Mining Predictive Process Models out of Low-level Multidimensional Logs

Francesco Folino, Massimo Guarascio, and Luigi Pontieri

National Research Council of Italy (CNR),
via Pietro Bucci 41C, I87036 Rende (CS), Italy
{ffolino,guarascio,pontieri}@icar.cnr.it

Abstract. Process Mining techniques have been gaining attention, especially as concerns the discovery of predictive process models. Traditionally focused on workflows, they usually assume that process tasks are clearly specified, and referred to in the logs. This limits however their application to many real-life BPM environments (e.g. issue tracking systems) where the traced events do not match any predefined task, but yet keep lots of context data. In order to make the usage of predictive process mining to such logs more effective and easier, we devise a new approach, combining the discovery of different execution scenarios with the automatic abstraction of log events. The approach has been integrated in a prototype system, supporting the discovery, evaluation and reuse of predictive process models. Tests on real-life data show that the approach achieves compelling prediction accuracy w.r.t. state-of-the-art methods, and finds interesting activities' and process variants' descriptions.

Keywords: Business Process Analysis, Data Mining, Prediction.

1 Introduction

Process Mining techniques aim at extracting useful information from historical process logs, possibly in the form of descriptive or predictive process models, which can support the analysis, design, and improvement of business processes. An emerging research stream [1,9,4] concerns the induction of models for predicting a given performance measure for new cases at run time.

Originally focused on workflow systems, Process Mining research has been moving towards less structured processes, possibly featuring a wide variety of behaviors and many low-level tasks. This calls for enhancing classical approaches with the capability to capture diverse execution scenarios (a.k.a. "process variants"), and to map log events to high-level activity concepts [3], in order to prevent the construction of useless models giving a cumbersome and undergeneralized view of process behavior.

The need of providing expressive process views is also witnessed by the proliferation of works on activity abstraction [12,11,7] and on log clustering [14,9,4], as well as by recent efforts to model different process variants and their link to

M. Jarke et al. (Eds.): CAiSE 2014, LNCS 8484, pp. 533–547, 2014.

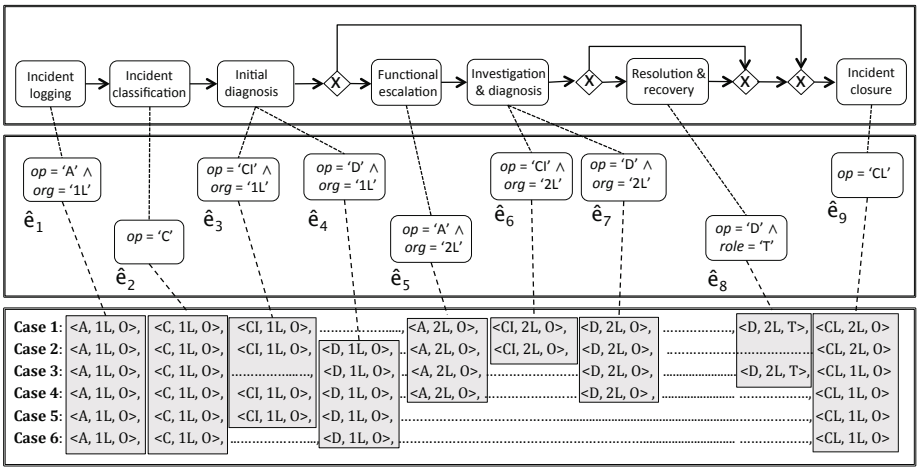

Fig. 1. A simplified Incident Management scenario: unknown high-level tasks (top), low-level event log (bottom), and data-driven event classes (middle)

environmental factors [13]. Some works also tried to combine trace clustering and activity abstraction, in order to build expressive process models, with different process variants represented via high-level tasks (or sub-processes, at different levels of aggregation/abstraction) [10,8,6].

Unfortunately, most of these methods assume that the log events are mapped to well defined process tasks, which is not true in many real-life collaborative work environments (such as issue tracking systems, or transaction systems), where lots of data are yet stored (in the form of event/case attributes) that might well help model/predict system behaviour.

Example 1. As an example scenario, consider the simplified version of the ITIL's "Incident Management" process (inspired to [3]) at the top of Figure 1. Assume that none of the high-level process tasks appear in the execution traces (like those at the bottom of the figure), where each event just stores low-level information as a triple $\langle op, org, role \rangle$. Specifically, attribute *op*, which encodes rather generically the operation performed, can take one of the following values: `A` (allocating the incident to someone), `C` (classifying the incident), `D` (describing the incident or its solution), `CI` (associating the incident with a configuration item) and `CL` (closing the incident). Attribute *org* tells which organizational entity the executor belongs to: `1L` (early intervention) and `2L` (second level team). Finally, the stored executor role can be either `O` (*operator*) or `T` (technician). ◁

In the log of lowly structured processes, like those above, none of the event attributes fully capture the semantics of all performed actions. In such a case, abstracting each event as a fixed combination of multiple attributes is likely to yield overfitting models, seeing as a very high number of distinct state/activity representations may be produced, most of which just cover a bunch of log events.

Conversely, simple event abstraction strategies (consisting in replacing each log event with the associated task or executor or them both) tend to be adopted in the area of predictive process mining, where the combination of model induction and automated activity abstraction has not been investigated so far.

Contribution To overcome the above limitations, we state the prediction of process performances as the search for an enhanced kind of performance model, consisting of three components: *(a)* an event classification function, for abstracting each low-level event into an event class, regarded as a distinct activity type (rather than as a subprocess or macro-activity, as in [12,11,7]); *(b)* a trace classification function, for discriminating among different process variants, based on case data; and *(c)* a collection of state-aware predictors, each associated with one process variant. The event classification function is meant to partition all (low-level) log events into clusters, like those in the middle layer of Figure 1, each associated with a classification rule (over event attributes). Analogous rules (over case attributes) can be found to partition all process traces into execution classes. Such a model supports the forecast of the analyzed performance measure on an ongoing process case τ via three logical steps: *(i)* each τ's event is abstracted by the event classification function; *(ii)* the obtained abstract trace is assigned to a process variant through the trace classification function and *(iii)* the predictor of the selected variant is eventually used to predict the performance of τ.

A discovery algorithm is presented in the paper for inducing both classification functions by way of a co-clustering scheme (extending the logics-based framework of *predictive clustering* [5]), prior to building a local predictor for each trace cluster. Besides enjoying compelling prediction accuracy (w.r.t. current methods, combined with usual log abstractions), in real-life application cases the approach managed to recognize relevant activity patterns at the right abstraction level. Moreover, the descriptive nature of the discovered classification rules (expressed in terms of event/trace data) can help comprehend process behaviors, and how its performances depend on both context factors and activity patterns.

Organization. The rest of the paper is structured as follows. After introducing some preliminary concepts in Section 2, we present, in Section 3, our solution approach and a prototype system implementing it. An empirical analysis on two real-life case studies is then discussed in Section 4, before drawing some concluding remarks and future work directions.

2 Preliminaries

Let us denote by E and $\mathcal{T}$ the universes of all possible events and (both fully unfolded and partial) traces, respectively, for the process under analysis – as usual, we assume that a *trace* is recorded for each process instance (a.k.a "case"), encoding the sequence of *events* happened during its enactment. For our purposes, an event $e \in E$ is regarded as a tuple storing a case identifier and a timestamp, denoted by $case(e)$ and $time(e)$, as well as a vector $prop(e)$ of data properties,

in some given space of event attributes. For example, in structured process management settings (like those handled by WfMSs), any event keeps information on both the task performed and the executor. However, as discussed above, this does not happen in many flexible BPM environments (e.g., issue/project management systems), where the tasks are not precisely conceptualized, or they just represent generic operations (e.g., update a document or exchange a message).

For each trace $\tau \in \mathcal{T}$, let $\tau[i]$ be the i-th event of τ, and let $\tau(i] \in \mathcal{T}$ be the *prefix* trace consisting of its first i events, for $i = 1 \,..\, len(\tau)$, where $len(\tau)$ is the number of events stored in τ. Moreover, let $prop(\tau)$ be a vector of data properties associated with any trace $\tau \in \mathcal{T}$, possibly including "environmental" variables characterizing the state of the BPM system (as proposed in [9]).

A *log* L (over $\mathcal{T}$) is a finite subset of $\mathcal{T}$, while the *prefix set* of L, denoted by $\mathcal{P}(L)$, is the set of all prefix traces that can be extracted from L. Finally, $events(L)$ indicates the set of all events stored in (some trace of) L.

Let us denote by $\lambda : \mathcal{T} \rightarrow \mathbb{R}$ the (unknown) performance measure (targeted by our predictive analysis) that virtually assigns a performance value to any (possibly partial) trace – for the sake of concreteness, and w.l.o.g., λ is assumed to range over real numbers. Two examples of such a measure are the remaining processing time and steps of a trace, i.e. the time and steps, respectively, needed to complete the respective process enactment.

A (predictive) *Process Performance Model (PPM)* is a model estimating the performance value of any process instance, based only on its current trace. Such a model can be viewed as a function $\tilde{\lambda} : \mathcal{T} \rightarrow \mathbb{R}$ that approximates λ all over the trace universe — including the prefix traces of unfinished enactments. Discovering such a model is an induction problem, where a given log L is used as a training set, and the λ is known for each (sub-)trace $\tau \in \mathcal{P}(L)$.

In order to focus on relevant facets of events, current approaches rely all on some abstraction functions such as those defined below (similarly to [1,9]).

Definition 1 (Event/Trace Abstraction). Let $\mathcal{T}$ be a trace universe, and E be its associated event universe. Let $\hat{E} = \{\hat{e}_1, \ldots, \hat{e}_k\}$ be a given set of abstract event representations. An *event abstraction function* $\mathcal{E} : E \rightarrow \hat{E}$ is a function mapping each event $e \in E$ to $\mathcal{E}(e) \in \hat{E}$, based on e's properties. Moreover, the trace abstraction function $abs^{\mathcal{E}} : \mathcal{T} \rightarrow \mathbb{N}^n$ is defined as follows: $abs^{\mathcal{E}}(\tau) = \langle\, count(\hat{e}_1, \tau), \ldots, count(\hat{e}_k, \tau) \,\rangle$, where $count(\hat{e}_i, \tau) = |\{\, i \in \{1, \ldots, len(\tau)\} \mid \mathcal{E}(\tau[i]) = \hat{e}_i \,\}|$, for any trace $\tau \in \mathcal{T}$. □

For example, let τ_1 be the complete trace associated with the first case (Case 1) in the log of Figure 1, and let $\mathcal{E}_{op}$ be an event abstraction function that replaces each event with its first field *op*. Then, by applying function $abs^{\mathcal{E}_{op}}$ to the (prefix) traces $\tau_1(1]$, $\tau_1(2]$, $\tau_1(3]$ and $\tau_1(4]$, four tuples are obtained, which encode the multi-sets $[\mathtt{A}]$, $[\mathtt{A}, \mathtt{C}]$, $[\mathtt{A}, \mathtt{C}, \mathtt{CI}]$, and $[\mathtt{A}^2, \mathtt{C}, \mathtt{CI}]$, respectively.

More expressive trace abstraction schemes can be defined, as in [1], which take account for the ordering of events, and possibly discard less recent ones (according to a horizon threshold). However, such an issue is not considered in this work, which mainly aims at studying how a given (even simple) trace abstraction scheme can be enhanced through ad-hoc event/trace clustering methods.

Based on such trace abstractions, an annotated finite state machine ("AFSM") can be derived as in [1], where each node corresponds to one abstract trace representation (produced by $abs^{\mathcal{E}}$) and stores an estimate for the target measure, while each transition is labelled with an event abstraction (produced by $\mathcal{E}$). Alternatively, classic regression methods can be used to extract a PPM from a propositional encoding of the log, like that in Definition 1.

Basic PPM learning methods were recently hybridized with Predictive Clustering techniques [5], which partition a data set into clusters, while assuming that all data instances own two kinds of features: *target* attributes (to be predicted), and *descriptive* attributes (used to define logical splits). Specifically, such a clustering procedure was combined with the AFSM method [9] and standard regression methods [4], with context data used as descriptive trace attributes.

3 Approach and Implementation

Clearly, the effectiveness of current performance mining approaches strongly depends on the capability of the event abstraction function $\mathcal{E}$ to focus on those properties of log events that are really connected with the behavior (and performances) of the process at hand. Unfortunately, the common solution of abstracting each event into a task's and/or an executor's label does not fit the case of logs storing fine grain records (corresponding to low-level and generic operations), where none of the event properties is suitable for making an effective event abstraction. For instance, in the case of Example 1, considering just the kind of operation performed leads to excessive information loss. Conversely, defining abstract activities as the mere combination of multiple properties (e.g., regarding each distinct triple in Example 1 as an activity) may yield a cumbersome and ineffective representation of process states (as proven empirically in our experimentation). On the other hand, many real-life tracing systems keep lots of context information for each process case, which may be used to build precise and articulated performance prediction models like those in [9].

In order to fully exploit the variety of (events' and cases') data stored in a process log, we try to build an expressive performance model for the process, hinging on two interrelated classification models: one allowing to grasp the right level of abstraction over log events, and the other encoding the business rules that determine each variant. A precise definition of such a model is given below.

Definition 2 (`CCPM`). Let L be a log, over an event (resp., trace) universe E (resp., $\mathcal{T}$). Then, a *Co-Clustering Performance Model* (`CCPM`) for L is a triple of the form $M = \langle \mathcal{C}_E, \mathcal{C}_\mathcal{T}, \Lambda \rangle$, where: *(i)* $\mathcal{C}_E : E \to \mathbb{N}$ is a partitioning function over E; *(ii)* $\mathcal{C}_\mathcal{T} : \mathcal{T} \to \mathbb{N}$ is a partitioning function over $\mathcal{T}$; and *(iii)* $\Lambda = \langle \lambda_1, \ldots, \lambda_q \rangle$ is a list of PPMs, all using $\mathcal{C}_E$ as event abstraction function, such that q is the number of clusters produced by $\mathcal{C}_\mathcal{T}$, and λ_i is the model of the i-th cluster, for $i \in \{1, \ldots, q\}$. The overall prediction function encoded by M (denoted by the same symbol M, for shortness) is: $M(\tau) = \lambda_j(\tau)$, where $j{=}\mathcal{C}_\mathcal{T}(\tau)$. □

Conceptually, a forecast for any new process instance τ can be made with the help of such a model, in three steps: *(i)* an abstract representation of τ is obtained

Input: A log L (over some trace universe $\mathcal{T}$) with an associated target measure λ, max. iterations' number $maxIter \in \mathbb{N}$, min. (relative) loss reduction $\gamma \in (0,1]$, max. number $maxCl_E, maxCl_T \in \mathbb{N} \cup \{\infty\}$ of (events', resp. traces') clusters, min. cluster coverages $\sigma \in (0,1]$ (for both events and traces).
Output: A CCPM model for L (fully encoding λ all over $\mathcal{T}$).
Method: Perform the following steps:

```
1 set C_T^(0) := {L};  C_E^(0) := {events(L)};  Err^(0) := ∞;  k := 0;
2 do
3    k := k+1;
4    EV := V_E(L, C_T^(k-1)); // build an e-view for L w.r.t. C_T^(k-1) (cf. Def. 4)
5    C_E^(k) := minePCM(EV, σ, maxCl_E) ; // induce a novel event clustering model
6    TV := V_T(L, C_E^(k)); // build a t-view for L w.r.t. C_E^(k) (cf. Def. 3)
7    C_T^(k) := minePCM(TV, σ, maxCl_T) ; // induce a novel trace clustering model
8    let Err^(k) = Loss(C_E^(k), C_T^(k), L); // estimate current prediction error
9    improved := Err^(k-1) − Err^(k) ≤ γ^(k) × Err^(k-1);
10 while k ≤ maxIter and improved;
11 if improved then  C_T := C_T^(k);  C_E := C_E^(k);
12                 else  C_T := C_T^(k-1);  C_E := C_E^(k-1);
13 let TC = ⟨t̂_1, ..., t̂_q⟩ be the list of trace clusters produced by C_T on P(L);
14 for each t̂_i in TC do
15    λ_i := minePPM(t̂_i, C_E);
16 end
17 return ⟨C_E, C_T, ⟨λ_1, ..., λ_q⟩⟩
```

Fig. 2. Algorithm CCD

in the form of a vector (as specified in Definition 1) that summarizes both its context data and structure, with each event abstracted into an event class via $\mathcal{C}_E$; *(ii)* τ is assigned to a trace cluster (representing a particular execution scenario for the process) via function $\mathcal{C}_\mathcal{T}$; *(iii)* the predictor of the chosen cluster is used to make a forecast for τ, by providing it with $abs^{\mathcal{C}_E}(\tau)$. The functions $\mathcal{C}_E$ and $\mathcal{C}_\mathcal{T}$, encoding two different classification models (defined over descriptive attributes), are hence exploited to abstract raw log events into high-level classes, and to discriminate among different process variants, respectively.

3.1 Solution Algorithm

In principle, one might seek an optimal CCPM for a given log L by trying to minimize some suitable loss measure (comparing the actual performance of each trace with the corresponding estimate). By contrast, in order to avoid prohibitive computation times across such a large search space, we rephrase the discovery problem into two simpler ones: *(1)* find a locally optimal pair of classification functions $\mathcal{C}_E$ and $\mathcal{C}_\mathcal{T}$, and *(2)* derive a collection of cluster-wise PPM predictors.

Our solution approach is summarized in Figure 2 as an algorithm, named CCD ("Co-Clustering based Discovery"). Since the quality of the trace clustering model $\mathcal{C}_\mathcal{T}$ strongly depends on the chosen abstraction function $\mathcal{C}_E$, and vice versa,

we regard the first subproblem as a co-clustering one, were an optimal partition must be found for both traces and events. This problem is approached via an iterative alternate-optimization scheme, where, at each iteration k, updated versions of the two partitioning functions are computed, denoted by $\mathcal{C}_E^{(k)}$ and $\mathcal{C}_\mathcal{T}^{(k)}$, until no satisfactory loss reduction (w.r.t. the previous iteration) is achieved. Notice that, for efficiency reasons, any loss $Err^{(k)}$ is measured by accounting for the distribution of performances in each "co-cluster" as follows: $Loss(\mathcal{C}_\mathcal{T}, \mathcal{C}_E, L) = \sum_{\tau \in \mathcal{P}(L)} [\lambda(\tau) - avg(\{\lambda(\tau'') \mid \mathcal{C}_\mathcal{T}(\tau'') = \mathcal{C}_\mathcal{T}(\tau) \text{ and } \mathcal{C}_E(\tau''[len(\tau'')]) = \mathcal{C}_E(\tau[len(\tau)])\})]^2$.

Each model $\mathcal{C}_E^{(k)}$ is induced, via function `minePCM`, from an "event-oriented" view (*e-view*) EV of the input log, which encodes both event data and a series of performance measurements, computed on current trace clusters. The discovered event clustering $\mathcal{C}_E^{(k)}$ is then used, as a novel event abstraction method, to provide `minePCM` with a "trace-oriented" view (*t-view*) TV of the log, in order to eventually induce an updated trace partitioning $\mathcal{C}_\mathcal{T}^{(k)}$. In this way, any novel trace clustering takes advantage of the most recent definition of event classes, and vice versa, according to a reinforcement learning scheme. Both kinds of views are formally defined below.

Definition 3 (T-View). Let L be a log, and $\mathcal{C}_E$ be a partitioning function defined over $events(L)$. Then, a *t-view* for L w.r.t. $\mathcal{C}_E$, denoted by $\mathcal{V}_T(L, \mathcal{C}_E)$, is a relation containing, for each trace $\tau \in \mathcal{P}(L)$, a tuple $z_\tau = prop(\tau) \oplus abs^{\mathcal{C}_E}(\tau) \oplus \langle \lambda(\tau) \rangle$, where $\oplus$ stands for tuple concatenation. For any such tuple z_τ, $prop(\tau)$ and $abs^{\mathcal{C}_E}(\tau)$ are considered as descriptive features, while $\lambda(\tau)$ is the associated (unidimensional) target. □

Definition 4 (E-View). Let L be a log, $\mathcal{C}_\mathcal{T}$ be a (trace) partitioning function over $\mathcal{P}(L)$, and $\{\hat{t}_1, \ldots, \hat{t}_q\}$ be the clusters which $\mathcal{C}_\mathcal{T}$ ranges over (with $\hat{t}_i = \{\tau \in \mathcal{P}(L) \mid \mathcal{C}_\mathcal{T}(\tau) = i\}$ for $i \in \{1 \ldots q\}$). Then, an *e-view* for L w.r.t. $\mathcal{C}_\mathcal{T}$, denoted by $\mathcal{V}_E(L, \mathcal{C}_\mathcal{T})$, is a relation consisting of a tuple $z_e = prop(e) \oplus \langle val(e, \hat{t}_1), \ldots, val(e, \hat{t}_q) \rangle$ for each $e \in events(L)$, where $\oplus$ still denotes tuple concatenation, and, for any $i \in \{1, \ldots, q\}$, it is:

$$val(e, \hat{t}_i) = \begin{cases} \texttt{NULL}, \text{if } \nexists\ \tau \in \hat{t}_i \text{ s.t. } prop(\tau[len(\tau)]) = prop(e); \\ avg(\{\lambda(\tau) \mid \tau \in \hat{t}_i \text{ and } prop(\tau[len(\tau)]) = prop(e)\ \}), \text{otherwise.} \end{cases}$$

For any tuple z_e, all the fields in $prop(e)$ are regarded as descriptive attributes, and $\langle val(e, \hat{t}_1), \ldots, val(e, \hat{t}_q) \rangle$ as its associated (multidimensional) target. □

Provided with such propositional views, function `minePCM` is meant to induce a logics-based partitioning function, by applying some predictive clustering procedure to the given (target and descriptive) data. Details on how this function was implemented in our prototype system can be found in the following subsection.

Once an (locally) optimal pair of event and trace clusterings has been found, each cluster predictor λ_i, for $\in \{1, \ldots, n\}$, is eventually computed by providing function `minePPM` with all the traces assigned to the cluster, and with the event abstraction function C_E. To this end, the function converts $\hat{t}_i$ in its t-view w.r.t. $\mathcal{C}_E$, prior to applying some suitable regression method to it.

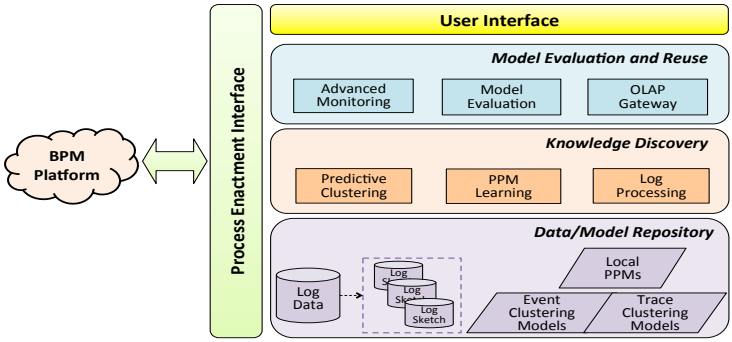

Fig. 3. Conceptual architecture of the developed prototype system

Notice that the auxiliary parameters of algorithm `CCD` are meant to give the analyst some control on both computation times ($maxIter$, γ) and the complexity of the discovered model ($maxCl_E$, $maxCl_T$). However, whatever setting is used, the algorithm is guaranteed to terminate, since the loss measure must decrease at each iteration ($\gamma > 0$), and the number of event/trace classification functions is finite. Notably, in a wide series of tests, the computation naturally finished in a few steps (less than $maxIter$), by just fixing $\gamma = 0$.

3.2 Implementation Issues and Prototype System

Functions `minePPM` and `minePCM` The current implementation of `minePPM` follows the method proposed in [5] for inducing a PCT (Predictive Clustering Tree), a logics-based predictive clustering model where the cluster assignment function is encoded in terms of decision rules (over descriptive attributes). Basically, such a model is built via a top-down partitioning scheme, where the log is split recursively, while selecting each time a descriptive attribute that locally minimizes the (weighted) average of the variances of the newly generated clusters — i.e. the 2-norm distances between the centroid of each new cluster and all instances in it. To curb the growth of the tree, an F-test based stopping criterion is used, possibly combined with a user-given upper bound to the total number clusters.

The current implementation of function `minePCM` just relies applying one of the two following standard regression methods to a propositional log view like that in Definition 1): the regression-tree induction algorithm *RepTree* and the k-NN procedure *IB-k*, both available in the popular Weka library [15].

System Prototype. The approach has been integrated into a prototype system, featuring the conceptual architecture in Figure 3, whose two lower layers leverage some core functionalities of the Process Mining framework ProM [2].

Basically, the bottommost layer is responsible for storing both historical process logs, and the different kinds of views extracted from them, as well as the different kinds of models composing each discovered `CCPM` models. All data mining and transformation mechanisms used in our approach are implemented in

Table 1. Summary features of the two application scenarios (including the associated event/trace attributes used in the tests)

Scenario	#events	#cases	Event Attributes	Trace Attributes	Target
Harbor	21484	5336	`movType`, `shift`, `area_from`, `area_to`, `vehicleType`, `block_from`, `block_to`	`service_in`, `service_out`, `imo`, `line_in`, `line_out`, `size`, `height`, `vessel_in`, `containerType`, `reefer`, `carrierType_in`, `carrierType_out`, `prevCall`, `nextCall`, `outOfGauge`, `prevCountry`, `nextCountry`	remaining time
Bug	8661	2283	`assignee`, `blocks`, `component`, `hardware`, `priority`, `product`, `resolution`, `os`, `severity`, `status`	`comments`, `votes`, `severity`, `QA`, `classification`, `component`, `URL`, `reporter`, `keywords`, `resolution`, `product`, `assignee`, `priority`, `status`, `hardware`, `flags`	remaining steps

the *Knowledge Discovery* layer, which supports the discovery of a new `CCPM`, in a interactive and iterative manner, based on the computation scheme of algorithm `CCD`. In particular, the *Predictive Clustering* and *PPM Learning* modules implement the functions `minePCM` and `minePPM` functions, respectively.

All models discovered out of a process log (i.e., traces' and events' clustering models, and the PPM models of each trace cluster) are made available to the *Model Evaluation and Reuse Layer*, which, in particular, provides the user with an easily-readable report, including the error metrics considered in our tests. The *OLAP Gateway* module is meant to reorganize historical log data into different aggregated views, in order to possibly support OLAP-like analyses.

Thanks to its predictive nature, each discovered `CCPM` can be used to configure a forecasting service for the process it was discovered for, to estimate (at run time and step-by-step) the performance outcome of any new instance of the process. Besides pure performance prediction, the *Advanced Monitoring* module supports the anticipated notification of Service Level Agreement (SLA) violations, whenever a process instance is estimated to fail a given quality requirement, previously established for one of the performance measures associated with the process.

4 Experiments

In order to assess the validity our approach, we conducted a series of tests on the logs of two real process management systems: the operational system of a maritime hub, and bug-tracking system. For readability purposes, Table 1 summarizes some features of both scenarios, indicated hereinafter as *harbor* and *bug*, respectively. Notice that a few trace attributes (e.g., `comments` and `votes`, in the *Bug* scenario) are not really known at the very beginning of a case. Clearly, such properties can be used to dynamically (re-)assign a process case to a trace cluster only if they have already taken a value. Anyway, as any trace clustering returned by our approach is ensured to cover the entire universe of traces at any step of its unfolding, each process instance falls into one of the trace clusters.

Three variants of the CCD algorithm were studied in our tests, which differ in the implementation of function minePPM: CCD-RT, using the regression-tree induction algorithm *RepTree* [15]; CCD-IBK, based on the *IB-k* procedure implemented in Weka [15]; and CCD-AVG, where each cluster predictor just returns the average performance in the cluster — the last method just serves as a baseline and quantifies co-clustering loss. In all cases, a fixed setting was used for the auxiliary parameters: $maxIter$=20, γ=0, σ=1%, and $maxCl_E$=$maxCl_T$=50. Notice that we bounded the number of event/traces to have handier process models and speed up the computation, at the cost of low precision loss (at least for CCD-RT and CCD-IBK), with respect the default setting $maxCl_E$=$maxCl_T$=∞.

For the sake of comparison, besides using the two base regressors mentioned above (denoted by RT and IBK) as baselines, we tested the FSM-based method in [1] (here named AFSM), the CATP algorithm of [9] (which reuses AFSM), and two variants of the approach in [4], denoted by as AATP-IBK and AATP-RT, using IBK and RepTree, respectively, as base learners. We remark that all competitors lack automated mechanisms for abstracting log events (into activity/action entities), and hence need the application of some a-priori event abstraction function. Conversely, all of the tested methods but AFSM can take advantage of case data.

Three standard error metrics have been used to evaluate prediction accuracy: *root mean squared error (rmse)*, *mean absolute error (mae)*, and *mean absolute percentage error (mape)*. For the sake of significance, all the error results reported next were computed via 10 fold cross-validation and averaged over 10 trials. Moreover, a statistical test was applied to check whether the methods performed really different. Specifically, for each error metrics, we used a paired two-tail Student's t-test to compare the outcomes of each method with those of the most precise one (i.e. the one achieving the lowest average error on that metric), at two different confidence levels: 95% and 99%. We then considered a method as almost equivalent to (resp., substantially worse than) the best performer if it did not differ at the 95% level (resp., it did differ at the 99% level) from the latter.

4.1 Tests on the *Harbor* Scenario

This scenario pertains the handling of containers in a maritime terminal, where a series of logistic activities are performed and traced for each container passing through the harbor. As mentioned in Example 1, each log event stores, by way of event attributes, different aspects of the logistics (move) actions performed on a container, including the followings: *(i)* the source and destination position it was moved between, in terms of yard's blocks (block_from and block_to, respectively) and areas (area_from and area_to, respectively) *(ii)* the kind of operation performed (movType), ranging over **MOV**e, **DR**ive to **B**ring, **DR**rive to **G**et, **LOAD**, **DIS**charge, **SH**u**F**fle, **OUT**; *(iii)* the type of instrument used (vehicleType), ranging from cranes to straddle-carriers and multi-trailers.

Trace attributes convey instead different properties of the handled container, ranging from the previous and next ports (prevCall and nextCall), and their associated countries (prevCountry and nextCountry), to several physical features (e.g., size and height). Like in [9], for each container, we also considered,

Table 2. Prediction results on the *harbor* scenario: errors made (over remaining times) by CCD and several competitors. For each metrics, the **best** outcome is reported in bold and underlined, while all methods **nearly equivalent** to the best one, and those *neatly worse* than it (according to T-test) are shown in bold and in italics, respectively.

Predictors		Error Measures		
Approach	*Methods*	rmse	mae	mape (%)
Algorithm CCD (Fig. 2)	CCD-IBK	**26.57±8.11**	**5.39±8.11**	**15.00±11.34**
	CCD-RT	**25.39±8.38**	**5.95±0.91**	**10.17±10.61**
	CCD-AVG	*28.58±11.44*	*8.27±1.44*	*38.10±15.86*
Competitors with the given (1-attribute) event abstraction *EA1*	AATP-IBK [4]	31.93±12.50	*7.04±1.20*	*63.62±5.65*
	AATP-RT [4]	29.95±9.67	*8.76±1.31*	*66.32±14.80*
	AFSM [1]	*80.46±11.93*	*30.74±1.40*	*279.15±26.72*
	CATP [9]	31.53±8.33	*8.35±0.60*	*58.26±26.96*
	IBK [15]	*33.66±9.37*	*7.64±0.89*	*72.50±9.85*
	RT [15]	*30.28±8.91*	*8.36±0.77*	*69.67±8.70*
Competitors with the given (5-attribute) event abstraction *EA2*	AATP-IBK [4]	*54.38±7.98*	*16.66±2.00*	*288.55±44.82*
	AATP-RT [4]	*43.84±8.08*	*16.58±1.08*	*144.19±46.31*
	AFSM [1]	*75.31±16.68*	*25.27±2.77*	*53.95±20.27*
	CATP [9]	*56.21±11.68*	*20.25±1.73*	*85.56±34.64*
	IBK [15]	*54.02±5.97*	*16.50±1.42*	*290.54±28.76*
	RT [15]	*43.33±6.04*	*15.59±0.76*	*205.06±48.89*

as a sort of environmental variables, the hour (resp., week-day, month) when it arrived, and the total number of containers ("workload") in the port at that time. The list of all events' and traces' attributes can be found in Table 1.

While our approach doesn't need any preliminary event abstraction/labelling, and it can deal with raw (multi-dimensional) event tuples, an event abstraction criterion must be defined prior to applying any other method. Two different solutions were used to this purpose in our tests: *(EA1)* abstracting each container-handling event with just the associated move type (namely, MOV, DRB, etc.); and *(EA2)* using the combination of the former five event attributes in Table 1.

Prediction Accuracy Results. Table 2 reports the (average and standard deviation for the) errors made by our methods and the competitors/baseline ones, when trying to predict the remaining processing time over a sample of 5336 containers, all exchanged with ports of the Mediterranean sea in the first third of 2006. Clearly, when faced with the challenge of dealing with complex events, according to setting *S2*, all competitors exhibit a neat worsening of results, w.r.t. the case where they were just made focus on the kinds of moves performed (setting *S1*). In fact, we verified empirically that this is the best possible single-attribute event abstraction for the given log — i.e. worse results are obtained by previous methods when abstracting the events via any other single attribute. Moreover, all base learners IBK, RT and AFSM seem to improve when embedded in a trace clustering scheme (see AAPT-IB, AAPT-RT and CATP, respectively). However, the best achievements are clearly obtained by our methods CCD-IBK and CCD-RT. Besides confirming the ability of our approach to find an effective abstraction over raw events, these results show its superiority to the two-phase (i.e. event abstraction, followed by model induction) strategy commonly used in the field of process mining, often relying on manually defined activities.

Table 3. Some event clusters (left) and trace clusters (right) returned by running `CCD-RT` on the *harbor* scenario with $\sigma = 0.01$, and $maxCl_E = maxCl_T = 50$. Incidentally, in this specific test the algorithm discovered 12 event clusters and 48 trace clusters. Cluster sizes are expressed as percentages (of all log events/traces).

id	condition	size
$\hat{e}_1$	`area_to` $\in \{C,BFS,SR\} \wedge$ `area_from` $\in \{A\text{-}NEW,T,B\text{-}NEW,\ldots\}$	12%
$\hat{e}_2$	`area_to` $\in \{C,BFS,SR\} \wedge$ `area_from` $\in \{C,CR,A,\ldots\}$	5%
$\hat{e}_3$	`area_to` $\in \{CR,BITTE\}$	17%
$\hat{e}_4$ 12.7%	`area_to` $\in \{MTR,T,GT,\ldots\}$	13%
$\hat{e}_9$	`area_to` $\in \{A\text{-}NEW,A,B\text{-}NEW,\ldots\} \wedge$ `movType` $= MOV \wedge$ `area_from` $\in \{A\text{-}NEW,A,BITTE,\ldots\}$	11%
$\hat{e}_{12}$	`area_to` $\in \{A\text{-}NEW,A,B\text{-}NEW,\ldots\} \wedge$ `movType` $\in \{OUT,DRG,LOAD,DRB,DIS,SHF\} \wedge$ `area_from` $\in \{A\text{-}NEW,A,BITTE,\ldots\}$	6%

id	condition	size
$\hat{t}_{20}$	$count(\hat{e}_1) \leq 0 \wedge count(\hat{e}_3) \leq 0 \wedge count(\hat{e}_4) \leq 0 \wedge count(\hat{e}_9) > 0 \wedge$ `nextCountry` $\in \{BG,BE,KR,\ldots\} \wedge$ `service_OUT` $\in \{ME3,GBX,\ldots\}$	6%
$\hat{t}_{33}$	$count(\hat{e}_1) \leq 0 \wedge count(\hat{e}_2) \leq 0 \wedge count(\hat{e}_3) \leq 0 \wedge count(\hat{e}_4) \leq 0 \wedge count(\hat{e}_9) \leq 0 \wedge count(\hat{e}_{12}) \leq 0 \wedge$ `nextCountry` $\in \{GE,HR,TN,\ldots\} \wedge$ `service_OUT` $\in \{AEC,GAX,\ldots\} \wedge$ `line_OUT` $\in \{CPP,CPS,SEN,\ldots\}$	4%
$\hat{t}_{44}$	$count(\hat{e}_1) \leq 0 \wedge count(\hat{e}_3) \leq 0 \wedge count(\hat{e}_4) \leq 0 \wedge count(\hat{e}_9) \leq 0 \wedge$ `nextCountry` $\in \{GR,ES,AE,\ldots\} \wedge$ `service_OUT` $\in \{EEX,BSS,\ldots\} \wedge$ `line_OUT` $\in \{MSK,APL,HLL,\ldots\} \wedge$ `prevCountry` $\in \{LB,SY,BE,EG,DZ\}$	2%

Qualitative Results. Table 3 summarizes some of the classification (i.e. partitioning) rules appearing in both clustering functions discovered with our approach (precisely, with `CCD-RT`) on the harbor scenario. Notice that these rules are quite easy to interpret and validate, and provide the analyst with a useful description of process behavior (besides supporting accurate predictions). In particular, the event clusters in the table confirm that performance-relevant activity patterns cannot be captured by just one of the event properties, nor by a fixed combination of them. Interestingly, indeed, while some event clusters just correspond to a subset of destination areas, some others also depend on the source area, or even further on the kind of move performed. On the other hand, the descriptions of trace clusters let us reckon the presence of different execution scenarios, linked to both context factors (e.g., the country of the previous/next port, or the line/service planned to bring the container) and to some of the discovered event clusters (hence playing as high-level activity patterns).

4.2 Tests on the *Bug* Scenario

As a second testing scenario, we analyzed the Eclipse project's bug repository, developed with the *Bugzilla* bug-tracking system (see *http://www.bugzilla.org*).

Essentially, each bug in the repository is associated with several fields (here regarded as trace attributes), which keep information, e.g., on: who reported the bug (`reporter`); who it has been allocated to (`assignee`); the affected software module (`component`, `product`, `version`, `hardware`); its `severity` and `priority` levels; its `status` and `resolution`. the number of comments written about the bug (`comments`). Almost all these fields can be updated as long as a bug evolves. In particular, the status of a bug can take one of the following values: *unconfirmed*, *new*, *assigned*, *resolved*, *verified*, *reopened*, and *closed*.

Table 4. Prediction results on the *bug* scenario: errors made (over remaining steps) by CCD and several competitors. The **best** result is in bold and underlined, methods **nearly equivalent** to the best one are in bold, those *neatly worse* than it in italics.

Predictors		Error Measures		
Approach	*Methods*	rmse	mae	mape (%)
Algorithm CCD (Fig. 2)	CCD-IBK	**1.369±0.666**	**0.448±0.111**	**0.167±0.010**
	CCD-RT	**1.345±0.658**	0.496±0.101	*0.210±0.008*
	CCD-AVG	*1.440±0.666*	*0.578±0.118*	0.197±0.005
Competitors, provided with ad-hoc defined activity labels	AATP-IBK [4]	**1.369±0.446**	0.472±0.143	**0.176±0.020**
	AATP-RT [4]	1.381±0.723	*0.566±0.128*	*0.767±0.134*
	AFSM [1]	*1.463±0.818*	*0.590±0.164*	*0.779±0.035*
	CATP [9]	*1.404±0.839*	*0.578±0.175*	*0.684±0.041*
	IBK [15]	1.392±0.848	0.484±0.164	*0.555±0.041*
	RT [15]	*1.499±0.787*	*0.637±0.154*	*0.873±0.020*

The `resolution` of a resolved bug can be: *fixed*, *duplicate*, *works-for-me*, *invalid*, or *won't-fix*.

As to log events, the history of a bug is kept in Bugzilla as by way of update records, possibly grouped in "bug activities", each of which gathers all changes made within a single access session. In order to let our propositional mining methods capture "simultaneous" updates, we encoded each bug activity a into an event having as many attributes as the number of (modifiable) bug fields, such that each attribute stores either *(i)* the new value assigned to the corresponding field, if it was really modified in a, or *(ii) null*, otherwise.

In order to provide the competitors with abstracted events, we tried different combinations of bug fields as possible activity labels, and empirically found that the best solution for them consists in only focusing on the changes made to the `status` (and to the `resolution` field, if modified "contemporaneously"), or to the `assignee`. In the former case, the activity label just encoded the new value assigned, without keeping any information about the person to whom a bug was (re-)assigned. The resulting abstract events look like the following activity labels: `status`:=new, `status`:=resolved + `resolution`:=fixed, `status`:=verified, etc., Δ`assignee` (simply indicating a generic change to the `assignee` field).

Prediction results. The tests were performed on a subset of the bugs created from January 1st, 2012 to April 1st, 2013, such that they were fixed at least once, but not opened and closed in the same day. Moreover, we filtered out all events (i.e. bug activities) that did not refer any of the fields `status`, `resolution`, and `assignee`. The resulting log consists of 2283 traces, with lengths (i.e. nr. of events) ranging from 2 to 25. Prediction errors on the *bug* scenario are reported in Table 4. Despite the fact that it was not provided with any suggestion on how events should be abstracted, our approach reached excellent prediction results (except when using the naïve regressor `CCD-AVG`), neatly better than all competitors, with the exception of `AATP` and, partially, of `IBK`.

Qualitative results. The models in Table 5 confirm that our approach really managed to automatically extract a suitable abstract representation for the given log events, which looks indeed very similar to the one defined for optimally

Table 5. All event clusters (left) and some of the 50 trace clusters (right) found by algorithm *CCD* on the *bug* scenario, with $\sigma = 0.01$, and $maxCl_E = maxCl_T = 50$

id	condition	size
$\hat{e}_1$	`status` = *closed*	47%
$\hat{e}_2$	`status` = *verified*	2%
$\hat{e}_3$	`status` $\in$ {*resolved*,*new*} $\wedge$ `resolution` = *fixed*	38%
$\hat{e}_4$	`status` $\in$ {*resolved*,*new*} $\wedge$ `resolution` $\in$ {*worksforme*,*invalid*}	1%
$\hat{e}_5$	`status` = *assigned*	8%
$\hat{e}_6$	`status` = *reopened*	4%

id	condition	size
$\hat{t}_1$	$count(\hat{e}_1) > 0 \wedge$ `comments` $> 6 \wedge$ `component` $\in$ {*build*, *foundation*,...}	1%
$\hat{t}_{22}$	$count(\hat{e}_1) \leq 0 \wedge 5 <$ `comments` $\leq 15 \wedge$ $count(\hat{e}_2) \leq 0 \wedge count(\hat{e}_3) > 0$ `component` $\in$ {*DBWS*, *Graphiti*,...}	4%
$\hat{t}_{47}$	$count(\hat{e}_1) \leq 0 \wedge count(\hat{e}_3) \leq 0 \wedge$ $count(\hat{e}_5) \leq 0 \wedge$ `comments` $\leq 10 \wedge$ `product` $\in$ {*Xtend*, *Aether*, *Jetty*,...} $\wedge$ `component` $\in$ {*Xpand*, *Debugger*, ...}	3%

applying the competitors, Indeed, the `status` and `resolution` attributes have been fully exploited for discriminating among event classes. Trace clusters seem to depend mainly on the number of comments associated with bugs, and on the component and/or product affected — as well as on some of the discovered event classes, here playing as high-level performance-relevant activity patterns.

5 Discussion and Conclusions

The method proposed in this paper enhances current process mining approaches for the analysis of business process performances in different respects. First of all, it removes the common assumption that all traced event logs refer explicitly (or can be easily mapped to) well defined process tasks, and allows to automatically replace the formers with high-level activity types, capturing performance behaviors at the right level of abstraction.

Empirical findings from two real application scenarios proved that the approach can achieve compelling prediction accuracy with respect to state-of-the-art process-mining methods, even when these latter are provided with a manual definition of process activities, carefully specified by an expert. We believe that prediction accuracy could be improved further by resorting to more powerful regression methods for inducing cluster-wise PPMs, in place of the straightforward ones used in our current implementation. Moreover, the descriptive power of logical event/trace partitioning rules, beside allowing for a quick validation and evaluation of the discovered models, can really help the analyst better comprehend the behavior of the process, and the way its performances depend on both context factors and activity patterns.

As to efficiency, the approach seems to work well in practice. Indeed, in the two scenarios discussed above, at most 6 co-clustering iterations were needed to find a solution (with σ=1% and $maxCl_E$=$maxCl_T$=50), and our approach only took 3.6 times longer than the quickest among the competitors — excluding `IBK`, which performs no real learning task. This ratio only became 5.4 when no finite upper bound was set for the numbers of clusters.

As future work, we plan to extend the expressive power of our event/trace classification models, and to integrate advanced regression methods for learning

cluster predictors, as well as to implement our discovery approach as a ProM [2]'s plugin, and to refine the OLAP-oriented capabilities of our prototype system.

Acknowledgments. The work was partially supported by the Italian Ministry of Education, Universities and Research (MIUR), under project *FRAME*.

References

1. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Information Systems 36(2), 450–475 (2011)
2. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W(E.), Weijters, A.J.M.M.T., van der Aalst, W.M.P.: The ProM framework: A new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 444–454. Springer, Heidelberg (2005)
3. Baier, T., Mendling, J.: Bridging abstraction layers in process mining by automated matching of events and activities. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 17–32. Springer, Heidelberg (2013)
4. Bevacqua, A., Carnuccio, M., Folino, F., Guarascio, M., Pontieri, L.: A data-driven prediction framework for analyzing and monitoring business process performances. In: ICEIS 2013, Revised Selected Papers (to appear)
5. Blockeel, H., Raedt, L.D.: Top-down induction of first-order logical decision trees. Artificial Intelligence 101(1-2), 285–297 (1998)
6. Bose, R.P.J.C., Verbeek, H.M.W., van der Aalst, W.M.P.: Discovering hierarchical process models using prom. In: CAiSE Forum (Selected Papers), pp. 33–48 (2011)
7. Jagadeesh Chandra Bose, R.P., van der Aalst, W.M.P.: Abstractions in process mining: A taxonomy of patterns. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 159–175. Springer, Heidelberg (2009)
8. Ekanayake, C.C., Dumas, M., García-Bañuelos, L., La Rosa, M.: Slice, mine and dice: Complexity-aware automated discovery of business process models. In: Daniel, F., Wang, J., Weber, B. (eds.) BPM 2013. LNCS, vol. 8094, pp. 49–64. Springer, Heidelberg (2013)
9. Folino, F., Guarascio, M., Pontieri, L.: Discovering context-aware models for predicting business process performances. In: Meersman, R., et al. (eds.) OTM 2012, Part I. LNCS, vol. 7565, pp. 287–304. Springer, Heidelberg (2012)
10. Greco, G., Guzzo, A., Pontieri, L.: Mining taxonomies of process models. Data & Knowledge Engineering 67(1), 74–102 (2008)
11. Günther, C.W., Rozinat, A., van der Aalst, W.M.P.: Activity mining by global trace segmentation. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 128–139. Springer, Heidelberg (2010)
12. Liu, D., Shen, M.: Workflow modeling for virtual processes: an order-preserving process-view approach. Information Systems 28, 505–532 (2003)
13. Milani, F., Dumas, M., Matulevičius, R.: Decomposition driven consolidation of process models. In: Salinesi, C., Norrie, M.C., Pastor, Ó. (eds.) CAiSE 2013. LNCS, vol. 7908, pp. 193–207. Springer, Heidelberg (2013)
14. 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)
15. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Kaufmann Publishers Inc. (2005)