Discovering Context-Aware Models for Predicting Business Process Performances

Francesco Folino, Massimo Guarascio, and Luigi Pontieri

Institute for High Performance Computing and Networking (ICAR)
National Research Council of Italy (CNR)
Via Pietro Bucci 41C, I87036 Rende (CS), Italy
{ffolino,guarascio,pontieri}@icar.cnr.it

Abstract. Discovering predictive models for run-time support is an emerging topic in Process Mining research, which can effectively help optimize business process enactments. However, making accurate estimates is not easy especially when considering fine-grain performance measures (e.g., processing times) on a complex and flexible business process, where performance patterns change over time, depending on both case properties and context factors (e.g., seasonality, workload). We try to face such a situation by using an ad-hoc predictive clustering approach, where different context-related execution scenarios are discovered and modeled accurately via distinct state-aware performance predictors. A readable predictive model is obtained eventually, which can make performance forecasts for any new running process case, by using the predictor of the cluster it is estimated to belong to. The approach was implemented in a system prototype, and validated on a real-life context. Test results confirmed the scalability of the approach, and its efficacy in predicting processing times and associated SLA violations.

1 Introduction

Process mining techniques [11] are widely reckoned as a precious tool for the analysis of business processes, owing to their capability to extract useful information out of historical process logs, possibly providing the analyst with a high-level process model. An emerging research stream (see, e.g., [6,13]) concerns the induction of state-aware models for predicting some relevant performance metrics, defined on process instances. For example, in [13], an annotated finite-state model is induced from a given log, where the states correspond to abstract representation of log traces. Conversely, a non-parametric regression model is used in [6] to build the prediction for a new (possibly partial) trace upon its similarity to a set of historical ones, while evaluating traces' similarity based on the comparison of their respective abstract views. The interest towards such novel mining tools stems from the observation that performance forecasts can be exploited to improve process enactments, through, e.g., task/resource recommendations [9] or risk notification [5]. However, accurate forecasts are not easy to make for fine-grain measures (like, e.g., processing times), especially when the analyzed

R. Meersman et al. (Eds.): OTM 2012, Part I, LNCS 7565, pp. 287–304, 2012.

process shows complex and flexible dynamics, and its execution schemes and performances change over time, depending on the context. In fact, the need to recognize and model the influence of context factors on process behavior is a hot issue in BPM community (see, e.g., [14]), which calls for properly extending traditional approaches to process modeling (and, hopefully, to process mining). In general, a way to increase process model precision is to partition the log by ad-hoc clustering methods [10,7,8], and to find a (more precise) model for each cluster, while regarding this latter as evidence for a peculiar execution scenario of the process. To the best of our knowledge, however, all previous clustering-oriented process mining approaches only focused on control-flow aspects, with no connections with the discovery of performance predictors.

In this paper we right attempt to overcome the above limitations by proposing an ad-hoc predictive clustering approach, capable to detect different context-related execution scenarios (or *process variants*), and to equip each of them with a tailored performance-prediction model. Our ultimate goal is to find a novel kind of predictive model, where performance forecasts for any (unfinished) process instance, are made in two steps: the instance is first assigned to a reference scenario (i.e., cluster), whose performance model is then used to eventually make the forecast. Technically, we extend and integrate a method for inducing predictive performance models [13] and a logics-oriented approach to predictive clustering [3], where the discovered model, named Predictive Clustering Tree (PCT), takes the form of a decision-tree. Specifically, the discovery of such scenarios (i.e., clusters) is carried out by partitioning the log traces based on their associated context features, which may include both internal properties of a case (e.g. the amount of goods requested in an order management process) and external factors that characterize the situation where it takes place (e.g., workload, resource availability, and seasonality indicators). Notably, the complex structure of (performance-annotated) process logs makes a trivial application of PCT learning methods likely ineffective and/or computationally expensive. We hence devise a method for encoding each log trace in a propositional form, featuring both its context properties and some associated performance measurements.

Organization. The rest of the paper is structured as follows. Section 2 introduces some notation and basic concepts. The specific problem faced in the paper and the proposed solution approach are described in Section 3. Section 4 discusses an implementation of the approach, and its usage in a real-life setting (as well as the quality metrics used for the evaluation). After discussing experiment results in Section 5, we finally draw a few concluding remarks in Section 6.

2 Formal Framework

Following a standard approach in the literature, we assume that for each process instance (a.k.a "case") a *trace* is recorded, encoding the sequence of *events* happened during the relative enactment. Different data parameters (e.g., the amount of goods asked in a order-handling process) can be kept for any process instance, while each event is associated with a process task and a timestamp –

we here disregard other event properties, such as, e.g., task parameters or executors.We also assume that a additional features can be associated with each trace that characterize the context where it takes place, and capture environmental factors (which may well influence performances).

Let us first denote by $\mathcal{T}$ and E the (fixed) reference universes of all (possibly partial) traces and associated events that may appear in a log. Moreover, let $\hat{\mu} : \mathcal{T} \rightarrow \mathcal{M}$ the unknown function assigning a performance value to each trace — w.r.t. to a given reference performance metrics and an associated space $\mathcal{M}$ of values. Note that $\hat{\mu}$ abstractly indicates the final target of our search, in that we aim at eventually predicting the values of the metrics on any novel enactment. We also assume that two kinds of context properties are defined for a process instance: *(i)* ("intrinsic") *case attributes* $A_1, \ldots, A_q$, with associated domains $D^{A_1}, \ldots, D^{A_q}$, resp., and *(ii)* ("extrinsic") *environmental features* $B_1, \ldots, B_r$, with domains $D^{B_1}, \ldots, D^{B_r}$, resp. – this latter kind of data are meant to capture the state of the BPM system in the moment when the instance starts. Finally, for any sequence s, let $len(s)$ denote its length, and $s[i]$ the element in position i, for $i = 1 \ldots len(s)$. Finally, $s(i]$ is its prefix of s of length i, for $i = 1 \ldots len(s)$, and $s(0] = \langle\rangle$ (the empty sequence). Some further concepts and notation are formally introduced next to conveniently refer to log contents.

Definition 1 (Trace). A *trace* τ ($\in \mathcal{T}$) is a triple $\langle v, \bar{a}, s\rangle$ such that id is a unique identifier, $\bar{a} \in D^{A_1} \times ... \times D^{A_q}$) are its data, and s is a sequence of events. For simplicity, let us also denote $v = id(\tau)$, $\bar{a} = data(\tau)$, $s = seq(\tau)$, $len(\tau) = len(s)$, and $\tau[i] = s[i]$. Moreover, $env(\tau) \in D^{B_1} \times ... \times D^{B_r}$ are the environment features associated with any trace τ, and $context(\tau) \in D^{A_1} \times ... \times D^{A_q} \times D^{B_1} \times ... \times D^{B_r}$ is the juxtaposition of vectors $data(\tau)$ and $env(\tau)$. Finally, $\tau(i] = \langle v^i, \bar{a}^i, s^i\rangle$ is a *prefix* of τ, for $i = 0 \,..\, len(\tau)$, such that v^i is a new identifier, $\bar{a}^i = \bar{a}$, $s^i = s(i]$, $env(\tau(i)] = env(\tau)$, and $context(\tau(i)] = context(\tau)$. □

Definition 2 (Log). A *log* L (over $\mathcal{T}$) is a finite subset of $\mathcal{T}$. Moreover, the *prefix set* of L, denoted by $\mathcal{P}(L)$, is the set of all prefix traces that can be extracted from L, i.e., $\mathcal{P}(L) = \{\tau(i] \mid \tau \in L \text{ and } 0 \leq i \leq len(\tau)\}$. For any log L, we will always assume that $\hat{\mu}(\tau)$ is known for any prefix trace $\tau \in \mathcal{P}(L)$. □

Note that any prefix $\tau(i]$ in Def. 1 is a partial unfolding of τ sharing its context data, while the last statement in Def. 2 can be handled by defining an auxiliary function encoding $\hat{\mu}$ on the prefixes of past log traces – e.g., the (real) remaining time of any prefix of such a trace τ is $\hat{\mu}_{RT}(\tau(i]) = time(\tau[len(\tau)]) - time(\tau[i])$.

2.1 State-Aware Performance Prediction

A *Performance Prediction (Process) Model* (*PPM*, for short), is for us a model that can predict the performance value of any future process enactment, represented as a partial trace. Such a model, indeed, can be regarded as a function $\mu : \mathcal{T} \rightarrow \mathcal{M}$ that tries to estimate $\hat{\mu}$ all over the reference universe of traces. Learning a PPM is then a special induction problem, where the training set is represented as a log L, such that the value $\hat{\mu}(\tau)$ of the target measure is known

for each (sub-)trace $\tau \in \mathcal{P}(L)$. Different solutions were proposed to this problem [13,6], which share the idea of capturing the dependence of performance values on traces (i.e., case histories) by regarding these latter at suitable abstraction levels.

Definition 3 (Trace Abstraction Functions). Let $h \in (N) \bigcup \{\infty\}$ be a threshold on past history. A *trace abstraction function* $abs_h^{mode} : \mathcal{T} \rightarrow \mathcal{R}$ is a function mapping each trace $\tau \in \mathcal{T}$ to an element $abs_h^{mode}(\tau)$ in a space $\mathcal{R}$ of abstract representations. For any $\tau \in \mathcal{T}$, while denoting $n = len(\tau)$ and $j = n-h+1$ if $n > h$ and $j = 1$ otherwise, it is: ***(i)*** $abs_h^{list}(\tau) = \langle task(\tau[j]), \ldots, task(\tau[n])\rangle$; ***(ii)*** $abs_h^{bag}(\tau) = [(t,p) \mid t \in abs_h^{set}(\tau)$ and $p = |\{\tau[k] \mid j \leq k \leq n, task(\tau[k]) = t\}|\ ]$, and ***(iii)*** $abs_h^{set}(\tau) = \{task(\tau[j]), \ldots, task(\tau[n])\}$. □

Each $\alpha \in \mathcal{R}$ is a high level representation for some traces, capturing some hidden state of the process analyzed. In particular, the three concrete abstraction functions defined above maps traces to sequences, sets and multisets, respectively, of task identifiers, and specialize the functions presented in [13] – we here only consider to abstract each trace event into its associated task, while disregarding other event properties (e.g., executors). This restriction could be easily removed from our approach – even though, often, using multiple properties for generalizing may lead to a combinatorial explosion of the abstract representations produced (and to overfitting patterns). In [13], a Finite State Machine (FSM) model is derived, such that a one-to-one mapping exists between its states and the representations produced by some abstraction function *abs*, while each transition is labelled with an event property (namely, a task label in our case). For example, let us assume that abs_∞^{list} is used, and that a, b and c refer to three process tasks. Then, the resulting FSM model will feature a transition labelled with c from state $\langle a, b\rangle$ to state $\langle a, b, c\rangle$, if there is some trace τ in the input log such that $abs_\infty^{list}(\tau[i]) = \langle a, b\rangle$ and $abs_\infty^{list}(\tau(i+1]) = \langle a, b, c\rangle$. In order to make this model capable to make predictions (w.r.t. a measure μ), it is turned into an *Annotated Finite State Machine (A-FSM)*, by equipping each node s with a bag gathering all the values that $\hat{\mu}$ takes at the end of any trace prefix $\tau \in \mathcal{P}(L)$ such that $abs(\tau)$ coincides with the abstraction of s. These measurements help estimate the target measure for any new process instance reaching s, e.g. by simply storing an aggregate statistics (e.g., the average) over them. In principle, our clustering-based scenario discovery scheme could be combined with other state-aware prediction techniques, for it is parametric to the kind of model that is eventually learnt for each scenario. However, in this paper we only consider using A-FSM models, and their associated learning method, to this purpose.

2.2 Predictive Clustering

The core idea of *Predictive Clustering* approaches [2] is that, once discovered an appropriate clustering model, a prediction for a new instance can be based only on the cluster where it is deemed to belong, according to some suitable assignment function. The underlying belief is that the higher similarity between instances of the same cluster will help derive a more accurate predictor – w.r.t. one induced from the whole dataset.

To this end, two kinds of features are considered for any element z in a given space $Z = X \times Y$ of instances: *descriptive* features, denoted by $descr(z) \in X$, and *target* features, denoted by $targ(z) \in Y$ – which are those to be predicted.

Then, a *predictive clustering model (PCM)*, for a given training set $L \subseteq Z$, is a function $m : X \to Y$ of the form $m(x) = p(c(x), x)$, where $c : X \to \mathbb{N}$ is a partitioning function and $p : \mathbb{N} \times X \to Y$ is a prediction function.

An important class of such models are *Predictive Clustering Trees (PCTs)* [2,3], where the cluster assignment function is encoded by a *decision tree*, which can be learnt by recursively partitioning the training set. At each step, a split test is greedily chosen, over one descriptive feature, which (locally) minimizes:

$$loss_d(m, L) \sum_{C_i} |C_i \in c(L)| \;/\; |T| \times \sum_{z \in C_i} d(targ(z), p(z))^2 \qquad (1)$$

where C_i ranges over the current partition of L, and d is a distance measure d over Z. – When working with numeric targets, a good trade-off between scalability and accuracy is typically achieved by simply instantiating d with the classical Euclidean distance over target features only. In this case, $targ(avg(C_i))$ over the target subspace can be also used as the local (constant) predictor of cluster C_i, with $avg(C_i) = |C_i|^{-1} \times \sum_{z \in C_i} z$ – i.e., the cluster's average/centroid.

A variety of PCT learning methods exists in the literature, which differ in the type/number of target features (e.g., decision trees, regression trees, multi-target regression models, clustering trees), or in the underlying representation of data instances – namely, relational (e.g., system TILDE [2]) and propositional (e.g., system CLUS [1]). In our setting, we focus on the discovery of a multi-target regression PCT out of propositional data, mainly owing to scalability reasons.

The core assumption under our work is that process performances really depend on context factors. Hence, to predict the performances of any (partial) trace τ, we regard its associated context data $context(\tau)$ as descriptive attributes.

We can now state the specific kind of performance model we want to discover.

Definition 4 (Context-Aware Performance Prediction Model (`CA-PPM`)). Let L be a log on trace universe $\mathcal{T}$, with context features $context(\mathcal{T})$, and $\hat{\mu} : \mathcal{T} \to \mathcal{M}$, be a performance measure, known for all $\tau \in \mathcal{P}(L)$. Then, a *context-aware performance prediction model* (`CA-PPM`) for L is a pair $M = \langle c, \langle \mu_1, \ldots, \mu_k \rangle \rangle$, encoding a predictive clustering model g_M for $\hat{\mu}$, such that: ***(i)*** $c : context(\mathcal{T}) \to \mathbb{N}$, ***(ii)*** $\mu_i : \mathcal{T} \to \mathcal{M}$, for $i \in c(context(\mathcal{T}))$, and ***(iii)*** $g_M(\tau) = \mu_j(\tau)$ with $j = c(context(\tau))$. □

Notice that the dependence of the target measure on context features relies on the separate modeling of different context-dependent execution scenarios (i.e., clusters), while the performance predictions are eventually based on a cluster assignment function c, which estimates the membership of (possibly novel) process instances to these scenarios. This model is a special kind of PPM model, relying on a predictive clustering one. As such, it can be instantiated by combining a predictive clustering tree (PCT) and multiple (performance-)annotated FSM (A-FSM) models, as building blocks for implementing the functions c and each μ_i, respectively, as discussed next.

3 Problem Statement and Solution Approach

In principle, seeking an explicit encoding for the hidden performance measure $\hat{\mu}$, based on a given log L, can be stated as the search for a `CA-PPM` (cf. Def. 4) minimizing some loss measure, like that in Eq. 1, possibly evaluated on an different sample $L' \subseteq \mathcal{T}$ than the one used as training set. However, to avoid incurring in prohibitive computation times, we rather follow a heuristics approach, where the problem is turned into a combination of two simpler ones, as defined below.

Definition 5 (Problem `CAPP`). *Given* a log L over $\mathcal{T}$, and a performance measure $\hat{\mu}$ only defined on $\mathcal{P}(L)$; *Solve* the following subproblems, sequentially: [**CAPP-S1**]: find a function c (locally) minimizing the loss over a concise representation of the given traces and associated measurements, irrespectively of the cluster-wise prediction function q; and [**CAPP-S2**]: find a function q based on the partition $c(L)$ produced by c (keeping it fixed to as found before). □

Such a simplifying rephrasing of the problem frees us from the burden of simultaneously searching over both any possible partitioning c and all of its associated prediction functions q. Moreover, we want to reuse existing tools for the induction of PCTs and of A-FSM models. This clearly requires to properly define the structure of the training data used to learn a PCT model, since a naïve application of PCT induction algorithms to log contents might lead to unsatisfactory achievements in terms of both scalability and prediction accuracy.

To this end, we propose the adoption of a propositional view of the log, where each (fully unfolded) trace in L acts as an individual training example. We hence dismiss the natural idea of learning the clustering model based on all partial traces in $\mathcal{P}(L)$ (and on their associated performance measurements), for two reasons. First, if working explicitly with all partial traces, the number of training samples will grow substantially, especially in the case where log traces were generated by a process featuring complex and flexible control logics (i.e., many tasks and a high degree of non-determinism). More importantly, since performance values tend to change notably along the course of a process instance – this is right the rationale behind state-aware prediction approaches like [6,13] – the learner may get confused when trying to separate groups of instances with similar target measurements. Think, e.g., to the case of the remaining processing time measure, which progressively decreases as a process enactment goes forward.

On the other hand, using full historical traces as clustering instances, we must decide what are their associated targets, which the PCT learning algorithm has to approximate at best. In fact, each trace τ corresponds to a sequence of target values $(\hat{\mu}(\tau(1], \ldots, \hat{\mu}(\tau))$, and we do not want to use sequences as cluster prototypes, in order to keep the evaluation of candidate split tests fast enough.

As a heuristics solution, each trace is mapped into a vector space, where the dimensions correspond to relevant states of the (hidden) process model. Such target features are computed by way of the trace abstraction functions in Def. 3, which attempt to transform, indeed, each trace into an abstract representation of its enactment state, based on its past history.

Input: A log L over a trace universe $\mathcal{T}$, with data attributes $A = A_1, \ldots, A_q$, and environment features $B = B_1, \ldots, B_r$, a target measure $\hat{\mu}$ known over $\mathcal{P}(L)$,
a trace abstraction function *abs*, and a relevance threshold $\sigma \in [0, 1]$.
Output: A `CA-PPM` model for L (fully encoding $\hat{\mu}$ all over $\mathcal{T}$).
Method: Perform the following steps:

```
1  Associate a vector context(τ) with each τ ∈ L, by computing features env(τ)
2  Compute a set PA_σ(L, abs) of pivot state abstractions (cf. Def. 6)
3  Let PA_σ(L, abs) = {α_1, ..., α_s}
4  Build a performance sketch S for L using context vectors and PA_σ(L, abs)
   // S = {( id(τ), context(τ), ⟨val(τ, α_1), ..., val(τ, α_s)⟩ ) | τ ∈ L} – cf. Eq.2
5  Learn a PCT T with classification (resp., prediction) function c (resp., q) using
   context(τ) (resp., val(τ, α_i), i=1..s) as descript. (resp., target) features, ∀τ ∈ L
6  Let L[1], ..., L[k] denote the discovered clusters – with {1, ..., k} = c(S)
7  for each L[i] do
8     Induce an FSM model f from L[i], using abs as abstraction function
9     Derive an A-FSM f+ model from f
10    Define prediction function μ_i : T → M (for cluster i) based on f+
11 end
12 return ⟨ c, { μ_1, ..., μ_k} ⟩
```

Fig. 1. Algorithm `CA-PPM Discovery`

Specifically, given an abstraction function $abs : \mathcal{T} \to \mathcal{R}$, a "candidate" target feature can be defined for each abstract (state) representation $\alpha \in \mathcal{R}$, such that the value $val(\tau, \alpha)$ of this feature for any trace τ is computed as follows:

$$val(\tau, \alpha) = \begin{cases} \texttt{NULL}, \text{ if } abs(\tau(i]) \neq \alpha \ \forall i \in \{0, ..., len(\tau)\}; \\ agg(\ \langle\ \hat{\mu}(\tau(i_1]), ..., \hat{\mu}(\tau(i_s])\ \rangle\ ), \text{ otherwise.} \end{cases} \quad (2)$$

where $\{i_1, ..., i_s\} = \{j \in \mathbb{Z} \mid 0 \leq j \leq len(\tau) \text{ and } abs(\tau(i]) = \alpha\}$, and $i_j < i_k$ for any $0 \leq j < k \leq s$, while agg is a function aggregating a sequence of measure values into a single one (e.g., the average, median, first, last in the sequence). Note that, for all the tests in Section 5, we always selected the last sequence element.

AS the number of state abstractions may be high, some suitable strategy is needed to select an optimal subset of them, as to prevent the PCT learner from getting lost in a high-dimensional and sparse target space (yet taking long computation times). To this end, we devise an ad-hoc, greedy, selection strategy, to identify a restricted set of "pivot" state abstractions, which looks to be the (locally) best ones in discriminating different performance profiles. The selection criterion used to this purpose relies on a fixed scoring function $\phi : \mathcal{R} \times 2^{\mathcal{T}} \to [0, 1]$ (which will be discussed in details later on), which assigns each state abstraction $\alpha \in \mathcal{R}$ to a score $\phi(\alpha, L)$, quantifying the confidence in α making a profitable target feature w.r.t. the search of a predictive clustering for L. More precisely:

Definition 6 (Pivot State Abstraction). Let L be a log, $abs : \mathcal{T} \to \mathcal{R}$ be a trace abstraction function, and $\sigma \in [0, 1]$ be a relevance threshold. Then, any

$a \in \mathcal{R}$ is a *pivot state abstraction* for L and σ w.r.t. abs, if $\phi(\alpha, L) \geq \sigma$. Moreover, $PA_\sigma(L, abs)$ is the set of all pivot state abstractions for L and σ w.r.t. abs. □

Provided with a set of pivot state abstractions $PA_\sigma(L, abs) = \{\alpha_{j1}, ..., \alpha_{ju}\}$, subproblem `CAPP-S1` can be eventually faced by solving a standard (multi-regression) PCT induction on a dataset where: *(i)* each trace τ in the log corresponds to a distinct instance, *(ii)* the vector $context(\tau)$ encodes the descriptive features of τ and *(iii)* $val(\tau, \alpha_{j1}), ..., val(\tau, \alpha_{ju})$ are the target features of τ. This dataset, called in the following a *performance sketch* of L (w.r.t. abs and σ), offers a propositional view over the log, enabling for a fast and effective calculation of a predictive clustering model.

A detailed description of the different steps of our approach is given in the `CA-PPM Discovery` algorithm, shown in Fig. 1. The meaning of its steps is quite straightforward, as it coincide to the computation process discussed so far. However, it is worth remarking that the induction of an FSM model for each discovered cluster (`step` 8), and its subsequent annotation with performance measurements (`step` 9) are carried out by taking advantage of the techniques presented in [13]. Notably, the performance measurements associated with each state in the model are eventually aggregated into a single constant estimator (namely, the average over them all), in the implementation of $\mu[i]$ (`step` 10). Moreover, whenever a new trace τ generates an unseen sequence of states, as a simple workaround, the function can be extended in a way that its next estimate for τ will be based on the last valid one made for it. Finally, the selection of pivot state features performed in `step` 2 hinges on the following scoring function:

$$\phi(\alpha, L) = \sqrt[3]{\phi_{var}(\alpha, L) \times \phi_{corr}(\alpha, L) \times \phi_{supp}(\alpha, L)} \tag{3}$$

where $\phi_{var}(\alpha, L)$, $\phi_{corr}(\alpha, L)$, and $\phi_{supp}(\alpha, L)$ are all functions ranging on $[0, 1]$.

Basically, function $\phi_{var}(\alpha, L)$ depends on the variability of the values produced by α on all input traces (i.e., $\{val(\alpha, \tau) | \tau \in L\}$) and gives preference to higher-variability features – the more the variability of trace measures the higher the score. Function $\phi_{corr}(\alpha, L)$ measures instead the maximal correlation between the value taken by the feature over each trace and the corresponding value of each descriptive (context) feature – the higher the correlation the higher the score. Finally, $\phi_{supp}(\alpha, L)$ simply is $2 \times \min(0.5, |\{\tau \in L \mid val(\tau, \alpha) > 0\}|)$ – low support state abstractions hardly help find significant groups of traces, indeed. In a sense, the overall scoring function is biased towards features guaranteeing a good compromise between support, correlation with descriptive features (which are the ones guiding the partitioning of log traces) and performance values' variability (in order to find clusters showing quite different performance models).

Before leaving the section, let us observe that the peculiar feature selection subproblem faced here is beyond the scope of the attribute selection capabilities of the heuristics search method embedded in predictive clustering algorithms, due to the fact that our candidate features correspond to target variables, and not to predictor ones. This is also the reason why we cannot trivially reuse feature-selection (i.e., attribute-selection) techniques available in the literature.

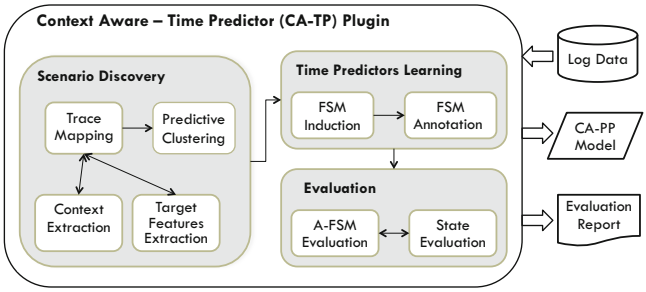

Fig. 2. CA-TP plug-in architecture

4 Case Study: Time Prediction on a Logistics Process

After illustrating the prototype system, in Section 4.1, in the remainder of this section, we discuss the experiments carried out on a real log data and the obtained results. In particular, in Section 4.2, we first illustrate the application scenario, by discussing the kind of data involved in it. Then, in Section 4.3, we introduce the setting adopted to evaluate the quality of discovered models. Finally, in Section 5, the results of tests performed on this scenario are evaluated.

4.1 The Prototype System: Plugin `CA-TP`

As a specialized version of algorithm `CA-PPM Discovery`, we implemented a prototype system, named `CA-TP` (i.e., *Context-Aware Time Prediction*), which can discover a `CA-PPM` for predicting the remaining processing time measure, in order to assess the validity of the approach on practical situations. The prototype system has been developed as plug-in for ProM framework [12], a popular Process Mining framework. The logical architecture of the system is sketched in Figure 2, where arrows between blocks stand for information flows. The whole mining process is driven by the control logic of the the plug-in, while the other modules basically replicate the main computation phases of the algorithm. By *Log Data* we here denote a collection of process logs represented in the MXML [12] format.

The *Scenario Discovery* module is responsible for identifying behaviorally homogeneous groups of traces in terms of both context data and remaining times. In particular, the discovery of different trace clusters is carried out by the *Predictive Clustering* submodule which groups traces sharing both similar descriptive and target values. This latter module leverages the *CLUS* system [1], a predictive clustering framework for inducing PCT models out of propositional data. Such a model is found by trying to optimize the multi-target regression models (w.r.t. a given set of target attributes) of clusters obtained by partitioning the space of descriptive attributes. In this regard, the *Trace Mapping* submodule acts as a "translator" which converts all log traces into propositional tuples, according to the (ARFF) format used in CLUS. As explained above, this mapping relies on the explicit representation of both context data and target attributes, derived

from the original (MXML) log. In particular, the *Context Extraction* module extract extrinsic (environmental) context features, including workload indicators and aggregated time dimensions, and add them to the descriptive attributes of each trace. Notice that this module takes advantage of auxiliary data structures to efficiently search all log data that help capture the local context of any trace τ. In particular, two indexes (based on search trees) over log traces are used, which allow to quickly find all the traces that started or finished, respectively, in a given time range. In fact, these indexes are meant to retrieve all the events occurred during the enactment of τ, to reconstruct its context. Complementarily, the *Target Features Extraction* submodule provides the *Trace Mapping* one with an quasi-optimal set of trace abstractions (obtained by combining trace activities in lists/sets/bags, possibly bounded in their size by a parameter h), which will be eventually used as target features for the predictive clustering step.

Log traces, labeled with cluster IDs, are delivered to the *Time Predictors Learning* module, which, leveraging the approach in [13], derives a collection of *A-FSM* models. More specifically, the submodule *FSM Induction* is used to build a transition model for each cluster, whereas the *FSM Annotation* annotates them with time information. As a final result, a `CA-PPM` model is eventually built, which integrates multiple *A-FSM* models for scenario-specific time predictions, with a set of logical rules (corresponding to the leaves of a PCT model) for discriminating among the discovered scenarios. For inspection purposes and further analysis, the whole model is then stored in an ad-hoc repository.

Module *Evaluator* helps the user assess the quality of time predictions on the test set, by leveraging two submodules: *A-FSM Evaluation* and *State Evaluation*, which compute a series of standard error metrics for an entire *A-FSM* model and for its individual states, respectively. The measures of all predictive models are gathered and eventually combined into global measures (described in Section 4.3), and arranged in a easily-readable report.

4.2 Application Scenario

Our approach has been validated on a real-life scenario, pertaining the handling of containers in a maritime terminal. There, a series of logistic activities are registered for each container passing through the harbor. Massive volumes of data are hence generated continually, which can profitably be exploited to analyze and improve the enactment of logistics processes. In particular, we consider only containers which both arrive and depart by sea, and focused on the different kinds of moves they undergo over the "yard", i.e., the main area used in the harbor for storage purposes. This area is logically partitioned into a finite number of tri-dimensional slots, which are the units of storage space used for containers, and are organized in a fixed number of sectors

The lifecycle of any container can be roughly summarized as follows. The container is unloaded from a ship and temporarily placed near to the dock, until it is carried to some suitable yard slot for being stocked. Symmetrically, at boarding time, the container is first placed in a yard area close to the dock, and then loaded on a cargo. Different kinds of vehicles can be used for moving a

container, including, e.g., cranes, straddle-carriers (a vehicle capable of picking and carrying a container, by possibly lifting it up), and multi-trailers (a train-like vehicle that can transport many containers). This basic life cycle may be extended with additional transfers, classified as "house-keeping", which are meant to make the container approach its final embark point or to leave room for other containers. More precisely, the following basic operations may be registered for any container: *(i)* `MOV`, when it is moved from a yard position to another by a straddle carrier; *(ii)* `DRB`, when it is moved from a yard position to another by a multi-trailer; *(iii)* `DRG`, when a multi-trailer moves to get it; *(iv)* `LOAD`, when it is charged on a multi-trailer; *(v)* `DIS`, when it is discharged off a multi-trailer; *(vi)* `SHF`, when it is moved upward or downward, possibly to switch its position with another container; *(vii)* `OUT`, when a dock crane embarks it on a ship.

In our experimentation, we focused on a subset of 5336 containers, namely the ones that completed their entire life cycle in the hub along the first four months of year 2006, and which were exchanged with four given ports around the Mediterranean sea. To translate these data into a process-oriented form, we viewed the transit of any container through the hub as a single enactment case of a (unknown) logistic process, where each log event refers to one of the basic operations above (i.e., `MOV`, `DRB`, `DRG`, `LOAD`, `DIS`, `SHF`, `OUT`) described above. Each of these operations hence acts as one activity of the reference logistics process.

Context Data. Several data attributes are available for each container (i.e., each process instance), which include, in particular, its origin and final destination ports, its previous and next calls, diverse characteristics of the ship that unloaded it, its physical features (e.g., size, weight), and a series of categorical attributes concerning its contents (e.g., the presence of dangerous or perishable goods). In addition to these internal properties of containers, some additional environmental features are associated with each container, which are meant to capture the context surrounding its arrival to the port. In particular, in our experimentation, we only considered two very basic environmental features: (i) a rough *workload* indicator, simply coinciding with the number of containers still in the port at time t_c, and (ii) a series of low-granularity time dimensions derived from the arrival time (namely, the hour, day of the week and month). Clearly, various additional context variables could be defined, in general, for a process instance (concerning, e.g., resource availability or refined workload indicators), possibly depending on the specific application domain. However, we leave this issue to future work. On the other hand, despite the narrow scope and simplistic nature of these feature, the benefits of using them to detect performance prediction scenarios were neat in our experimentation, as discussed later on.

4.3 Performance Measures and Evaluation Setting

With regard to the scenario above, we want to assess the quality of our approach in predicting the (remaining) time needed to completely process a container (i.e., until the `OUT` activity is performed on it). Knowing in advance such a metrics is of great value for harbor managers, in order to optimize the allocation of resources,

and to possibly prevent, for instance, incurring in violations of SLA (service level agreement) terms. In fact, certain typical SLAs establish that process enactments must not last more than a *Maximum Dwell Time (MDT)*; otherwise pecuniary penalties will be charged to the trans-shipment company. By the way, besides MDT, another important parameter for the scenario on hand is the *average dwell-time (ADT)*, i.e., the average sojourn time for containers in the terminal, which will be also used next for normalizing time measures.

Among the variety of metrics available in the literature, in order to assess the prediction accuracy of our models we resort (like in [13]) to the classic *root mean squared error (rmse)*, *mean absolute error (mae)*, and *mean absolute percentage error (mape)*. In order to reduce the estimation bias, errors are measured according to a (10 fold) cross-validation procedure.

Formally, let us assume that $\tau \in \mathcal{P}(L')$ be a (possibly partial) trace in current test fold L' (amounting to 10% of L's trace), and that $\hat{\mu}_{RT}(\tau)$ (resp., $\mu_{RT}(\tau)$) denote the actual (resp., predicted) remaining time for τ. Then the individual prediction errors associated with all the prefixes (i.e., partial enactments) of τ's are measured as follows: *(i)* $\mathbf{mae} = (1/|\mathcal{P}(L')|) \times \sum_{\tau \in \mathcal{P}(L')} |\hat{\mu}_{RT}(\tau) - \mu_{RT}(\tau)|$; *(ii)* $\mathbf{rmse} = (\ \sum_{\tau \in \mathcal{P}(L')} (\hat{\mu}_{RT}(\tau) - \mu_{RT}(\tau))^2 / |\mathcal{P}(L')|\ )^{1/2}$; and *(iii)* $\mathbf{mape} = (1/|\mathcal{P}(L')|) \times \sum_{\tau \in \mathcal{P}(L')} (|\hat{\mu}_{RT}(\tau) - \mu_{RT}(\tau)|\ /\ \hat{\mu}_{RT}(\tau)$.

In addition to the average prediction errors above (providing actual loss measures), we will also evaluate the capability of a `CA-PPM` to support the prediction of "overtime faults", regarded as a specific form of SLA violations. To this end, let us denote by τ_c a trace encoding the full history of a container c, and $\tau_c[i]$ be its projection till some given step i. Then, an overtime fault for $\tau_c[i]$ is predicted based on the likelihood $\ell_{fault}(\tau_c[i])$ that the total time $\mu_{RT}(\tau_c[i])$, which will be eventually spent to fully handle c, does not exceed MDT. Precisely, letting $eTime(\tau_c[i])$ denote the time already elapsed for c from its arrival at the system, this likelihood is computed as follows:

$$\ell_{fault}(\tau_c) = \begin{cases} 1 - \frac{MDT}{eTime(\tau_c[i]) + \mu_{RT}(\tau_c[i])}, & \text{if } eTime(\tau_c[i]) + \mu_{RT}(\tau_c[i]) > MDT \\ 0, & \text{if } eTime(\tau_c[i]) + \mu_{RT}(\tau_c[i]) \leq MDT \end{cases}$$

For a suitably chosen risk tolerance threshold γ_{risk}, an alert is eventually triggered, while looking at the partial enactment $\tau_c[i]$, whenever $\ell_{fault}(\tau_c[i]) > \gamma_{risk}$, to notify the high risk of an incoming overtime fault – the greater the threshold, the lower sensitivity to the detection of potential overtime faults. Then, interpreting fault prediction as a classification problem with two given classes, i.e., `true` vs. `false` overtime faults, we can measure the prediction accuracy by computing the rates FN of False Negatives (i.e., overtime faults that were not deemed as such) and FP of False Positive (i.e., normal cases signaled as risky), as well as classical measures of *Precision* (i.e., $P = TP/(TP + FP)$, *Recall* (i.e., $R = TP/(TP + FN)$), with TP denoting the number of true positives, i.e., correctly predicted overtime faults. Incidentally, $\tau_c[i]$ is a `true` positive if $time(\tau(len(\tau))) > MDT$, and `true` negative otherwise.

5 Experiment Results

A series of tests were performed to assess the effectiveness and the efficiency of our approach in discovering a `CA-PPM` for remaining time prediction, based on the log described in the previous section. To this end, we tested our approach with various configurations of its parameters. In the following, we will report results obtained for different configurations of the two parameters associated with the abstraction function abs_h^{mode}: the horizon limit h, and the abstraction $mode \in \{list, bag\}$ – results with set-based abstractions are not shown here, due to their minor relevance, as discussed afterwards. Conversely, a fixed configuration is shown for threshold σ (namely, $\sigma = 0.4$), which was chosen pragmatically based on a series of specific tests, omitted here for space reasons.

All error results shown next were averaged over 10 trials, whereas their respective variance are not reported for the sake of brevity. Notice, however, that standard deviations were always lower than 5% of the average for all the metrics.

Qualitative Results. Before illustrating quantitative results in detail, let us show an example of one `CA-PPM` (when $abs_h = abs_4^{bag}$ and $\sigma = 0.4$) induced from the above log, in order to enable for a rough evaluation of the descriptive features of the model – even though its main goal is to offer operational support by means of performance predictions. In particular, the Figure 3 (a) reports, as a portion of the clustering function, the decision rule corresponding to one of the clusters found (namely, cluster 37), which actually corresponds to one of the leaves of the PCT model discovered with CLUS. This rule allows for easily interpreting the semantics of the cluster in terms of both container properties (namely, the origin port `PrevHarbor`, the destination port `NextHarbor`, the navigation line that is going to take it away `NavLine_OUT`, the navigation line bringing it in the current port `NavLine_IN`), and environmental context data (namely, the basic workload indicator `Workload`, based on instance counts, and aggregated time dimensions `ArrivalDay` or `ArrivalHour`). Despite its simplicity, the rule helps characterize a very peculiar, and yet relatively frequent scenario (the cluster gathers, indeed, 43 out of the 5336 traces) for the handling of containers.

As a matter of fact, the A-FSM model found for the same cluster (shown in Figure 3 (b)) witnesses that for this peculiar configuration of context factors (i.e.,

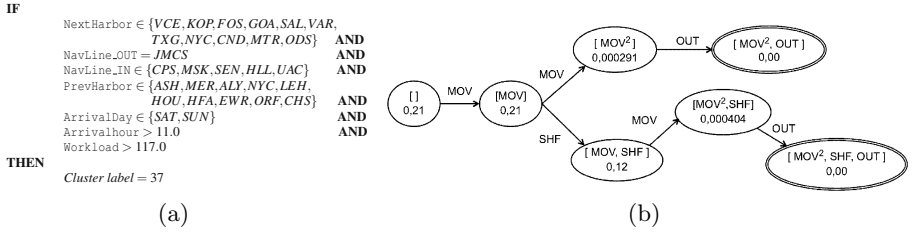

Fig. 3. Excerpt from a `CA-PPM` model for the harbor log, showing the (a) decision rule and (b) A-FMS model of one of the clusters found – (a) and (b) are a sort of (data-driven) descriptions for a context variant and its associated process variant, resp.

context variant), the containers tend to undergo a very small, and quite specific, paths over logistics operations. By the way, each node in the $A\text{-}FSM$ is labelled with the bag of (the 4 more recent) operations leading to it – e.g., the node tagged with [MOV^2, OUT] encodes all the traces in the cluster that undergo two MOVs before leaving the yard (operation OUT). Along with labels, each node also reports a constant prediction for the remaining time (normalized w.r.t. ADT). Edge labels codify, instead, which operations can trigger the corresponding node transition. For the sake of clarity, if a container is in the state labelled as [MOVE, SHF] and a further MOV operation occurs, then the next state will be the one associated with [MOV^2, SHF]. Notably, this simple A-FMS model gave a neatly positive contribution to the accuracy of the global CA-PPM model – very low errors (namely, $rmse = 0.138$, $mae = 0.080$, and $mape = 0.302$) were produced, indeed, on the test traces that were assigned to it.

Time Prediction Effectiveness. Table 1 summarizes the errors made in predicting remaining times (normalized by the average dwell time ADT) for the case of *rmse* and *mae*), using both our CA-TP plug-in and the prediction method proposed in [13] (here denoted by FSM, and also employed as a base learner in our approach). The tests were performed using different trace abstraction functions abs_h, and keeping fixed threshold $\sigma = 0.4$. For the sake of comparison, Table 2 also reports the percentage of error reduction ($\Delta\%$) obtained by CA-TP w.r.t. FSM. Moreover, the results of CA-TP are further differentiated according to which kinds of descriptive features were used. Specifically, CA-TP$^-$ refers to the case where a CA-PPM is built only considering static container properties (e.g., dimensions, origin/destination ports). Conversely, CA-TP$^+$ indicates the case where log traces are also associated with extrinsic context features (namely, workload indicators and seasonality dimensions), in addition to their primitive data attributes. These figures clearly show that our clustering-based method performs always better than the baseline, no matter of the parameter setting.

By a closer look, two factors appear to affect more the performances: the usage of derived context features and the value of history horizon h. In particular, the advantage of using environment-driven features is neat, despite they were very rough and partial, seeing as the average error reduction (computed over all error metrics) of CA-TP$^+$ is close to 37%, whereas CA-TP$^-$ "just" gets a 24% improvement. As to h, it is easily seen that, although the benefits of using our solution gets appreciable as soon as $h > 1$, the best performances are reached for $h = 4$, when all kinds of errors shrink more than 65% w.r.t. the baseline (see Table 2). Stretching the horizon beyond 8 seem to bring no further advantages (apart minor improvements for the *mape* error with abs_8^{BAG}). This result is not surprising, seeing as accuracy achievements might even fall when using high values of h, due to the excessive level of detail on trace histories (and to the consequent high risk of overfitting).

The effect of the abstraction mode looks less marked, as very similar (good) results are found in both cases. Actually, whatever h and the kind of context features, less than 1% error reduction is obtained (on all metrics) when adopting bag abstractions, w.r.t. the case where lists were used. Finally, we notice that

Table 1. Average prediction errors (computed via a 10-fold cross-validation), for `CA-TP` and the baseline method (FSM), and different abstraction functions abs_h^{type} ($\sigma = 0.4$)

Parameters (abs_h^{mode})		**FSM [13]**			**CA-TP$^-$**			**CA-TP$^+$**		
mode	h	*rmse*	*mae*	*mape*	*rmse*	*mae*	*mape*	*rmse*	*mae*	*mape*
LIST	1	0.655	0.444	2.985	0.649	0.436	2.964	0.647	0.436	2.811
	2	0.465	0.211	0.516	0.335	0.102	0.376	0.335	0.095	0.355
	4	0.465	0.204	0.418	0.342	0.102	0.246	0.160	0.058	0.114
	8	0.465	0.204	0.407	0.349	0.102	0.175	0.164	0.058	0.107
	16	0.465	0.204	0.407	0.349	0.102	0.175	0.164	0.058	0.107
	Total	**0.503**	**0.253**	**0.947**	**0.409**	**0.169**	**0.787**	**0.298**	**0.141**	**0.699**
BAG	1	0.655	0.444	2.985	0.649	0.436	2.964	0.647	0.436	2.811
	2	0.473	0.218	0.560	0.342	0.109	0.404	0.342	0.102	0.375
	4	0.465	0.211	0.420	0.335	0.095	0.248	0.156	0.058	0.118
	8	0.465	0.211	0.420	0.342	0.095	0.170	0.156	0.058	0.107
	16	0.465	0.211	0.420	0.342	0.095	0.170	0.156	0.058	0.107
	Total	**0.505**	**0.259**	**0.961**	**0.406**	**0.166**	**0.791**	**0.296**	**0.143**	**0.704**
Grand Total		**0.504**	**0.256**	**0.954**	**0.407**	**0.167**	**0.789**	**0.297**	**0.142**	**0.701**

Table 2. Error reductions (%) – derived from Table 1 – achieved by `CA-TP` w.r.t. FSM

Parameters (abs_h^{mode})		**CA-TP$^-$** (Δ%)			**CA-TP$^+$** (Δ%)		
mode	h	*rmse*	*mae*	*mape*	*rmse*	*mae*	*mape*
LIST	1	-0.8%	-1.6%	-0.7%	-1.2%	-1.6%	-5.8%
	2	-28.1%	-51.7%	-27.2%	-28.1%	-55.2%	-31.3%
	4	-26.6%	-50.0%	-41.1%	-65.6%	-71.4%	-72.8%
	8	-25.0%	-50.0%	-57.0%	-64.8%	-71.4%	-73.8%
	16	-25.0%	-50.0%	-57.0%	-64.8%	-71.4%	-73.8%
	Total	**−18.8%**	**−33.3%**	**−16.8%**	**−40.8%**	**−44.3%**	**−26.2%**
BAG	1	-0.8%	-1.6%	-0.7%	-1.2%	-1.6%	-5.8%
	2	-27.7%	-50.0%	-27.8%	-27.7%	-53.8%	-33.0%
	4	-28.1%	-55.2%	-41.0%	-66.4%	-72.4%	-72.0%
	8	-26.6%	-55.2%	-59.6%	-66.4%	-72.4%	-74.5%
	16	-26.6%	-55.2%	-59.6%	-66.4%	-72.4%	-74.5%
	Total	**−19.6%**	**−36.0%**	**−17.7%**	**−41.4%**	**−44.9%**	**−26.8%**
Grand Total		**−19.2%**	**−34.7%**	**−17.3%**	**−41.1%**	**−44.6%**	**−26.5%**

poorer performances were obtained, in general, when using all methods with set-oriented trace abstraction functions (i.e., abs_h^{set}). However, since our approaches confirmed, even in such a case, similar degrees of improvement over the baseline, as those in Table 2, these results are not reported here for lack of space.

Fault Prediction Effectiveness. In general, the quality of overtime fault estimation is measured w.r.t. a given maximum dwell-time MDT, set in predefined agreements on service quality between the shipping lines and the terminal handler. In our tests we fixed $MDT = 2 \times ADT$ (namely, MDT=11.46 days).

Figure 4 sheds light on the ability our approach discriminate "over-time" from "in-time" containers. To this purpose, we report both Precision and Recall scores for different values of the risk threshold γ_{risk}, when a fixed, good-working, configuration of the underlying trace abstraction criterion (namely, abs_h=abs_4^{BAG}) is used for both our approach and the baseline one [13] (FSM), and σ=0.4 in our feature selection procedure. Notice that we only consider here the case where our tool (referred to as `CA-TP`$^+$ in the figure) is provided with all kinds of (both intrinsic and extrinsic) context features available in the application scenario. As expected, recall tends to worsen when increasing γ_{risk}, while an opposite trend

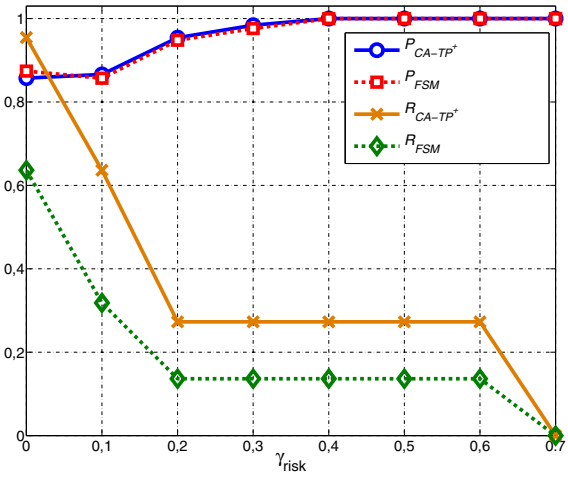

Fig. 4. Accuracy scores for the prediction of overtime faults by CA-TP$^+$ and by the baseline methods, when varying γ_{risk}, while fixing σ =0.4, h=4, and abs_h^{bag}

is perceived for precision results. Interestingly, when using lower values of γ_{risk} (i.e., a more aggressive warning policy) the capability of our approach to recognize real overtime cases is compelling w.r.t. the baseline predictor – in particular, an astonishing recall of 0.95 (vs. 0.64) is reached with γ_{risk}=0. In general, recall scores are usually more important than precision ones in our scenario, since containers "stuck" in the yard implies high monetary costs, and if effectively recognizing them, suitable counter-measures could be undertaken – possibly resorting to the usage of additional (storage/processing) resources, which are not used in normal conditions for economical reasons. Clearly, such remedial policies as well come with a cost, even though it is typically far lower than SLA-violation penalties. Anyway, seeing as our method gets quite good precision scores over a wide range of γ_{risk}'s values, it is reasonable to expect that a suitable trade-off can be reached, according to actual application requirements. More specifically, notice that the precision scores of the two methods are very similar for any value of γ_{risk} (in particular, our method never work significantly worse than the baseline one), and both flatten on 1 with $\gamma_{risk} = 0.4$.

Scalability Analysis. Table 3 shows the average computation times (in seconds) taken by CA-TP$^+$ and by the method in [13] for building a prediction model, as well as the number of clusters found in the first case (for completeness) – obviously, the second method does not perform any clustering of the log. Again, different abstraction methods abs_h^{mode} and a fixed value of σ were considered in the tests, which were all performed on a dedicated computer, equipped with an Intel dual-core processor and a 2GB (DDR2 1033 MHz) RAM, and running Windows XP Professional. For both methods, the real computation times are reported in the columns denoted by $Time$. Conversely, $Time_{par}$ corresponds to the time that would be spent in a virtual scenario, where an idealistic "overhead-free" parallelization of our approach is

Table 3. Number of clusters found by CA-TP$^+$, and computation times for CA-TP$^+$ and the baseline method, for different abstraction functions asb_h^{mode} and $\sigma = 0.4$

Parameters (abs_h^{mode})		**CA-TP$^+$**			**FSM [13]**
mode	h	*Cluster#*	*Time* [sec]	$Time_{par}$ [sec]	*Time* [sec]
LIST	1	9	16.8	7.4	3.9
	2	51.3	20.0	9.7	5.6
	4	63.8	19.6	7.9	10.7
	8	57.9	20.2	8.1	16.0
	16	57.9	92.3	32.6	89.8
	Total	**46.8**	**32**	**13.1**	**25.2**
BAG	1	9	17	7.3	4.0
	2	50.7	19.7	9.6	5.5
	4	64	18.7	7.6	8.4
	8	57.9	19.8	8.0	10.6
	16	57.9	79.0	36.0	32.3
	Total	**46.68**	**30.9**	**13.7**	**12.2**
Grand Total		**46.74**	**31.4**	**13.4**	**18.7**

used for concurrently learning the $A\text{-}FSM$ models of all trace clusters. Although, as expected, our approach takes always longer times than the baseline method, the former achieves a satisfactory trade-off between effectiveness and efficiency. We are further comforted by the idealistic estimates $Time_{par}$, which let us be confident in the possibility of strengthen the scalability of our approach by resorting to a parallel implementation of it.

6 Conclusions

In this paper we have proposed an ad-hoc predictive clustering approach to the discovery of performance-oriented models, capable to provide performance forecasts at run time. Several innovative features distinguish our proposal from current literature. In particular, by automatically reckoning process variants with different performance patterns, prediction accuracy can be improved considerably, as witnessed by test results in the paper. Further, as the clustering model is represented via logical rules, the discovered process/context variants can be easily interpreted and validated. This makes our approach helpful in the ex-post analysis (revision, and consolidation) of tacit context-adaptation policies, and in the design of contextualized process models, capable to adapt to context changes. The methodology has been implemented as a plug-in in the ProM framework and validated on a real case study. Empirical results confirm the efficacy of the approach in predicting processing times, and in helping foresee SLA violations, as well as its scalability. As future work, we plan to investigate on making tighter the link between the clustering phase and the induction of cluster predictors, and on the usage of novel methods both for defining environment-related context variables, and for selecting performance-relevant space abstraction, as well as on combining our approach with other basic performance prediction methods (e.g., [6]), and adopting more refined models capable to capture concurrent behaviors more effectively. In particular, it is worth considering the possibility to automatically abstract and merge together similar states (e.g., by suitably

extending methods like those in [4]), in order to obtain more compact and generalized intra-cluster proces models. Moreover, further efforts are needed in order to make the approach fully exploitable in practical application contexts. In particular, it would be beneficial to complement the prediction of SLA violations with explanations and suggestions about possible remedial actions.

References

1. CLUS: A predictive clustering system, `http://dtai.cs.kuleuven.be/clus/`
2. Blockeel, H., Raedt, L.D.: Top-down induction of first-order logical decision trees. Artificial Intelligence 101(1-2), 285–297 (1998)
3. Blockeel, H., Raedt, L.D., Ramon, J.: Top-down induction of clustering trees. In: Proc. of 15th Intl. Conference on Machine Learning (ICML1998). pp. 55–63 (1998)
4. Caragea, C., Silvescu, A., Caragea, D., Honavar, V.: Abstraction augmented Markov models. In: Proc. of 2010 IEEE Int. Conf. on Data Mining (ICDM 2010), pp. 68–77 (2010)
5. Conforti, R., Fortino, G., La Rosa, M., ter Hofstede, A.H.M.: History-Aware, Real-Time Risk Detection in Business Processes. In: Meersman, R., Dillon, T., Herrero, P., Kumar, A., Reichert, M., Qing, L., Ooi, B.-C., Damiani, E., Schmidt, D.C., White, J., Hauswirth, M., Hitzler, P., Mohania, M. (eds.) OTM 2011, Part I. LNCS, vol. 7044, pp. 100–118. Springer, Heidelberg (2011)
6. van Dongen, B.F., Crooy, R.A., van der Aalst, W.M.P.: Cycle Time Prediction: When Will This Case Finally Be Finished? In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part I. LNCS, vol. 5331, pp. 319–336. Springer, Heidelberg (2008)
7. Folino, F., Greco, G., Guzzo, A., Pontieri, L.: Mining usage scenarios in business processes: Outlier-aware discovery and run-time prediction. Data & Knowledge Engineering 70(12), 1005–1029 (2011)
8. Greco, G., Guzzo, A., Pontieri, L., Saccà, D.: Discovering expressive process models by clustering log traces. IEEE Trans. on Knowl. and Data Engineering 18(8), 1010–1027 (2006)
9. Schonenberg, H., Weber, B., van Dongen, B.F., van der Aalst, W.M.P.: Supporting Flexible Processes through Recommendations Based on History. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 51–66. Springer, Heidelberg (2008)
10. Song, M., Günther, C.W., van der Aalst, W.M.P.: Trace Clustering in Process Mining. In: Ardagna, D., Mecella, M., Yang, J. (eds.) BPM 2008 Workshops. LNBIP, vol. 17, pp. 109–120. Springer, Heidelberg (2009)
11. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: a survey of issues and approaches. Data & Knowledge Engineering 47(2), 237–267 (2003)
12. van der Aalst, W.M.P., van Dongen, B.F., Günther, C.W., Mans, R.S., de Medeiros, A.K.A., Rozinat, A., Rubin, V., Song, M., Verbeek, H.M.W., Weijters, A.J.M.M.T.: ProM 4.0: Comprehensive Support for *Real* Process Analysis. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 484–494. Springer, Heidelberg (2007)
13. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Information Systems 36(2), 450–475 (2011)
14. de la Vara, J.L., Ali, R., Dalpiaz, F., Sánchez, J., Giorgini, P.: COMPRO: A Methodological Approach for Business Process Contextualisation. In: Meersman, R., Dillon, T.S., Herrero, P. (eds.) OTM 2010, Part I. LNCS, vol. 6426, pp. 132–149. Springer, Heidelberg (2010)