Activity Prediction in Process Management using the WoMan Framework

Stefano Ferilli[1,2](✉), Domenico Redavid[3], and Sergio Angelastro[1]

[1] Dipartimento di Informatica – Università di Bari, Bari, Italy
{stefano.ferilli, sergio.angelastro}@uniba.it
[2] Centro Interdipartimentale di Logica ed Applicazioni – Università di Bari, Bari, Italy
[3] Artificial Brain S.r.l., Bari, Italy
redavid@abrain.it

Abstract. In addition to the classical exploitation of process models for checking process enactment conformance, a very relevant but almost neglected task concerns the prediction of which activities will be carried out next at a given moment during process execution. The outcomes of this task may allow to save time and money by taking suitable actions that facilitate the execution of those activities, may support more fundamental and critical tasks involved in automatic process management, and may provide indirect indications on the correctness and reliability of a process model. This paper proposes an enhanced declarative process model formalism and a strategy for activity prediction using the WoMan framework for workflow management. Experimental results on different domains show very interesting prediction performance.

Keywords: Process Mining, Activity Prediction, Process Model

1 Introduction & Background

A *process* consists of actions performed by agents [1, 2]. A *workflow* is a formal specification of a process. It may involve sequential, parallel, conditional, or iterative execution [12]. A process execution, compliant to a given workflow, is called a *case*. It can be described as a list of *events* (i.e., identifiable, instantaneous actions, including decisions upon the next activity to be performed), associated to *steps* (time points) and collected in *traces* [13]. Relevant events are the start and end of process executions, or of activities [2]. A *task* is a generic piece of work, defined to be executed for many cases of the same type. An *activity* is the actual execution of a task by a *resource* (an agent that can carry it out).

Process Management techniques are useful in domains where a production process must be monitored (e.g. in the industry) in order to check whether the actual behavior is compliant with a desired one. When a process model is available, new process enactments can be automatically supervised. The complexity of some application domains requires to learn automatically the process models, because building them manually would be very complex, costly and error-prone.

P. Perner (Ed.): ICDM 2017, LNAI 10357, pp. 194–208, 2017.
DOI: 10.1007/978-3-319-62701-4_15

The area of research aimed at inferring workflow models from specific examples of process executions is known as *process discovery* [2] or *process mining* [14, 9]. *Declarative* process mining approaches [11] learn models expressed in terms of a set of constraints, instead of monolithic models (usually expressed as some kind of graph, e.g. Petri Nets). More precisely, given a set of tasks T and a set of cases $C \subseteq T^*$, the aim of process mining is discovering a workflow model that fulfills the following requirements [2, 1, 8, 14]:

Completeness it can generate ('explain', 'cover') all event sequences in C;
Irredundancy it generates as few event sequences of $T^* \setminus C$ as possible;
Minimality it is as simple and compact as possible.

Having to deal with real-world environments, an additional requirement may be the ability to deal with noise [14] (i.e., the presence of wrong process executions in the training examples).

The WoMan framework [5, 4] lies at the intersection between *Declarative* Process Mining and Inductive Logic Programming (ILP) [10]. Indeed, it pervasively uses First-Order Logic as a representation formalism, that provides a great expressiveness potential and allows one to describe contextual information using relationships [6]. Differently from all previous approaches in the literature, it is *fully incremental*: not only can it refine an existing model according to new cases whenever they become available, it can even start learning from an empty model and a single case, while others need a (large) number of cases to draw significant statistics before learning starts. This allows to carry out continuous adaptation of the learned model to the actual practice efficiently, effectively and transparently to the users [4]. Experiments proved that WoMan can handle efficiently and effectively very complex processes, thanks to its powerful representation formalism and process handling operators [5, 4].

While the most classical task in Process Management is supervision of a process enactment in order to check its compliance with given model, in some context an extremely important task may be activity prediction. It may be stated as follows: given a process model and the current (partial) status of a new process execution, guess which will be the next activity that will take place in the execution. Its importance follows from the applications that it may serve. Let us give a few examples.

– Given an intermediate status of a process execution, knowing how the execution will proceed might allow the environment, or the (human or automatic) supervisor, to take suitable actions that facilitate the next activities. In industrial environments, this may bring significant savings in terms of time and money. In smart environments, considering the daily routines of people at home or at work as a process may allow the environment to provide more comfortable support to the users, improving their quality of life.
– Also, having a reliable list of expected activities to be carried out next can support the activity recognition task, which is one of the most critical requirements for automatic process management. In fact, being able to determine which high-level process-related activities are being carried out in terms

of the low-level data obtained from the sensors placed in the environment is a very complex and not yet fully solved issue.
- Another, very relevant and interesting, application of process-related predictions is in the assessment of the quality of a model. Indeed, since models are learned automatically exactly because the correct model is not available, only an empirical validation can be run. In literature, this is typically done by applying the learned model to new process enactments. Getting correct predictions when using a model may be interpreted as an indirect indication that the model is correct.

Despite its relevance, the task of activity prediction has received very little attention so far in the literature. This is possibly due to the fact that, when using traditional graph-based formalisms for expressing process models, determining the next activities may be quite simple. Indeed, reporting the current status of the process execution on the graph (e.g., as a marking of tokens in Petri Nets) allows to determine quite straightforwardly which tasks are currently enabled. Using declarative approaches the issue becomess less straightforward. Also, in traditional domains the rules that determine how the process must be carried out may be quite strict, so that predicting the process evolutions becomes a trivial consequence of conformance checking. Other, less traditional application domains (e.g., the cited routines of people), involve much more variability, and obtaining reliable predictions becomes both more difficult and more useful.

In this paper, we show how the activity prediction task can be carried out effectively using WoMan. Interesting preliminary results were obtained in [7] on various application domains using the formalism proposed in [6]. Here we extend the formalism in [6] and the approach in [7] by considering additional information in the models in order to improve the prediction performance. Full details about the extended formalism and the prediction approach are given in this paper for the first time.

This paper is organized as follows. The next section presents the WoMan (extended) formalism, while Section 3 describes in details its approach to activity prediction. Then, Section 4 reports and discusses the experimental outcomes. Finally, in Section 5, we draw some conclusions and outline future work issues.

2 The WoMan Formalism

WoMan representations [6] are based on the Logic Programming formalism, and works in Datalog, where only constants or variables are allowed as terms. Following foundational literature [1, 8], trace elements in WoMan are 7-tuples, represented in WoMan as facts

$$\texttt{entry}(T,E,W,P,A,O,R).$$

that report information about relevant events for the case they refer to:

1. T is the event timestamp (all events in a case must have different timestamps),

2. E is the type of the event (one of **begin_process**, **end_process**, **begin_activity**, **end_activity**, and **context_description**),
3. W is the name of the workflow the process refers to,
4. P is a unique identifier for each process execution,
5. A is the name of the activity,
6. O is the progressive number of occurrence of that activity in that process,
7. R (optional) specifies the agent that carries out activity A.

Activity begin and end events allow to properly handle time span and to identify concurrency in task execution, avoiding the need for inferring it by means of statistical (possibly wrong) considerations [13]. When $E =$ **context_description**, A is used to describe contextual information at time T, in the form of a conjunction of FOL atoms built on domain-specific predicates.

Given a set of training cases $\mathcal{C}$, WoMan learns a model consisting of a set of atoms built on several predicates, each expressing a different kind of constraint[4]. The core of the model, established in its very first version, is expressed by predicates `task/2` and `transition/4`.

– `task(`t`,`C_t`)` : task t occurred in training cases C_t.
– `transition(`I`,`O`,`t`,`C_t`)` : transition[5] t, occurred in training cases C_t, is enabled if all input tasks in $I = [i_1, \ldots, i_n]$ are active; if fired, after stopping the execution of all tasks in I (in any order), the execution of all output tasks in $O = [o_1, \ldots, o_m]$ is started (again, in any order). If several instances of a task can be active at the same time, I and O are multisets, and application of a transition consists in closing as many instances of active tasks as specified in I and in opening as many activations of new tasks as specified in O.

`task/2` atoms express the tasks that are allowed in the process. `transition/4` atoms express the allowed connections between activities in a very modular way. Transitions can be seen as 'consumers' of their input tasks, and 'producers' of their output tasks. In this perspective, the completion of an activity during a case can be seen as the production of a resource, that is to be consumed by some transition.

Compared to classical representations, in which the overall topology of the graph is fixed, this representation breaks the process models in several small pieces, that might in principle be recombined together in many ways (when different transitions have input multisets whose intersection is non-empty). To enforce irredundancy, WoMan exploits a number of additional information items. A fundamental one is the C_t parameter. First, and most important, it allows

[4] In the following, the extended part of the formalism with respect to [6] is marked by an asterisk '*'

[5] Note that this is a different meaning than in Petri Nets. A convenient notation for expressing transitions is

$$t : I \Rightarrow O \; [C_t]$$

where the C_t parameter can be omitted if irrelevant.

WoMan to check that all transitions involved in a new execution were all involved in the same (at least one) training case [4]. Second, it allows WoMan to compute the probability of a task or transition t, as the relative frequency $|C_t|/n$ where $n = |\mathcal{C}|$ is the number of training cases. This can be used for process simulation, for activity prediction and for noise handling (ignoring all tasks/transition in the model whose probability does not pass a specified noise threshold). Third, it allows WoMan to bound the number of repetitions of loops. Indeed, C_t is a multiset, because if a task or transition t was executed k times in case c, then C_t includes k occurrences of c. So, WoMan knows the maximum number of times that a task or transition can be executed in the same case.

Another limitation to the possible combinations of transitions is expressed using the following predicate:

* `transition_provider(`$[\tau_1, \ldots, \tau_n]$`,`t`,`q`)` : transition t, involving input tasks $I = [i_1, \ldots, i_n]$, is enabled provided that each task $i_k \in I, k = 1, \ldots, n$ was 'produced' as an output of transition τ_k, where the τ_k's are placeholders (variables) to be interpreted according to the Object Identity assumption ("terms (even variables) denoted with different symbols must be distinct (i.e., they must refer to different objects)"); several combinations can be allowed, numbered by progressive q, each encountered in cases C_{tq}.

that partitions the input multiset of a transition according to the producers of the activities to be consumed[6].

Additional constraints concern the agents that may run the activities:

- `task_agent(`t`,`A`)` : an agent, matching the roles A, can carry out task t.
- `transition_agent(`$[A'_1, \ldots, A'_n]$`,`$[A''_1, \ldots, A''_m]$`,`t`,`C_{tq}`,`q`)` : transition t, involving input tasks $I = [i_1, \ldots, i_n]$ and output tasks $O = [o_1, \ldots, o_m]$, may occur provided that each task $i_k \in I, k = 1, \ldots, n$ is carried out by an agent

[6] Let us see this through an example. Consider a model that includes, among others, the following transitions:

$$t_1 : \{x, y, z\} \Rightarrow \{a, b\} \quad ; \quad t_2 : \{x, y\} \Rightarrow \{a\} \quad ; \quad t_3 : \{x\} \Rightarrow \{a, d\}$$

and suppose that the current set of activities to be 'consumed' is $\{x, y, z\}$. If an activity a is started, any of the above transitions might be the 'consumer'. Suppose that WoMan also knows the producers of these activities: $\{x/p22, y/p21, z/p22\}$, and that the model includes the following atoms related to transitions t_1, t_2 and t_3:

`transition_provider(`$[\tau_1, \tau_1, \tau_2], t_1, 1$`).`
`transition_provider(`$[\tau_1, \tau_2, \tau_2], t_1, 2$`).`
`transition_provider(`$[\tau_1, \tau_2, \tau_1], t_1, 3$`).`
`transition_provider(`$[\tau_1, \tau_1], t_2, 1$`).`
`transition_provider(`$[\tau_1], t_3, 1$`).`

In this case, transition t_2 is not a valid consumer, since it would require that both x and y were produced by the same transition τ_1, while they were actually produced by two different transitions ($p22$ and $p21$, respectively). Conversely, pattern #3 of transition t_1 is compliant with the available producers, which makes it an eligible candidate. Also transition t_3 is enabled.

matching roles A'_k, and that each task $o_j \in O, j = 1, \dots, m$ is carried out by an agent matching roles A''_j; several combinations can be allowed, numbered by progressive q, each encountered in cases C_{tq}.

WoMan can handle taxonomies of agent roles. Each A'_k or A''_j is an expression in disjunctive normal form:

$$(r_{11} \wedge \dots \wedge r_{1n_1}) \vee \dots \vee (r_{m1} \wedge \dots \wedge r_{mn_m})$$

where each r_{ij} is an individual or a role in the taxonomy, meaning that the agent must match all roles in at least one disjunct. The conjuncts are introduced to handle multiple inheritance. The generalization/specialization relationship is handled, in that a role is considered as matched by an agent if the agent matches any of its subclasses in the taxonomy. During the mining phase, generalizing means replacing one or more roles/instances with one of their superclasses.

The new version of the WoMan formalism added the following predicates to deal with time constraints:

* `task_time`(t,$[b',b'']$,$[e',e'']$,d) : task t must begin at a time $i_b \in [b', b'']$ and end at a time $i_e \in [e', e'']$, and has average duration d;
* `transition_time`(t,$[b',b'']$,$[e',e'']$,g,d) : transition t must begin at a time $i_b \in [b', b'']$ and end at a time $i_e \in [e', e'']$; it has average duration d (from the beginning of the first activity in I to the end of the last activity in O), and requires an average time gap g between the end of the last input task in I and the activation of the first output task in O;
* `task_in_transition_time`(t,p,$[b',b'']$,$[e',e'']$,d) : task t, when run in transition p, must begin at a time $i_b \in [b', b'']$ and end at a time $i_e \in [e', e'']$, and has average duration d;

where i_b, b', b'', i_e, e', and e'' are relative to the start of the process execution, i.e. they are computed as the timestamp difference between the **begin_process** event and the event they refer to.

In addition to the exact timestamp of events, WoMan internally associates each activity in a case to a unique integer identifier, called *step*, assigned by progressive start timestamp. So, the above constraints may be expressed also in terms of steps, as follows:

* `task_step`(t,$[b',b'']$,$[e',e'']$,d) : task t must start at a step $s_b \in [b', b'']$ and end at a step $s_e \in [e', e'']$, along an average number of steps d;
* `transition_step`(t,$[b',b'']$,$[e',e'']$,g,d) : transition t must start at a step $s_b \in [b', b'']$ and end at a step $s_e \in [e', e'']$; it takes place along an average number of steps d (from the step of the first activity in I to the step of the last activity in O), and requires an average gap of g steps between the end of the last input task in I and the beginning of the first output task in O;
* `task_in_transition_step`(t,p,$[b',b'']$,$[e',e'']$,d) : task t, when run in transition p, must start at a step $s_b \in [b', b'']$ and end at a step $s_e \in [e', e'']$, along an average number of steps d;

Finally, WoMan can expresses pre- and post-conditions for tasks (in general), transitions, and tasks in the context of a specific transition. Specifically, conditions on transitions define when a transition may take place; task conditions define what must be true for a given task in general, task in transition conditions define further constraints for allowing a task to be run in the context of a specific transition (provided that its general conditions are met). They are defined as FOL rules of the following form:

- `act_`T`(`A`,`S`,`R`) :- ...` meaning that "activity A, of type T, can be run by agent R at step S of a case execution provided that ...";
- `trans_`T`(`S`) :- ...` meaning that "transition T can be run at step S of a case execution provided that ...";
- `act_`T`_in_trans_`P`(`A`,`S`,`R`) :- ...` meaning that "activity A, of type T, can be run by agent R in the context of transition P at step S of a case execution provided that ...";

where the premises '...' are conjunctions of atoms based on contextual and control flow information. Conditions are not limited to the current status of execution. They may involve the status at several steps using two predicates:

- `activity(`s`,`t`)` : at step s (unique identifier) t is executed;
- `after(`s'`,`s''`,[`n'`,`n''`],[`m'`,`m''`])` : step s'' follows step s' after a number of steps ranging between n' and n'' and after a time ranging between m' and m''.

Due to concurrency, predicate `after/3` induces a partial ordering on the set of steps. The difference between pre- and post- conditions is that premises in the former refer only to steps up to S, while in the latter they may refer to any step, both before and after S.

3 Workflow Exploitation: Supervision and Prediction

The previous section has already pointed out that different transitions may be composed in different ways with each other, and that, as a consequence, many transitions may be eligible for application at any moment, and when a new activity takes place there may be some ambiguity about which one is actually being fired. This is clear from the example in footnote 6, where each of the two proposed options would change in a different way the status of the process, as follows: firing t_1 would consume x, y and z, leaving no activity to be consumed and causing the system to wait for a later activation of b, 'produced' by t_1; firing t_2 is inhibited, because the transition providers do not match the required pattern of variables (if it were enabled, firing it would consume x and y, leaving $\{z/p22\}$ to be consumed and causing the completion of transition t_2); firing t_3 would consume x, leaving $\{y/p21, z/p22\}$ to be consumed and causing WoMan to wait for a later activation of d, 'produced' by t_3. Another example, taken from [7], is the following (for the sake of simplicity, here we will assume that the constraints on the producers are all fulfilled). Consider a model that includes, among others, the following transitions:

$$t_1 : \{x\} \Rightarrow \{a, b\} \quad ; \quad t_2 : \{x, y\} \Rightarrow \{a\} \quad ; \quad t_3 : \{w\} \Rightarrow \{d, a\}$$

Suppose that the current set of activities to be 'consumed' is $\{x, y, z\}$, and that activity a is started. It might indicate that either transition t_1 or transition t_2 have been fired. Also, if an activity d is currently being executed due to transition t_3, the current execution of a might correspond to the other output activity of transition t_3, which we are still waiting to happen to complete that transition. Each of the these options would change in a different way the process evolution, as follows: firing t_1 would consume x, leaving $\{y, z\}$ to be consumed and causing the system to wait for a later activation of b; firing t_2 would consume x and y, causing the completion of transition t_2 and leaving $\{z\}$; firing t_3 would not consume any element in the marking, but would cause the completion of transition t_3.

We call each of these alternatives a *status*. This ambiguity about different statuses that are compliant with a model at a given time of process enactment must be properly handled when supervising the process enactment. Since it can be resolved only at a later time, all the corresponding alternate evolutions of the status must be carried on by the system, and each new event must be handled with respect to each alternate status. Then, in some cases, the same ambiguity issues will arise, and more alternate evolutions will be generated; in other cases, the new event will point out that some current alternate statuses were wrong, and will cause them to be cancelled. So, as long as the process enactment proceeds, the set of alternate statuses that are compliant with the activities carried out so far can be both expanded with new branches, and pruned of all alternatives that become incompatible with the activities carried out so far.

This procedure is carried out by WoMan's supervision module, **WEST** (Workflow Enactment Supervisor and Trainer). As explained in [7], to handle this ambiguity, WEST maintains the set $\mathcal{S}$ of alternate statuses. Given a status $S \in \mathcal{S}$ and a new event e, the compliance check of the latter to the former checks may yield 3 possible outcomes:

ok : e is compliant with S;

error : there is a syntactic inconsistency in e (e.g., the termination of an activity that was never started, or the completion of a case while activities are still running); or

warning : indicating a deviation from the model that does not violate syntactic constraints; more specifically, the following types of warnings are available:
1. the pre-/post-conditions of a task, a transition or a task in the context of a transition are not fulfilled;
2. unexpected agent running a certain activity, in general or in the context of a specific transition;
3. a known task or transition, not expected at the current point of process execution, was run;
4. a new task or transition was run;

5. a task or transition was run more times than expected;
6. a task, transition or task in the context of a specific transition started or ended out of the expected time or step bounds.

Each warning carries a different degree of severity, expressed as a numeric weight. E.g., an unexpected task or transition also implies that the agent that runs it was not expected, and so has a greater severity degree than the unexpected agent alone. The degrees of severity currently embedded in WoMan for each type of warning were heuristically determined (a discussion and experimentation on how to determine these weights in order to improve performance is outside the scope of this paper).

Each status in $\mathcal{S}$ is represented as a 5-tuple $\langle M, R, C, T, W \rangle$ recording the following information:

M the 'Marking', i.e., the multiset of produced (i.e., terminated) activities associated with their provider identifier, not yet consumed (i.e. used to fire a transition);
R (for 'Ready') the multiset of activities that have not been started yet, but that are expected because they are in the output of some transition that was fired but not yet completed, each associated to the identifier of transition that produces it;
C the set of training cases that are compliant with that status;
T the sequence of (hypothesized) transitions that have been fired to reach that status;
W the multiset of warnings raised by the various events that led to this status (of course, each status may have a different multiset of warnings).

Algorithm 1 shows how the set of statuses is maintained as a consequence of the compliance check of a new event.

Given a status $S \in \mathcal{S}$, the set of candidate activities to be expected next in the case is made up of the activities in the 'Ready' component of that status and the activities that are reported in the output component of any transition that is enabled in that status. So, in principle, each status may expect a different set of activities to be carried out next. This ambiguity is more than in classical, graph-based, process model formalisms. Indeed, in those formalism, one always knows at which point of the graph the status of the current execution is located (e.g., based on the marking in Petri Nets), and thus which are the enabled tasks. On one hand, this increased ambiguity makes the activity prediction task harder, but, on the other hand, the availability of several alternate statuses, and of the associated information, allows WoMan to compute more refined statistics and to perform more elaborate reasoning to support activity prediction.

The module of WoMan, that is in charge of activity prediction, is **SNAP** (Suggester of Next Activity in Process). It exploits $\mathcal{S}$ (maintained by WEST) to determine which are the expected next activities and to rank them by some sort of likelihood, according to Algorithm 2. Specifically, components M and R of the statuses in $\mathcal{S}$ are used to determine the candidate activities, and components C and W are used to rank them by likelihood. Indeed, each status $S \in \mathcal{S}$ may have

Algorithm 1 Maintenance of the structure recording valid statuses in WEST

```
Require: $\mathcal{M}$: process model
Require: $\mathcal{S}$ : set of currently compliant statuses compatible with the case
Require: $\mathit{Running}$ : set of currently running activities
Require: $\mathit{Transitions}$: list of transitions actually carried out so far
Require: $\langle T, E, W, P, A, O, R \rangle$ : log entry
  if $E =$ begin_activity then
    $\mathit{Running} \leftarrow \mathit{Running} \cup \{A\}$
    for all $S = \langle M, R, C, T, P \rangle \in \mathcal{S}$ do
      $\mathcal{S} \leftarrow \mathcal{S} \setminus \{S\}$
      if $A \in R$ then
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{\langle M, R \setminus \{A\}, C, T, P \rangle\}$
      for all $p : I \Rightarrow O\ [C_p] \in \mathcal{M} \ni' A \in O$ do
        if $\exists \texttt{transition\_provider}(Q, p, q) \in \mathcal{M}$ : matches$(Q, M)$ then
          $P' \leftarrow P \cup P_{A,p,S}$ /* $P_{A,p,S}$ warnings raised by running $A$ in $p$ given $S$ */
          $R' = \{t'/p \mid t' \in O \wedge t' \neq A\} \cup R$
          $\mathcal{S} \leftarrow \mathcal{S} \cup \{\langle M \setminus I, R', C \cap C_p, T\&\langle p \rangle, P' \rangle\}$
  if $E =$ end_activity then
    if $A \notin \mathit{Running}$ then
      Error
    else
      $\mathit{Running} \leftarrow \mathit{Running} \setminus \{A\}$
      for all $S = \langle M, R, C, T, P \rangle \in \mathcal{S}$ do
        select transition $t : I \Rightarrow O \in T$ that produced $A$
        $S \leftarrow \langle M \cup \{A/t\}, R, C, T, P \rangle$
      if a transition $t$ has been fully carried out then
        Transitions $\leftarrow$ Transitions $\&\langle t \rangle$
        for all $S = \langle M, R, C, T, P \rangle \in \mathcal{S}$ do
          if $T \neq$ Transitions then
            $\mathcal{S} \leftarrow \mathcal{S} \setminus \{S\}$
```

where matches(Q, M) checks that provider constraint Q is fulfilled by marking M.

a different set of warnings and of compliant training cases, and this variability can be exploited to rank the next activities that are expected in the process execution by likelihood. The algorithm is organized in phases. In the first phase, all evolutions $\mathcal{S}'$ of the current set of statuses $\mathcal{S}$ that are compliant with the new event are computed. Then, the multiset N of all candidate activities to be performed next are collected from the 'Ready' component of the evolved statuses. To make the prediction more selective, only those statuses whose overall weight of warnings does not pass a given threshold can be considered. In the third step, each candidate activity is associated with a score that represents an estimation of its likelihood, computed based on the following parameters:

– number of occurrences of that activity in the computed evolutions (activities that appear more often are more likely to be carried out next);
– number of cases supporting that activity in the computed evolutions (activities expected in the statuses supported by more training cases are more likely to be carried out next); and

Algorithm 2 Activity Prediction in WoMan using SNAP

```
Require: $\mathcal{M}$: process model
Require: $\mathcal{S}$ : set of currently compliant statuses compatible with the case
Require: $E$ : current event of trace
Require: $\epsilon$: threshold to filter only more compliant statuses
  if $E$ = end_activity $\vee$ $E$ = begin_process then
    $\mathcal{S}' \leftarrow \emptyset$
    for all $S = \langle M, R, C, T, P \rangle \in \mathcal{S}$ do
      for all $p : I \Rightarrow O\ [C_p] \in \mathcal{M}$ do
        if $\exists$transition_provider$(Q, p, q) \in \mathcal{M}$ : matches$(Q, M)$ then
          $P' \leftarrow P \cup P_{p,S}$ /* $P_{p,S}$ warnings raised by firing $p$ given $S$ */
          $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{\langle M \setminus I, R \cup O, C \cap C_p, T\&\langle t \rangle, P' \rangle\}$
  if $E$ = begin_activity then
    $\mathcal{S}' \leftarrow \mathcal{S}$
  $N = \{a \mid \exists S = \langle M, R, C, T, P \rangle \in \mathcal{S}' : \delta(S) < \epsilon \wedge a \in R\}$ /* multiset of candidate
  next activities */
  $Ranking \leftarrow \{\}$
  for all $a \in N$ do
    $\mathcal{S}_a = \{\langle M, R, C, T, P \rangle \in \mathcal{S}' \mid a \in R\}$
    $\delta_a = \sum_{S \in \mathcal{S}_a} \delta(S)$ /* overall discrepancy of all statuses involving $a$ */
    $C_a = \bigcup_{\langle M,R,C,T,P \rangle \in \mathcal{S}_a} C$ /* overall set of cases supporting the execution of $a$ in
    all statuses */
    $score \leftarrow (|C_a| \cdot |\mathcal{S}_a| \cdot |a|_N)\ /\ \delta_a$
    $Ranking \leftarrow Ranking \cup \{\langle score, a \rangle\}$
Ensure: $Ranking$
```

where:

- matches(Q, M) checks that provider constraint Q is fulfilled by marking M.
- $|\cdot|$ denotes the cardinality of a set or multiset;
- $|\cdot|_M$ denotes the number of occurrences of an element in a multiset M;
- $\delta(\cdot)$ is the discrepancy of a status, computed as the sum of the weights of the warnings raised by the status.

- overall discrepancy of the statuses that expect that activity (activities expected in statuses that raised less warnings are more likely to be carried out next).

The final prediction is obtained by ranking the candidate activities by decreasing score (higher positions indicating more likelihood).

4 Evaluation

The performance of the proposed activity prediction approach was evaluated on several datasets, concerning different kinds of processes associated with different kinds and levels of complexity. The datasets related to Ambient Intelligence concern typical user behavior. Thus, they involve much more variability and

Table 1. Dataset statistics

	cases	avg events	avg activities	tasks	transitions
Aruba	220	62.67	30.34	10	92
GPItaly	253	734.56	366.28	8	79
White	158	232.71	115.35	681	4083
Black	87	243.01	120.51	663	3006
Draw	155	209.17	103.59	658	3434

subjectivity than in industrial process, and there is no 'correct' underlying model, just some kind of 'typicality' can be expected:

Aruba from the CASAS benchmark repository[7]. It includes continuous recordings of home activities of an elderly person, visited from time to time by her children, in a time span of 220 days. Each day is mapped to a case of the process representing the daily routine of the elderly person. Transitions correspond to terminating some activities and starting new activities. The resources (persons) that perform activities are unknown.

GPItaly from one of the Italian use cases of the GiraffPlus project[8] [3]. It concerns the movements of an elderly person (and occasionally other people) in the various rooms of her home along 253 days. Each day is a case of the process representing the typical movements of people in the home. Tasks correspond to rooms; transitions correspond to leaving a room and entering another.

The other concerns chess playing, where again the 'correct' model is not available:

Chess from the Italian Chess Federation website[9]. 400 reports of actual top-level matches were downloaded. Each match is a case, belonging to one of 3 processes associated to the possible match outcomes: *white* wins, *black* wins, or *draw*. A task is the occupation of a square by a specific kind of piece (e.g., "black rook in a8"). Transitions correspond to moves: each move of a player terminates some activities (since it moves pieces away from the squares they currently occupy) and starts new activities (that is, the occupation by pieces of their destination squares). The involved resources are the two players: 'white' and 'black'.

Table 1 reports statistics on the experimental datasets: number of cases, average number of events and activities per case, number of tasks and transitions in a model learned using the whole dataset as a training set. There are more cases for the Ambient Intelligence datasets than for the chess ones. However, the chess datasets involve many more different tasks and transitions, many of which are rare or even unique. The datasets are different also from a qualitative viewpoint. Aruba cases feature many short loops and some concurrency (involving up to 2

[7] `http://ailab.wsu.edu/casas/datasets.html`
[8] `http://www.giraffplus.eu`
[9] `http://scacchi.qnet.it`

Table 2. Activity prediction statistics

	Pred	Recall	Rank	Tasks	1st	Quality
Aruba	0.88	0.97	0.86	6.3	0.84	0.78
GPItaly	1.0	0.99	0.98	8.2	0.88	0.97
black	0.53	0.98	1.0	11.09	0.91	0.51
white	0.55	0.98	1.0	10.9	0.91	0.5
draw	0.65	0.98	1.0	10.6	0.91	0.64
chess	0.58	0.98	1.0	10.9	0.91	0.55

activities), optional and duplicated activities. The same holds for GPItaly, except for concurrency. The chess datasets are characterized by very high concurrency: each game starts with 32 concurrent activities (a number which is beyond the reach of many current process mining systems [5]). This number progressively decreases (but remains still high) as long as the game proceeds. Short and nested loops, optional and duplicated tasks are present as well. The number of agent and temporal constraints is not shown, since the former is at least equal, and the latter is exactly equal, to the number of tasks and transitions.

The experimental procedure was as follows. First, each dataset was translated from its original representation to the input format of WoMan. Then, a 10-fold cross-validation procedure was run for each dataset, using the learning functionality of WoMan (see [4]) to learn models for all training sets. Finally, each model was used as a reference to call WEST and SNAP on each event in the test sets: the former checked compliance of the new event and suitably updated the set of statuses associated to the current case, while the latter used the resulting set of statuses to make a prediction about the next activity that is expected in that case (as described in the previous section).

Table 2 reports average performance, on the measures reported in the columns, for the processes on the rows ('chess' refers to the average of the chess subdatasets)[10]. *Pred* is the ratio of cases in which SNAP returned a prediction. Indeed, when unknown tasks or transitions are executed in the current enactment, it assumes a new kind of process is enacted, and avoids making predictions. *Recall* is the ratio of cases in which the correct activity (i.e., the activity that is actually carried out next) is present in the ranking, among those in which a prediction was made. *Rank* reports its position in the ranking, normalized into $[0, 1]$ (1 meaning it is the first, and 0 meaning it is the last), *Tasks* is the average length of the ranking (the shorter, the better), and *1st* is the lower bound of the Rank interval associated to the first activity (*Rank* $>$ *1st* means that the activity is the first on average). $\mathit{Quality} = \mathit{Pred} \cdot \mathit{Recall} \cdot \mathit{Rank} \in [0, 1]$ is a compound index that provides an immediate indication of the overall activity prediction performance. When it is 0, it means that predictions are completely unreliable; when it is 1, it means that WoMan always makes a prediction, and that such a prediction is correct (i.e., the correct activity is at the top of the ranking).

[10] This can be considered as a baseline: fine-tuning the weights for the different kinds of warnings might result in even better performance.

As expected, the number of predictions is proportional to the number of tasks and transitions in the model. Indeed, the more variability in behaviors, the more likely it is that the test sets contain behaviors that were not present in the training sets. WoMan is almost always able to make a prediction in the Ambient Intelligence domain, which is extremely important in order to provide continuous support to the users. While much lower, the percentage of predictions in the chess domain still covers more than half of the match, the worst performance being on 'black' (the one with less training cases). In all cases, when WoMan makes a prediction, it is extremely reliable: the correct next activity is almost always present in the ranking (97-99% of the times). For chess processes, this means that WoMan is able to distinguish cases in which it can make an extremely reliable prediction from cases in which it prefers not to make a prediction at all. Also, the correct activity is always in the range associated with the top place. For the chess processes, in particular, it is always at the very top.

In conclusion, the experimental outcomes confirm that WoMan is effective on processes having very different degrees and kinds of complexity. In the Ambient Intelligence domain, this means that it may be worth spending some effort to prepare the environment in order to facilitate that activity, or to provide the user with suitable support for that activity. In the chess domain, our figures are better than those of other attempts purposely devised to apply Machine Learning to make the machine able to play autonomously.

5 Conclusions

In addition to the classical exploitation of process models for checking process enactment conformance, a very relevant but almost neglected task concerns the prediction of which activities will be carried out next at a given moment during process execution. The outcomes of this task may allow to save time and money by taking suitable actions that facilitate the execution of those activities, may support more fundamental and critical tasks involved in automatic process management, and may provide indirect indications on the correctness and reliability of a process model. This paper proposed an extended formalism and approach to make these kinds of predictions using the WoMan framework for workflow management. Experimental results on different domains show very good prediction performance, also on quite complex processes. This makes us confident that it can be successfully applied to industrial domains, as well.

Given the positive results, we plan to carry out further work on this topic. First of all, we plan to check the prediction performance on other domains, e.g. Industry 4.0 ones. Also, we will investigate how to further improve the prediction accuracy by means of more refined strategies. Finally, we would like to embed the prediction module in other applications, in order to guide their behavior.

Acknowledgments

This work was partially funded by the Italian PON 2007-2013 project PON02_00563_3489339 'Puglia@Service'.

References

1. R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In *Proceedings of the 6th International Conference on Extending Database Technology (EDBT)*, 1998.
2. J.E. Cook and A.L. Wolf. Discovering models of software processes from event-based data. Technical Report CU-CS-819-96, Department of Computer Science, University of Colorado, 1996.
3. S. Coradeschi, A. Cesta, G. Cortellessa, L. Coraci, J. Gonzalez, L. Karlsson, F. Furfari, A. Loutfi, A. Orlandini, F. Palumbo, F. Pecora, S. von Rump, Štimec, J. Ullberg, and B. tslund. Giraffplus: Combining social interaction and long term monitoring for promoting independent living. In *Proc. of the 6th International Conference on Human System Interaction (HSI)*, pages 578–585. IEEE, 2013.
4. S. Ferilli. WoMan: Logic-based workflow learning and management. *IEEE Transaction on Systems, Man and Cybernetics: Systems*, 44:744–756, 2014.
5. S. Ferilli and F. Esposito. A logic framework for incremental learning of process models. *Fundamenta Informaticae*, 128:413–443, 2013.
6. Stefano Ferilli. The woman formalism for expressing process models. In *Advances in Data Mining*, volume 9728 of *Lecture Notes in Artificial Intelligence*, pages 363–378, 2016.
7. Stefano Ferilli, Floriana Esposito, Domenico Redavid, and Sergio Angelastro. Predicting process behavior in woman. In *AI*IA 2016: Advances in Artificial Intelligence*, volume 10037 of *Lecture Notes in Artificial Intelligence*, pages 308–320, 2016.
8. J. Herbst and D. Karagiannis. An inductive approach to the acquisition and adaptation of workflow models. In *Proceedings of the IJCAI'99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business*, pages 52–57, 1999.
9. IEEE Task Force on Process Mining. Process mining manifesto. In *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. 2012.
10. S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.
11. M. Pesic and W. M. P. van der Aalst. A declarative approach for flexible business processes management. In *Proceedings of the 2006 international conference on Business Process Management Workshops*, BPM'06, pages 169–180. Springer-Verlag, 2006.
12. W.M.P. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8:21–66, 1998.
13. W.M.P. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16:1128–1142, 2004.
14. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering workflow models from event-based data. In *Proc. 11th Dutch-Belgian Conference of Machine Learning (Benelearn 2001)*, pages 93–100, 2001.