  $S, E, T \leftarrow \varnothing$
2  foreach $\sigma \in L$ do
3      for $k \leftarrow 0$ to $|\sigma|$ do
4          if $s = f^{state}(hd^k(\sigma)) \notin S$ then
5              $S \leftarrow S \cup \{s\}$
6          end
7      end
8  end
9  foreach $\sigma \in L$ do
10     for $k \leftarrow 0$ to $|\sigma|$ do
11         $s \leftarrow f^{state}(hd^k(\sigma))$
12         $e \leftarrow f^{event}(\sigma(k+1))$
13         $s' \leftarrow f^{state}(hd^{k+1}(\sigma)))$
14         if $e \notin E$ then
15             $E \leftarrow E \cup \{e\}$
16         end
17         if $t = (s,\ e,\ s') \notin T$ then
18             $T \leftarrow T \cup \{t\}$
19         end
20     end
21 end
22 $TS \leftarrow (S, E, T)$
23 return $TS$
```

IV. Machine Learning Background

A. Naïve Bayes classifier

Classification is a machine learning task that consists in predicting category membership of data instances. More formally, given a "concept" $\mathcal{F} : X \to Y$ that maps elements of the domain X into a range $Y = \{y_1, y_2, \ldots, y_m\}$ (i.e., the possible categorizations), the classification task consists in learning a function $\tilde{\mathcal{F}}$ which constitutes a good approximation of $\mathcal{F}$.

Naïve Bayes (NB) [13], [14], [15] is a probabilistic classifier which is based on the application of Bayes' theorem. This classifier belongs to the family of the so called *supervised algorithms*. These algorithms need a set of pre-classified instances in order to learn how to classify new, unseen, instances.

Let $\vec{x} = (x_1, x_2, \ldots, x_n) \in X$ be an n-dimensional vector. From a probabilistic point of view, the probability that $\vec{x}$ belongs to a category $y_i \in Y$ is given by the Bayes' theorem:

$$P(y_i \mid \vec{x}) = \frac{P(y_i)P(\vec{x} \mid y_i)}{P(\vec{x})}$$

where $P(y_i)$ is the a-priori probability of y_i and $P(\vec{x})$ is the a-priori probability of $\vec{x}$. The estimation of the conditional probability $P(\vec{x} \mid y_i)$ can be very hard to compute. In order to simplify this problem, it can be assumed that the components x_i of the vector $\vec{x}$ (viewed as random variables) are conditionally independent each other given the target information y_i. With this assumption we can rewrite $P(\vec{x}|y_i)$ as:

$$P(\vec{x} \mid y_i) = \prod_{k=1}^{n} P(x_k \mid y_i).$$

However, the assumption just presented is quite strong and in most cases it does not hold. That's why the described method is usually named *naïve*.

To get the computed classification of the vector $\vec{x}$, we have to find out the *maximum a posteriori (MAP)* class y_{MAP} :

$$\begin{aligned} y_{MAP} &= \arg\max_{y \in Y} \; P(y \mid x_1, x_2, \ldots, x_n) && (1)\\ &= \arg\max_{y \in Y} \; \frac{P(x_1, x_2, \ldots, x_n \mid y)P(y)}{P(x_1, x_2, \ldots, x_n)} && (2)\\ &= \arg\max_{y \in Y} \; P(x_1, x_2, \ldots, x_n \mid y)P(y) && (3)\\ &= \arg\max_{y \in Y} P(y) \prod_{k=1}^{n} P(x_k \mid y) && (4) \end{aligned}$$

the last step (3 to 4) is correct because of the independence assumption. We have to notice that in (4), if one (or more) probability $P(x_k \mid y)$ is zero the whole product goes to zero. In order to avoid this situations, it is possible to apply the Laplacian (or additive) smoothing (see, for example [15]) to the conditional probabilities $P(x_k \mid y)$. It is also possible, given a vector $\vec{x}$, to get the category distribution, computing $P(y) \prod_{k=1}^{n} P(x_k \mid y)/P(x_1, \ldots, x_n)$ for all $y \in Y$.

The training phase of this method consists in the collection of the statistics, from the training set, necessary to calculate (4). The classification is simply the application of (4) over the input vectors.

B. Support Vector Regression

Regression analysis is a statistical process for estimating the relationships among variables. This approach is widely used for prediction and forecasting. One of the most recently proposed approaches is the Support Vector Regression (SVR) [16], [17], [18], which is, as the name suggests, based on Support Vector Machines (SVM).

Let $Tr = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_l, y_l)\} \in \mathcal{X} \times \mathbb{R}$ be the training data, where $\mathcal{X}$ denotes the space of the input vectors. In ϵ-SVR [17], the goal is to find a function $f(\vec{x})$ that deviates from the target y_i by at most ϵ, for all the training instances. In addition to that, the function f has to be as "flat" as possible. Let's start considering the linear case, in which f has the following form:

$$f(\vec{x}) = \langle \vec{w}, \vec{x} \rangle + b \;\text{ with } \vec{w} \in \mathcal{X}, b \in \mathbb{R} \qquad (5)$$

where $\langle \vec{w}, \vec{x} \rangle$ is the dot product between $\vec{w}$ and $\vec{x}$ in $\mathcal{X}$. In (5) *flatness* means that the norm of $\vec{w}$ has to be as small as possible. A possible way to cope with this problem is to minimize the Euclidean norm, i.e., $\|\vec{w}\|^2$. Formally, this can be written as a quadratic constrained optimization problem:

$$\textit{minimize} \quad \frac{1}{2}\|\vec{w}\|^2$$

$$\text{subject to} \quad \begin{cases} y_i - \langle \vec{w}, \vec{x}_i \rangle - b \le \epsilon \\ \langle \vec{w}, \vec{x}_i \rangle + b - y_i \le \epsilon \end{cases}$$

This convex optimization problem is feasible whether a function f actually exists approximating, with an error less or equal than ϵ, all pairs $(\vec{x}_i, y_i)$. However, in real world problems it is not always possible to get such approximation, and hence some errors (grater than ϵ) should be permitted. The introduction of *slack variables* ξ_i, ξ_i^* is useful to allow the violation of some constraints, but all these violations must pay some cost. Now the optimization problem can be rewritten as:

$$\textit{minimize} \quad \frac{1}{2}\|\vec{w}\|^2 + C \sum_{i=1}^{l} (\xi_i + \xi_i^*)$$

$$\text{subject to} \quad \begin{cases} y_i - \langle \vec{w}, \vec{x}_i \rangle - b \le \epsilon + \xi_i \\ \langle \vec{w}, \vec{x}_i \rangle + b - y_i \le \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \end{cases}$$

where the constant $C > 0$ represents a trade-off between the amount of allowed deviations, greater than ϵ, and the flatness of f. The dual formulation of this convex optimization problem [17], [16], [18] provides the key for extending SVM to nonlinear functions. The dual form, passing through the Lagrange multiplayers and applying partial derivatives, is:

$$\textit{maximize} \quad \begin{cases} \sum_{i,j=1}^{l} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle \vec{x}_i, \vec{x}_j \rangle \\ -\epsilon \sum_{i=1}^{l} (\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} y_i(\alpha_i - \alpha_i^*) \end{cases}$$

$$\text{subject to} \quad \begin{cases} \sum_{i=1}^{l} (\alpha_i + \alpha_i^*) = 0 \\ \alpha_i, \alpha_i^* \in [0, C] \end{cases}$$

The nonlinearity of this algorithm could be achieved mapping ($\Phi : \mathcal{X} \to \mathcal{S}$) the input vector $\vec{x}$ to a highly dimensional space $\mathcal{S}$ (i.e. feature space) and then applying the standard SVR method. Unfortunately this direct approach is, in most cases, infeasible from a computational point of view. However, it is worthwhile to note that the dual form depends only on the dot product between the input vectors. Hence, to achieve the nonlinearity, it is sufficient to know the dot product of the input vectors in the feature space $k(\vec{x}, \vec{x}') = \langle \Phi(\vec{x}), \Phi(\vec{x}') \rangle$. Then, it is easy to rewrite the dual form with $k(\vec{x}, \vec{x}')$ instead of $\langle \vec{x}_i, \vec{x}_j \rangle$. Exploiting the Mercer's Theorem [18] it is possible to characterize these type of functions k, called *kernel functions*. After some other simplifications it is possible to get the so called *support vector expansion*:

$$f(x) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) k(\vec{x}_i, \vec{x}) + b.$$

During the training phase the SVR model learns the α_i, α_i^* and b values by solving the optimization problem. The regression phase simply applies f to the input vector.

V. Working Framework

The previous section reported a technique to build a transition system out of an event log. In this section, instead, we show how to enrich such transition system with classification and regression models, and how to use these models to predict the remaining time of running cases of the business process.

The approach presented in this paper exploits the same idea described in [6]. The core difference is the introduction of classification and regression models. Once the standard transition system is built, we enrich each state with a Naïve Bayes classifier and each transitions with a Support Vector Regressor trained with historical data. The intuition is that the Naïve Bayes classifier is used to estimate the probability of transition from state s to state s', while the SVR regressor is used for predicting the remaining time from s'. Formally, let $\hat{p}_{s'}$ be the Naïve Bayes estimated probability to reach state $s' \in s\bullet$ from state s, and $\hat{\tau}(s, e, s')$ the estimated time returned by the SVR associated to transition (s, e, s'). Moreover, given a trace $\sigma \neq \langle\rangle \in \mathcal{T}$ and an event index $i \in ([1, |\sigma| - 1] \subset \mathbb{N})$, we define soj as:

$$soj(\sigma, i) = \pi_t(\sigma(i + 1)) - \pi_t(\sigma(i)).$$

If $\sigma = \langle\rangle \vee i \notin ([1, |\sigma| - 1] \subset \mathbb{N})$ then $soj(\sigma, i) = 0$.

At this point, given the state s reached after observing a (partial) trace σ, the prediction returned by the annotated transition system is:

$$\sum_{(s,e,s') \in T_s} \hat{p}_{s'} \cdot \hat{\tau}(s, e, s') + \hat{\mathbb{E}}_L \left[soj(\sigma, |\sigma|) \right],$$

where $\hat{\mathbb{E}}_L$ refers to the estimated expected value computed over the training log L.

For training we also need to define the remaining time function rem. Given a trace $\sigma \neq \langle\rangle \in \mathcal{T}$ and an event index $i \in ([1, |\sigma|] \subset \mathbb{N})$, we define rem as:

$$rem(\sigma, i) = \pi_t(\sigma(|\sigma|)) - \pi_t(\sigma(i)).$$

If $\sigma = \langle\rangle \vee i \notin ([1, |\sigma|] \subset \mathbb{N})$ then $rem(\sigma, i) = 0$.

In order to exploit the classification and regression models we need some preprocessing to extract data from the event log. In particular, given a partial trace σ_c^i we have to extract the last values (in temporal order) of all the additional data attributes $d_1, \ldots, d_m$, and then put these values into a numerical vector. As mentioned in Section I, in an event log, additional attributes may belong to a continuous data domain, to a discrete numerical, or to a discrete nominal data domain. In the first two cases, data values are simply put into a single component vector without any further processing. In the third case (i.e., discrete nominal data) we use a *one-hot* encoding, in which the nominal data value d_i is represented as a binary vector $v \in \{0, 1\}^{|\mathcal{D}_i|}$, with all values set to 0 except for the component referring to d_i, which is set to 1. Finally, all these vectors are concatenated together. Let us recall the example depicted in Fig. 1. We suppose that events have timestamps respectively 0, 3, 7 and case id equals to 17. We assume also that the *Customer category* attribute can assume this set of values $D = \{\mathit{gold}, \mathit{silver}, \mathit{bronze}\}$. Let's call $\sigma_{17} = \langle e_1, e_2, e_3 \rangle$, where $e_1 = (A, 17, 0, 1000, \perp, \perp)$, $e_2 = (B, 17, 3, \perp, \mathit{gold}, \perp)$ and $e_3 = (C, 17, 7, \perp, \perp, 10)$. The attributes *Amount* and *Payments required* are both numerical, while *Customer category* is nominal. For this last attribute, applying the *one-hot* encoding means to translate the domain D into a vector $v \in \{0, 1\}^{|D|}$, with all components set to 0 except one. In this example, $|D| = 3$, so v belongs to $\{0, 1\}^3$ and $\pi_{\mathit{Customer\ category}}(e_2)$ is translated into $v = [0, 1, 0]$. Considering the last values of all additional attributes, we get the three vectors $[1000]$, $[0, 1, 0]$ and $[10]$. The concatenation of these vectors returns the desired vector $[1000, 0, 1, 0, 10]$.

Formally, we define a function last which, given a (partial) trace $\sigma_c \in \mathcal{T}$ and an additional attribute index $i \in [1, m] \subset \mathbb{N}$, returns the last value assigned to the d_i attribute:

$$\mathit{last}(\sigma_c, i) = \max_{\substack{1 \leq j \leq |\sigma_c| \\ \pi_{d_i}(\sigma_c(j)) \neq \perp}} \pi_{d_i}(\sigma_c(j)).$$

If there is no index j such that $\pi_{d_i}(\sigma_c(j)) \neq \perp$, $\mathit{last}(\sigma_c, i) = \perp$.

Now each attribute must be represented in a vectorial form. In particular, given a (partial) trace σ_c, for each attribute we need to consider the last assigned value. In order to convert all attributes $d_1, \ldots, d_m$ into a single numerical vector, we check their domains and:

- if $\mathcal{D}_j$ is a numerical domain, we project the value $\mathit{last}(\sigma_c, j)$ into a vector with one corresponding component;
- otherwise, we use the one-hot encoding for representing $\mathit{last}(\sigma_c, j)$: we consider a vector with size $|\mathcal{D}_j|$ where each component refers to one element in $\mathcal{D}_j$; the vectorial representation for $v \in \mathcal{D}_j$ is given by the vector with all zeros except for the one referring to v.

If $\mathit{last}(\sigma_c, j) = \perp$ then the null vector, of the right size, is considered. The final vector $\gamma^*(\sigma_c)$ is built considering the concatenation of the representations just mentioned for all attributes d_j.

Let us now define the three kinds of annotation used in this work to enrich the transition system model. In general, an annotation, over a transition system $\mathit{TS} = (S, E, T)$, is a function which associates some data to a state $s \in S$ or to a transition $t \in T$.

Definition 7 (Sojourn Time Annotation): Let TS be a transition system, obtained from an event log L, based on an event representation function f^{event} and a state representation function f^{state}. A *Sojourn Time Annotation* is a function $A : S \to \mathbb{R}$, which returns the average sojourn time for a given state $s \in S$, based on the event log L.

Definition 8 (Naïve Bayes Annotation): Let TS be a transition system, obtained from an event log L, based on an event representation function f^{event} and a state representation function f^{state}. Let's call k the size of the $\gamma^*(\sigma)$ vector calculated for traces $\sigma \in L$. A *Naïve Bayes Annotation* is a function $NB : S \times \mathbb{R}^k \times S \to [0,1] \subset \mathbb{R}$, which, given two states $s_i, s_j \in S$ and a data attribute vector $v \in \mathbb{R}^k$, returns the probability to reach the state s_j starting from s_i through a single transition, applying Naïve Bayes (see Section IV).

Definition 9 (SVR Annotation): Let *TS* be a transition system, obtained from an event log L, based on an event representation function f^{event} and a state representation function f^{state}. An *SVR Annotation* is a function $R : T \times \mathbb{R}^k \to \mathbb{R}$, which, given a transition $t \in T$ and a data attribute vector $v \in \mathbb{R}^k$, applies Support Vector Regression (see Section IV) to return an estimation of the remaining time.

Using these three annotations, we define an annotated transition system as follows:

Definition 10 (Annotated TS): Let $TS = (S, E, T)$ be a transition system, obtained from an event log L, based on an event representation function f^{event} and a state representation function f^{state}. A *Annotated Transition System* is a tuple $PTS = (S, E, T, A, NB, R)$ where, A, NB, R are respectively a sojourn time, a Naïve Bayes and a SVR annotation, based on the event log L and the transition system *TS*.

Training

In this section, we describe how to construct an Annotated Transition System. Algorithm 2 summarizes the construction procedure.

Steps 1 to 3 intializes the training set for each transition to the empty set. The external loop (Steps 4-18) goes through the log, extracting one trace σ_c at a time, while the inner loop (Steps 5-17) replays the trace σ_c over the transition system. For each partial trace the newest attributes data (i.e., *currentData*) are collected, and the corresponding state is identified in the transition system (Steps 6 and 9). Steps 11 and 12, respectively, calculates the remaining time (i.e., *currentRem*), and adds the pair (*currentData*, *currentRem*) as training instance to the SVR associated to the current transition. Then the average sojourn time of the current state is updated, and the training instance (*currentData*, *state*) is added to the Naïve Bayes associated to the current state. Within the last loop (Steps 19-21) all the SVR are trained using the training sets built in the previuos steps. Finally, the annotated transition system is constructed, using the predictors and the collected statistics.

Prediction

In this section, we describe how to predict the remaining time for a running case using an Annotated Transition System. Algorithm 3 summarizes the prediction procedure.

First of all (in Steps 2-3), the algorithm identifies the state reached by the current partial trace (i.e., σ_p) and extracts the additional data attributes. Within the *if* branch of the condition (Steps 6-9), the algorithm treats the case in which the current state originates more than one transition. In this case, the Naïve Bayes classifier calculates the probability to reach any following state (i.e., $NB(state, data, s)$). Such probabilities are used to weigh the remaining time estimation returned by the regression model (Steps 6-8). Finally, the

Algorithm 2: Construction of an Annotated Transition System

```
Input: $L$: event log; $TS = (S, E, T)$: transition system
Output: $T'$: annotated transition system
/* Initialization */
1  foreach $t \in T$ do
2      $svrTrain[t] = \varnothing$  /* Training set for $t$ */
3  end
4  foreach $\sigma_c \in L$ do
5      for $i \leftarrow 1$ to $|\sigma_c| - 1$ do
6          $currentState \leftarrow f^{state}(\sigma_c^i)$
7          $nextState \leftarrow f^{state}(\sigma_c^{i+1})$
8          $nextEvent \leftarrow f^{event}(\sigma_c(i+1))$
9          $currentData \leftarrow \gamma^*(\sigma_c^i)$
10         $trans \leftarrow (currentState, nextEvent, nextState)$
11         $currentRem \leftarrow rem(\sigma_c, i)$
12         $svrTrain[trans] \leftarrow$
13             $svrTrain[trans] \cup (currentData, currentRem)$
           /* Update statistics $A$ */
14         $currentSoj \leftarrow soj(\sigma_c, i)$
15         Update average sojourn time for currentState
           with currentSoj
           /* Update NB, function $NB$ */
16         Update NB for state currentState with the
           training instance $(currentData, nextState)$
17     end
18 end
19 foreach $t \in T$ do
       /* Train SVR, function $R$ */
20     Train SVR for transition $t$ with training set
       $svrTrain[t]$
21 end
22 $T' \leftarrow (S, E, T, A, NB, R)$
23 return $T'$
```

average state sojourn time is added to the time prediction. In the *else* branch, instead, the time prediction comes directly from the regression model associated with the previous transition (Steps 11-14), since there is just a single transition from state s to state s'.

VI. Implementation and Experimental Results

The technique has been implemented for the ProM framework [19]. To mine the transition systems, we rely on the miner's implementation available in ProM. Naïve Bayes Classifications and the Support Vector Regressions are performed using the implementation in the Weka framework [20].

We also aimed to minimize the configuration requirement and, thus, the number of parameters to be set up. The Naïve Bayes Classifier requires no parameters, whereas Support Vector Regression requires to choose the kernel type: either polynomial or RBF [21]. The first kernel requires, as

TABLE I
RESULTS ON TWO REAL-LIFE CASE STUDY. FOR COMPARISON, THE MEAN ABSOLUTE PERCENTAGE ERROR (MAPE) AND THE ROOT MEAN SQUARE PREDICTION ERROR (RMSPE) HAVE BEEN USED. RESULTS ARE ASSESSED USING A 5-FOLD CROSS VALIDATION. BETTER RESULTS ARE HIGHLIGHTED IN BOLD.

	Data-aware Polynomial		**Data-aware RBF**		**van der Aalst et al.**	
Transition System Abstraction	MAPE	RMSPE	MAPE	RMSPE	MAPE	RMSPE
Event Log with 1500 Traces						
Set with Limit 1	**8.93%**	**2.69%**	9.13%	2.73%	23.04%	3.53%
Set with No Limit	7.61%	1.96%	**7.60%**	**1.80%**	22.95%	3.48%
List with No Limit	7.68%	1.97%	**7.61%**	**1.80%**	22.90%	3.41%
Event Log with 5000 Traces						
Set with Limit 1	**5.78%**	**1.87%**	7.42%	2.14%	9.98%	2.33%
Set with No Limit	**5.07%**	**1.13%**	6.68%	1.37%	9.82%	2.37%
List with No Limit	**5.08%**	**1.13%**	6.69%	1.38%	9.13%	2.27%

Algorithm 3: Remaining time prediction for a running case

```
Input: $\sigma_p$: (partial) trace; $TS$: annotated transition system
Output: $P$: remaining time prediction
1  $P \leftarrow 0$
2  $state \leftarrow f^{state}(\sigma_p)$
3  $data \leftarrow \gamma^*(\sigma_p)$
4  $exiting \leftarrow T_{state}$
5  if $|exiting| > 1$ then
6      foreach $(state, trans, nextSt) \in exiting$ do
7          $P \leftarrow P + NB(state, data, nextSt) \cdot R(trans, data)$
8      end
9      $P \leftarrow P + A(state)$
10 else
11     $prevSt \leftarrow f^{state}(\sigma_p^{|\sigma_p|-1})$
12     $prevTrans \leftarrow (s, e, s') \in T_{prevSt}$ s.t.
13                                   $s = prevSt \wedge s' = state$
14     $P \leftarrow R(prevTrans, data)$
15 end
16 return $P$
```

parameter, the polynomial degree; the latter requires the γ value.

The improvement in the prediction with respect to existing approaches is assessed by comparing our technique versus the technique reported in van der Aalst et al. [6]. We also aimed to compare our approach versus the techniques proposed in [8], [9]. Unfortunately, up to this date, neither an implementation of these approaches nor the event logs used in the papers are publicly available. Of course, it is not possible to compare results obtained through different event logs, as the quality of a prediction heavily depends on the information available in the event log.

The comparison with the technique in [6] was made using a real-life case study. It concerns the execution of process instances in an information system for the management of road-traffic fines by a local police of an Italian municipality. The management of road-traffic fines has to comply with Italian laws, which detail the precise work flow. Usually, when a driver commits a violation, a policeman opens a new fine's management and leaves a ticket on the car's glass. The fine's amount depends on the violation performed. Within 180 days, the fine notification must be sent to the offender. The payment can occur in any moment, i.e. before or after that the fine notification is sent by post. If the offender does not pay within 60 days since the reception of the fine notification, the fine doubles. If the offender never pays, eventually the fine is sent to a special agency for credit collection.

In particular, from the Information Systems we extracted two event logs that refer to executions that end with sending for credit collection, i.e. the offenders have not paid the fine in full. These event logs refer to non-overlapping periods in time and contains 1500 log traces and 5000 log traces, respectively. The experiments were performed using the Weka default value for the C SVR hyper-parameter, and two kernel types: polynomial with degree 3 and RBF with γ equal to 10. The transition system was mined using different abstractions, namely set with limit 1 or no limit and list with no limit. Since, in the event logs, for 99% of traces, every activity was performed at most once, the multi-set abstraction was not considered to perform experiments.

Table I reports the results of the experiments for the polynomial and RBF case. For both of cases, the values in the table refers to a 5-fold cross validation. The results are also compared with those obtained by the technique in [6]. To measure and compare the accuracy, we used two indicators: the Mean Absolute Percentage Error (MAPE) and the Root Mean Square Prediction Error (RMSPE). Let n be the number of samples and let A_i and F_i be respectively the actual value and the predicted value for the i-th sample. MAPE usually expresses the accuracy as a percentage:

$$MAPE = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{A_i - F_i}{A_i} \right|$$

RMSPE is also defined as a percentage:

$$RMSPE = 100\% \sqrt{\frac{\sum_{i=1}^{n}(A_i - F_i)^2}{n}}$$

The experimental results show that, especially for the polynomial case, the predictions have been improved quite significantly, thus reducing them to half or, even, to one third for the smaller log. The RBF case returns slightly less accurate predictions. On the other hand, the training is accomplished in one fourth of time with respect to the polynomial case. Therefore, our data-aware approach can provide significantly better predictions with respect to the technique in [6].

Readers can observe that the results are similar for the different transition-system abstractions that were taken into account. This suggests that, for the specific case study, only the last performed activity is relevant when performing predictions.

Last but not least, we aimed to verify whether the data awareness can really help the prediction. For this purpose, we removed every data attribute from the event log and we applied our technique again. In this case, we obtain similar mean percentage errors for both the technique in [6] and ours. This confirms once more that the consideration of data attributes can greatly improve the predictions.

VII. Conclusions and Future Work

In this paper we proposed a new approach for the prediction of the completion time of running process instances. The prediction is based on multiple perspectives: on one hand a transition system encodes information coming from the control-flow perspective; on the other hand all the data recorded by each activity are collected and used to refine the forecast. Approaches already available in the literature use the transition system to store some information associated to a specific trace. Starting from this transition system, we annotate it with three more entities. The first extension encodes the average time spent on every state. The second extension consists of a Naïve Bayes classifier, associated to every state, which, given the set of data attributes, is useful to determine the probability distribution over the set of states reachable from the current one. The last extension is a Support Vector Regressor which, for each transition, given the set of data attributes, predicts the completion time.

Given a partial trace, leading to a particular state of the annotated transition system, its completion time consists of the average time spent on the current state, summed up to the weighted average (with weights coming from the Naïve Bayes) of the predictions (performed by the Support Vector Regressor) of all the reachable states.

Experimental validation is perform on real datasets and, in all cases, our approach outperforms the baseline [6]. Moreover, the improvement of our approach, over the same baseline, seems to reach higher values with respect to the work reported in [8], [9].

Possible future work includes the automatic fine tuning of the parameters of the SVR, in order to simplify the usage of this approach for not-expert users and the improvement of the prediction quality for traces that are not completely fitting the transition system.

Acknowledgment

The work reported in this paper is supported by the Eurostars-Eureka project PROMPT (E!6696).

References

[1] B. Schellekens, "Cycle time prediction in Staffware," Master Thesis, Technische Universiteit Eindhoven, 2009.

[2] B. F. van Dongen, R. Crooy, and W. M. P. van der Aalst, "Cycle Time Prediction: When Will This Case Finally Be Finished?" in *Proceedings of the 16th International Conference of Cooperative Information Systems, OTM 2008*, vol. 5331, no. Chapter 22. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 319–336.

[3] R. Crooy, "Predictions in Information Systems - A process mining perspective," Master Thesis, Technische Universiteit Eindhoven, 2008.

[4] H. Schonenberg, B. Weber, B. F. van Dongen, and W. M. P. van der Aalst, "Supporting flexible processes through recommendations based on history," in *Proceedings of 6th International Conference BPM*. Springer, 2008, pp. 51–66.

[5] H. Reijers, "Case prediction in BPM systems: a research challenge," *Journal of the Korean Institute of Industrial Engineers*, vol. 33, no. 1, pp. 1–10, 2006.

[6] W. van der Aalst, M. Schonenberg, and M. Song, "Time prediction based on process mining," *Information Systems*, vol. 36, no. 2, pp. 450–475, Apr. 2011.

[7] P. Leitner, B. Wetzstein, F. Rosenberg, A. Michlmayr, S. Dustdar, and F. Leymann, "Runtime prediction of service level agreement violations for composite services," in *International Workshops, ICSOC/ServiceWave*. Springer, 2009, pp. 176–186.

[8] F. Folino, M. Guarascio, and L. Pontieri, "Discovering context-aware models for predicting business process performances," in *Proceedings of On the Move to Meaningful Internet Systems Conference: OTM*, vol. 7565. Springer Berlin Heidelberg, 2012, pp. 287–304.

[9] ——, "Discovering High-Level Performance Models for Ticket Resolution Processes," in *Proceedings of On the Move to Meaningful Internet Systems Conference: OTM*, vol. 8185. Springer Berlin Heidelberg, 2013, pp. 275–282.

[10] G. T. Lakshmanan, D. Shamsi, Y. N. Doganata, M. Unuvar, and R. Khalaf, "A markov prediction model for data-driven semi-structured business processes," *Knowledge and Information Systems*, Oct. 2013.

[11] W. M. P. van der Aalst, *Process Mining - Discovery, Conformance and Enhancement of Business Processes*, 1st ed. Springer, 2011.

[12] W. M. P. van der Aalst, V. Rubin, E. Verbeek, B. F. van Dongen, E. Kindler, and C. W. Günther, "Process mining: a two-step approach to balance between underfitting and overfitting," *Software & Systems Modeling*, vol. 9, no. 1, pp. 87–111, Nov. 2008.

[13] J. Han, M. Pei, and K. Jian, *Data Mining Concepts and Techniques*, 3rd ed. Elsevier Science Publishers B. V., 2012.

[14] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*, 1st ed. Cambrige University Press, 2008.

[15] T. M. Mitchell, *Machine Learning*, 1st ed. McGraw-Hill, 1997.

[16] D. Basak, S. Pal, and D. C. Patranabis, "Support Vector Regression," *Neural Information Processing - Letters and Reviews*, vol. 10, no. 10, pp. 203–224, 2007.

[17] H. Drucker, C. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support Vector Regression Machines," *Neural Information Processing Systems*, vol. 1, pp. 155–161, 1996.

[18] A. J. Smola and B. Schölkopf, "A Tutorial on Support Vector Regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, 2004.

[19] H. M. W. Verbeek, J. C. A. M. Buijs, B. F. V. Dongen, and W. M. P. van der Aalst, "ProM 6 : The Process Mining Toolkit," in *BPM 2010 Demos*. Springer, 2010, pp. 34–39.

[20] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, Nov. 2009.

[21] J. Shawe-Taylor and N. Cristianini, *Kernel methods for pattern analysis*. Cambridge University Press, 2004.