Beyond Process Mining: From the Past to Present and Future

Wil M.P. van der Aalst[1], Maja Pesic[1], and Minseok Song[2]

[1] Department of Mathematics and Computer Science,
Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, The Netherlands
w.m.p.v.d.aalst,m.pesic@tue.nl
[2] School of Technology Management
Ulsan National University of Science and Technology,
100 Banyeon-ri, Ulju-gun, Ulsan Metropolitan City, 689-798, South Korea
minseok.song@gmail.com

Abstract. Traditionally, process mining has been used to extract models from event logs and to check or extend existing models. This has shown to be useful for improving processes and their IT support. Process mining techniques analyze historic information hidden in event logs to provide surprising insights for managers, system developers, auditors, and end users. However, thus far, *process mining is mainly used in an offline fashion and not for operational decision support*. While existing process mining techniques focus on the process as a whole, this paper focuses on individual process instances (cases) that have not yet completed. For these running cases, process mining can used to *check conformance*, *predict the future*, and *recommend appropriate actions*. This paper presents a framework for operational support using process mining and details a coherent set of approaches that focuses on time information. *Time-based operational support* can be used to detect deadline violations, predict the remaining processing time, and recommend activities that minimize flow times. All of this has been implemented in *ProM* and initial experiences using this toolset are reported in this paper.

1 Introduction

Processes are everywhere. Organizations have business processes to manufacture products, provide services, purchase goods, handle applications, etc. Also in our daily lives we are involved in a variety of processes, for example when we use our car or when we book a trip via the Internet. More and more information about these processes is captured in the form of *event logs*. Contemporary systems, ranging from copiers and medical devices to enterprise information systems and cloud infrastructures, record massive amounts of events. These events can be used to make processes visible. Using *process mining* techniques it is possible to discover processes [2, 5]. Moreover, event logs can be checked to assess conformance/compliance with respect to defined processes and process models can be modified and extended using process mining techniques. This provides the

insights necessary to manage, control, and improve processes. Process mining has been successfully applied in a variety of domains ranging from healthcare and e-business to high-tech systems and auditing.

Despite the success of process mining, a limitation is that existing techniques are rarely used in an operational setting. Process mining is mainly used in an offline setting where historical information is analyzed without considering the *running cases*, i.e., instances of the process that have not completed yet. The goal of this paper is to demonstrate that process mining techniques can be used for *operational decision support*. Based on process models, either discovered through process mining or (partly) made by hand, we can (a) *check*, (b) *predict*, and (c) *recommend*. We can "replay" a running case on the process model and check whether the observed behavior fits. The moment the case deviates, an appropriate actor can be alerted. The process model based on historic data can also be used to make predictions for running cases, e.g., it is possible to estimate the remaining processing time and the probability of a particular outcome. Similarly, this information can be used to provide recommendations, e.g., proposing the activity that will minimize the expected costs and time.

This paper presents a general framework for operational decision support. It shows that process mining is not limited to the "Past" but can also be used for the "Present" and "Future". To make this concrete, we present a new set of approaches for *time-based operational support* implemented in our process mining tool *ProM* [1]. These approaches center around an annotated transition system that contains time information extracted from event logs. The *annotated transition system* can be used to *check* (time) conformance while cases are being executed, *predict* the remaining processing time of incomplete cases, and *recommend* appropriate activities to end users working on these cases.

In the remainder, we first present our framework for operational decision support. Then we describe a concrete application of the framework aiming at time-based operational support, its implementation in ProM, and some initial experiences. Finally, we discuss related work and conclude the paper.

2 Framework for Operational Support

To position the results presented in this paper, we first introduce the classical form of process mining typically done offline. Starting point for process mining is the so-called *event log*. An event log consists of a set of *traces*. Each trace is a sequence of *events* corresponding to a particular *case*. Note that a case represents one process instance, i.e., one run of the process/system. Each event refers to a task and typically also has a timestamp. Moreover, additional data elements, information about resources, event types, etc. may be attached to events. For example the trace $\langle A^{10}_{John}, B^{15}_{Mary}, C^{25}_{Mary}, D^{33}_{Pete}\rangle$ could represent a case for which four tasks are executed A, B, C, and D. Each event also has a timestamp and a reference to a resource. For example A^{10}_{John} refers to the execution of A by *John* at time 10. An example of a log consisting of three cases would be:

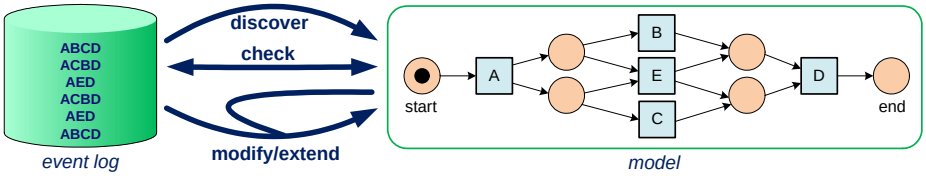


Fig. 1. Overview of classical forms of process mining: *discover*, *check*, *modify*, and *extend*.

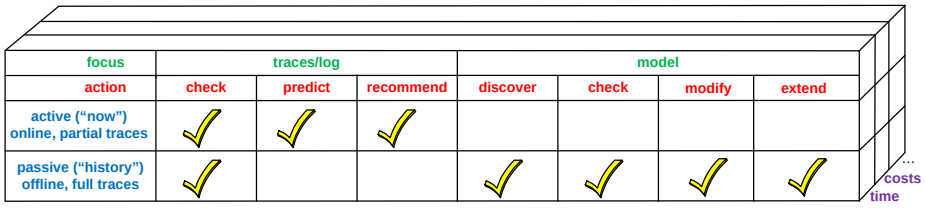


Fig. 2. Overview of the process mining spectrum distinguishing between the active use of partial traces and passive use of completed traces.

$L = \{\langle A^{10}_{John}, B^{15}_{Mary}, C^{25}_{Mary}, D^{33}_{Pete}\rangle, \langle A^{12}_{Ivan}, C^{16}_{Joe}, C^{24}_{Mary}, B^{31}_{John}\rangle, \langle A^{14}_{John}, E^{18}_{Chris}, D^{44}_{Joe}\rangle\}$.

As Figure 1 shows there are three types of process mining. First of all, there are various techniques for *process discovery* [2, 5]. These techniques aim at extracting models (for example Petri nets, EPCs, UML ADs, BPMN models) from event logs.[3] Secondly, there are techniques for *conformance checking* [16]. These techniques compare the log and the model, measure the "fit", and highlight deviations in the models. In a model with a fitness of 0.88, 88% of the events in traces can be "replayed" using the model while 12% of the events can not be "replayed" using the model.[4] In the model, it could be highlighted that a particular task happened 343 times without being enabled according to the model. Finally, there are techniques to *modify* or *extend* the model. Based on an analysis of the event log there could be suggestions to change the model, e.g., to make it better fitting. Moreover, an existing model just describing the control-flow could be extended with temporal aspects extracted from the event log. This way bottlenecks are highlighted and the extended model can be used for simulation purposes [17].

The techniques just mentioned have in common that they focus on offline analysis. This means that only *full traces* are being considered, i.e., completed cases that were handled in the past are used. Moreover, process mining is used

[3] Note that the Petri net model shown in Figure 1 was obtained by applying the α-algorithm [5] to the event log shown on the left-hand side of the figure. This is for illustration purposes only; in this paper we present a generic approach and do not favor a particular representation or discovery technique.

[4] There are various definitions of fitness [16], but this conveys the basic idea.

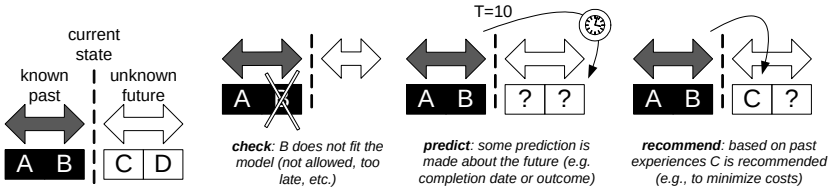


Fig. 3. Overview of operational support.

only in a *passive* manner not directly influencing the running cases. As Figure 2 shows, one can also use process mining in an online setting. Now the focus is on *partial traces*, i.e., cases that are still running and did not yet complete. For these cases, the *active* use of process mining is interesting, e.g., to check the last step executed, to predict the future of a case, and to recommend the next task to be executed. The right-hand side of Figure 2 shows the classical forms of process mining (*discover*, *check*, *modify*, and *extend*) already mentioned in Figure 1. These focus on the model rather than the traces or the log, i.e., the result is reported as a (partially) new model or annotations of an existing model. The left-hand side of Figure 2 shows other types of analysis focusing on the traces in the log rather than the model. The third dimension shown in Figure 2 shows the aspect the analysis is focusing on, e.g., time, costs, logic (routing), quality, etc.

The most likely combinations are highlighted using a ✔. Note that most of the passive forms of process mining focus on the model rather than traces. There is one exception (see the ✔in the bottom-left cell). When doing conformance checking one compares a model and an event log. The deviations can be highlighted in the model as discussed before. However, the deviations can also be shown in the event log, i.e., parts of completed traces that do not fit into the model are highlighted in the log.

In this paper, we focus on the active use of process mining involving partial traces corresponding to cases that did not complete yet. As shown in Figure 2, we identify three types of actions related to such running cases: (a) *check*, (b) *predict*, and (c) *recommend*. We refer to these actions as *operational support* as they aim at influencing the process while it is running.

Figure 3 illustrates the three types of operational support. Starting point is some model and a partial trace. Note that the model is typically learned using classical process mining techniques. The partial trace refers to a case that is running. The left-hand side of Figure 3 shows a partial trace $\langle A, B \rangle$. Note that we abstract from timestamps, resources, data, etc. For this case, we know that A and B occurred, but we do not know its future. Suppose now that the partial trace $\langle A, B \rangle$ is not possible according to the model. In this case, the operational support system would generate an alert. Another possibility would be that B took place three weeks after A while this should happen within one week. In such a case another notification could be sent to the responsible case manager. Such scenarios correspond to the *check* action mentioned before. Figure 3 also

Table 1. A fragment of an event log.

case id	task	trans.	resource	timestamp
1	*check*	complete	admin	2009-01-01 11:55:25
	advertise	complete	admin	2009-01-15 14:03:18
	inspect	complete	admin	2009-01-28 16:56:53
	decide	complete	admin	2009-02-02 09:08:03
2	*check*	complete	admin	2009-01-01 09:36:21
	process	complete	admin	2009-01-15 14:19:59
	decide	complete	admin	2009-01-20 17:47:13
...	...	...	...	...

illustrates the goal of *predictions*. Given the current state of a case, the model is used to make some kind of prediction. For example, given the $\langle A, B \rangle$ trace it could be predicted that the remaining processing time is ten days. This prediction would be based on historic information both in the partial trace and in the event log used to learn the model. Predictions are not restricted to time, but can also refer to costs, probability of a particular outcome, resource availability, etc. Closely related to predictions are *recommendations*. The main difference is that recommendations suggest the next action based on possible continuations of the case. Based on the model, one can try all possible actions and see which one would lead to the best (predicted) performance. Note that recommendations are not only used for determining the next task, but also for allocating resources to work-items or for timing a particular action.

The process mining framework *ProM* aims to support the whole spectrum shown in Figure 2. Earlier versions of ProM focused mainly on passive forms of process mining [1]. In the new version of ProM, we aim to also *support operational decision making in a generic manner*. The basic idea is that some operational system, e.g., a workflow management system, business process management system, or other Process-Aware Information System (PAIS), sends partial traces to ProM as shown in Figure 3. ProM then does the appropriate checks, generates predictions, or sends recommendations while using models derived from event logs (or alternatively use manually created models).

3 Application of the Framework to Time-based Operational Support

To illustrate that process mining is not limited to passive/offline analysis, we will use a small event log of a process for handling requests of citizens for building permits. Using this example, we present new process mining techniques that cover the whole process mining spectrum. Our example process contains five tasks: (1) *check* for checking whether the requested building permit is compliant to the regulations, (2) *advertise* for advertising the requested permit in the local newspaper for a period of 30 days, (3) *inspect* for inspecting the construction site, (4) *process* for handling requests that are not compliant with the regulations,

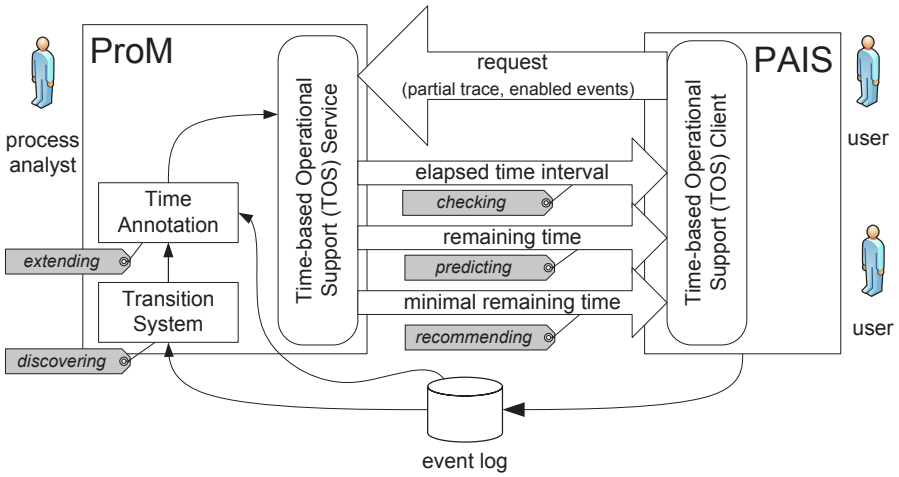


Fig. 4. Architecture of our system to support users based on time information in logs.

and (5) *decide* for deciding whether to issue or decline the permit. Table 1 shows a fragment of the log. Each line in the event log of this process corresponds to an event related to one of the five mentioned tasks. For each event, information about the task name, event type, resource that triggered the event and the timestamp is available. Moreover, each event is associated to one case, i.e., a single permit request. For example, Table 1 shows events of two cases containing four and three events, respectively. Note that, for the purpose of simplicity, in the remainder of this paper we will use only the task name to refer to one event.

In the remainder of this section we will use this log to show how process mining techniques can be used for discovering, extending, checking, predicting and recommending in the context of execution times of processes. Figure 4 shows this example and the functionalities of the ProM tool that we will use in this paper. We start with describing existing procedures (i.e., ProM plugins) for *discovering* a transition system from an event log [3] and for *extending* a transition system with time information from an event log (i.e., time annotations) [4] in sections 3.1 and 3.2, respectively. The generated transition system and time annotations can be used to provide useful information about active processes. For this purpose, we have implemented the *Time-based Operational Support* (TOS) Client and Service. The TOS Client can be used by any PAIS to request temporal information about active processes. The TOS Service uses the transition system and its time annotations to generate information about active processes and sends them to the TOS Client. The TOS Client sends the partial trace (i.e., all events executed until the moment of request) and currently enabled tasks of the running case when requesting the information from the TOS Service. The TOS Service generates three types of information about the current case. First, the TOS Service *checks* whether the elapsed time of the current case is

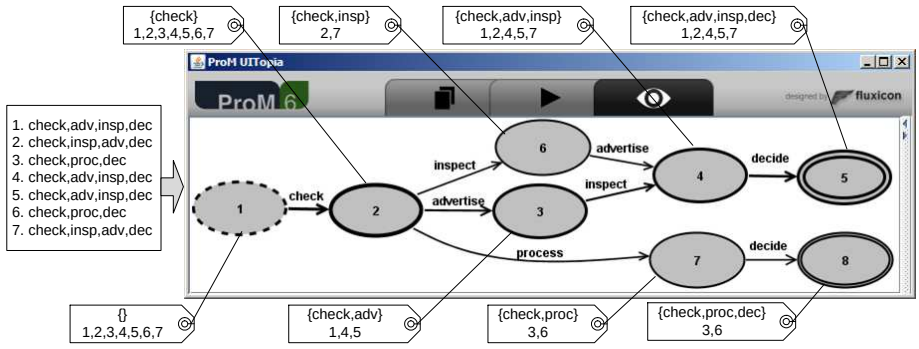


Fig. 5. A transition system constructed from an event log with seven traces.

within certain temporal boundaries calculated based on elapsed times of earlier completed cases visiting the same state (cf. Section 3.3). Second, Section 3.4 describes how the TOS Service can *predict* the remaining execution time based on the past processes. Finally, in Section 3.5 the possibility to *recommend* the enabled events that, based on historic information, are most likely to lead to minimal execution times is described.

3.1 Discovering a Transition System from History

An approach that uses various abstractions for discovering a transition system from an event log is described in [3]. The advantage of this process mining technique is that it is very flexible as it allows for a wide range of abstractions, i.e., the discovered model can be tailored towards the needs of the analyst. A transition system is a triplet (S, E, T) where S is the set of states, E is the set of event (transition) labels, and $T \subseteq S \times E \times S$ is the transition relation describing how the system can move from one state to another. For example, Figure 5 shows a ProM screen of a transition system mined from our event log with seven traces containing events referring to tasks *check*, *advertise*, *inspect*, *process* and *decide*. The transition system has eight states ($S = \{s_1, s_2, \ldots, s_8\}$), five event labels ($E = \{\mathit{check}, \mathit{advertise}, \mathit{inspect}, \mathit{decide}, \mathit{process}\}$) and eight transitions ($T = \{(s_1, \mathit{check}, s_2), (s_2, \mathit{advertise}, s_3), (s_2, \mathit{inspect}, s_6), (s_2, \mathit{process}, s_7), (s_3, \mathit{inspect}, s_4), (s_6, \mathit{advertise}, s_4), (s_7, \mathit{decide}, s_8), (s_4, \mathit{decide}, s_5)\}$).

The transition system in Figure 5 is mined from the event log using two types of abstractions [3]. First, an event abstraction is used when considering which event information is relevant. For example, the transition system shown in Figure 5 is mined using the event abstraction that considers only the task name and ignores the event type, resource and timestamp. Second, a state abstraction is used when it comes to how a sequence of events is 'replayed' on the transition system. For example, the transition system shown in Figure 5 is mined using the "set state abstraction" that considers only which tasks were executed and ignores the execution order and frequency. The tags connected to

states in Figure 5 show two types of state-related information. First, the set abstraction for the state is shown in the upper line. For example, state s_4 refers to a trace prefix that contains tasks *check*, *advertise* and *inspect* in any order. Second, the bottom line shows which traces are replayed in which state. For example, traces *1*, *2*, *4*, *5* and *7* all visit state s_4: traces *1*, *4* and *5* after executing sequence $\langle$*check*, *advertise*, *inspect*$\rangle$ and traces *2* and *7* after executing sequence $\langle$*check*, *inspect*, *advertise*$\rangle$. It is important to note that this state considers all traces where these three tasks were executed, regardless of the order.

3.2 Extending the Transition System with Time Information

Event logs can also be used to enrich models with information about past executions. An approach for annotating a transition system with time information from an event log is described in [4]. This procedure starts by replaying each trace of the event log on the transition system, and collecting three types of time information extracted from the trace for each visited state. First, the *time elapsed* from the beginning of the trace is assigned to the state as the difference between the timestamp of the current event and the timestamp of the first event in the trace. Second, the *remaining time* until the end of the trace is assigned to the state as the difference between the timestamp of the last event in the trace and the timestamp of the current event. Finally, the *sojourn time*, the time that the trace spent in this state is assigned to the state as the difference between the timestamp of the next event in the trace and the timestamp of the current event.

Figure 6 shows how elapsed, remaining and sojourn times are collected from the building permits event log and transition system in Figure 5. Note that the actual time data extracted from the event log refers to milliseconds, but for the reasons of simplicity displayed time data is rounded to days. Because s_1 is the initial state, elapsed and sojourn times for all traces are zero and remaining times are equal to the total execution times at s_1. The elapsed (remaining) times in the initial state s_1 correspond to remaining (elapsed) times in the two final states s_5 and s_8. This is expected, because the remaining time in the initial state must be equal to the elapsed time in the final state for each trace. For example, trace *1* has a total duration time of 68 days. The elapsed, remaining and sojourn times for this trace are shown as the first elements for the states that this trace visits: s_1, s_2, s_3, s_4 and s_5. While the *remaining time* value decreases from 68 in state s_1 to zero in state s_5, the elapsed time increases from zero in s_1 to 68 in state s_5. Note that, in each of these states, the sum of elapsed and remaining time is equal to the trace's total execution time. For example, the sum of elapsed and remaining times for trace *1* in each of the visited states is 68 days. The sum of sojourn times in all states one trace visits is equal to the total duration of that trace. For example, the sum of sojourn times for trace *1* is $0+6+39+23+0 = 68$ days.

The collected time information can be used to annotate each state with statistical data for elapsed, remaining and sojourn times: average, standard deviation, etc. In this paper we focus on elapsed and remaining time annotations. We have

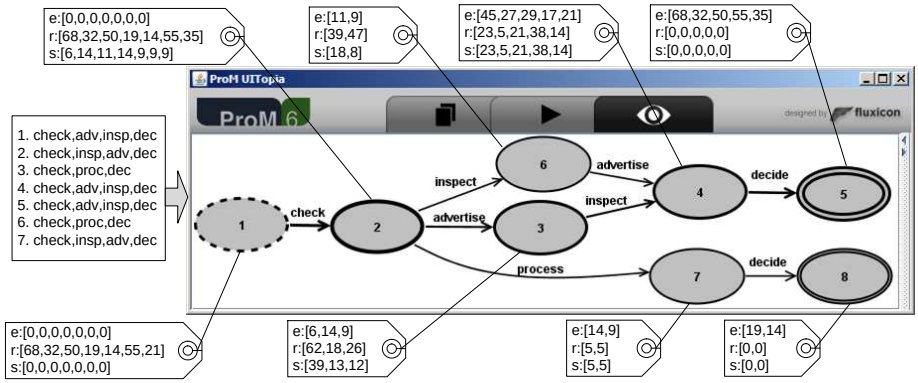


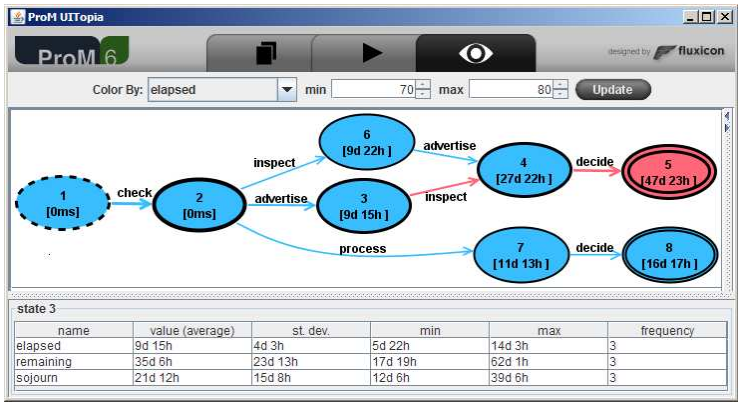
Fig. 6. Collecting elapsed (e), remaining (r) and sojourn (s) times for the transition system from Figure 5.

used time information in our example event log to annotate the transition system mined from this log (cf. Figure 5). Figure 7 shows the ProM screen with elapsed and remaining times in days and hours. For example, the average elapsed time in state s_3 is 9 days and 15 hours and average remaining time in state s_2 is 39 days and 1 hour.

3.3 Checking Running Cases

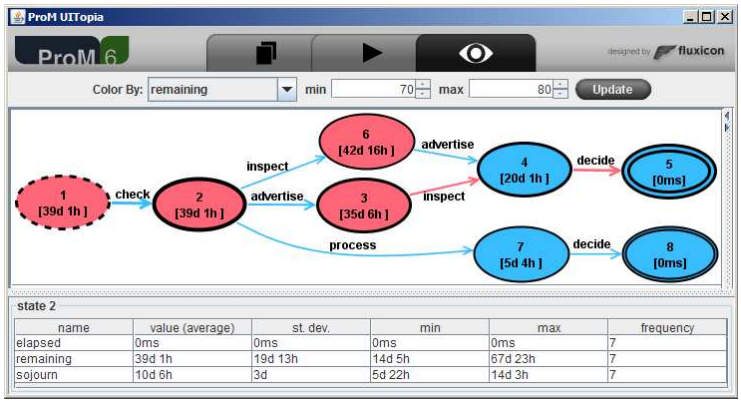
The elapsed time annotations can be used to check how fast currently running cases are being executed when compared to past cases. The procedure for checking a particular running case is as follows:

1. *Replay the partial trace* of the case under consideration on the transition system and identify the current state of the case under consideration.
2. *Calculate the a time interval* for the elapsed time in the current state. The time interval sets upper and lower bounds. There are many ways to define such a time interval. Here, three simple approaches are considered:
   - all elapsed times seen in the past: $[min, max]$,
   - a pre-defined deviation from the average elapsed time: $[\mu - C, \mu + C]$, where μ is the average elapsed time and C is a constant, and
   - standard deviation from the average elapsed time: $[\mu - C * \sigma, \mu + C * \sigma]$, where μ is the average elapsed time, C is a constant and σ is the standard deviation.
3. *Calculate the elapsed time* of the case under consideration as the difference between timestamps of the first and the last event in its partial trace.
4. *Check the execution speed* of the running case: if the elapsed time of the case is under, within or above the time interval, then the case is considered to be slower, as fast as, or faster than processes executed in the past, respectively.
5. *Alert* interested parties (e.g., employees who work on the running case, the manager, etc) if the case is too fast or too slow.



name	value (average)	st. dev.	min	max	frequency
elapsed	9d 15h	4d 3h	5d 22h	14d 3h	3
remaining	35d 6h	23d 13h	17d 19h	62d 1h	3
sojourn	21d 12h	15d 8h	12d 6h	39d 6h	3

(a) elapsed times



name	value (average)	st. dev.	min	max	frequency
elapsed	0ms	0ms	0ms	0ms	7
remaining	39d 1h	19d 13h	14d 5h	67d 23h	7
sojourn	10d 6h	3d	5d 22h	14d 3h	7

(b) remaining times

Fig. 7. Time annotations based on Figure 5 in ProM.

Consider, for example, a situation where inspectors processing permit requests want to be alerted by their TOS Clients if it takes them too long to process a particular request. Assume that the time bounds are set by the time interval $[\mu - 2 * \sigma, \mu + 2 * \sigma]$. Further, assume that an inspector is currently working on a request (i.e, running case) with the partial trace $\langle check, advertise \rangle$ where tasks *check* and *advertise* were executed on 26/10/2009 and 26/11/2009, respectively. The procedure for checking the elapsed time based on the annotated transition system shown in Figure 6 is as follows:

1. Replaying this partial trace on the transition system leads to state s_3.
2. The time interval $[\mu-2*\sigma, \mu+2*\sigma]$ is calculated for the elapsed time in state s_3. As Figure 7(a) shows, average elapsed time in this state is μ= 9 days and 15 hours and standard deviation is σ= 4 days and 3 hours. Therefore, the time interval for elapsed times in this state is [1d9h,17d21h].

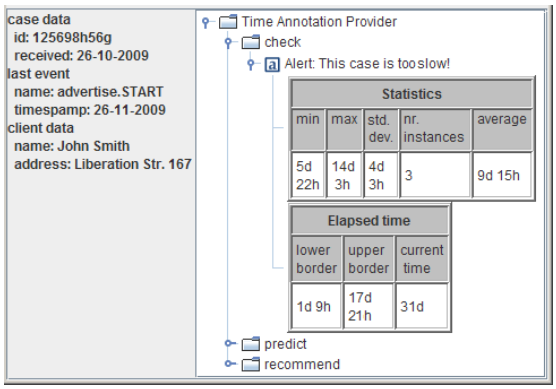


Fig. 8. Checking the elapsed time of a running case with partial trace $\langle check, advertise \rangle$.

3. The elapsed time of the current case is 31 days (time between execution of tasks *check* and *advertise*).
4. The elapsed time of the active process is above the upper bound set by the time interval.
5. Figure 8 shows the alert popping up in the TOS Client. The considered case is substantially slower than cases from the past. In addition to the alert itself, additional information about the running case and the used statistics are included to interpret the warning.

3.4 Predicting the Future of Running Cases

The remaining time annotations created from past cases can also be used to *predict* the remaining execution time of the running cases. The prediction procedure for one running case is simple:

1. *Replay the partial trace* of the case under consideration on the transition system and identify the current state of the case under consideration.
2. *Take the average or median value of the remaining time annotations in the current state* as the prediction for the remaining execution time of the case under consideration.
3. *Notify* interested parties about the predicted remaining time.

Assume, for example, that the inspectors working on building permits want to be informed by their TOS Clients regarding the expected remaining processing time. Consider, for example, a situation where a client who submitted a building permit request (for which the partial trace is $\langle check \rangle$) is interested how much longer it will take to get the final decision. By using the remaining time annotations shown in Figure 7(b), the prediction for this running case can be generated in the following way:

1. Replaying this partial trace on the transition system leads to state s_2.

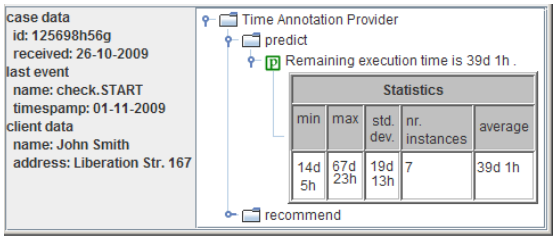


Fig. 9. Predicting the remaining time of a running case with partial trace $\langle check \rangle$.

2. The predicted remaining execution time based on the average of the remaining times in state s_2 is 39 days and 1 hour.
3. Figure 9 shows how the result presented by the TOS Client.

3.5 Recommending the Next Step for Running Cases

The remaining time annotations of a transition system can also be used to recommend steps that lead to shortest execution times in past cases. In addition to the partial trace of the running case, this procedure also uses the set of enabled events in the case to recommend which one of them should be executed:

1. For each enabled event, identify the state in which the transition system would be if this enabled event would indeed be executed in the running case using the following two steps: (a) *Create a new trace* by extending the partial trace of the running case with the enabled event under consideration; and (b) *Replay the new trace* in the transition system to identify the state to be assigned to this event.
2. *Create an ordered list of recommendations* by sorting enabled events in the increasing order of average remaining times annotated to assigned states: the state assigned to the first recommended event has a shorter (or equal) predicted remaining time than the state assigned to the second recommended event, etc.
3. *Inform* interested parties about the recommendations.

Consider, for example, a situation when inspectors working on a building permit request (i.e., running case) with partial trace $\langle check \rangle$ would like to get the recommendation whether to execute events *advertise.start* or *inspect.start* next (i.e., enabled tasks) in order to process this request as quickly as possible. Based on the remaining time annotations shown in Figure 7(b), the recommendation is generated in the following way:

1. Transition system states are assigned to enabled events *advertise.start* and *inspect.start* by extending the partial trace of the running case: (a) State s_3 is assigned to *advertise.start* because replaying trace $\langle check, advertise \rangle$ on the transition system leads to state s_3; and (b) State s_6 is assigned to

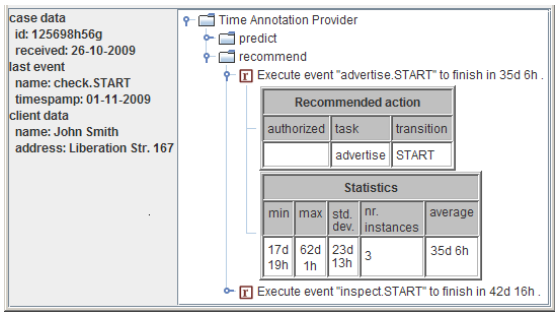


Fig. 10. Recommending the next step for a running case with partial trace $\langle check \rangle$.

inspect.start because replaying trace $\langle check, inspect \rangle$ on the transition system leads to state s_6.

2. The list of recommended events contains *advertise.start* or *inspect.start*, where *advertise.start* has higher priority than *advertise.start* or *inspect.start*, because the state s_3 has a shorter predicted remaining time (i.e., 35 days and 6 hours) than the state s_6 (i.e., 42 days and 16 hours).
3. Figure 10 shows how the recommendations are shown by the TOS Client.

4 Related Work

Lion's share of process mining research has been focusing on passive forms of process mining such as process discovery [2, 5, 3, 6, 8, 9, 12]. These serve as a basis for learning good models, but are not the focus of this paper. Conformance checking is typically also done in an off-line fashion [16] while the extension of models into full-fledged simulation models is also not used in an operational sense [17]. See www.processmining.org for more pointers to process mining literature.

There have been some initial attempts to support operational decision making using process mining techniques or simulation. In [18] both techniques are combined in the context of YAWL and in [10] non-parametric regression is used to predict completion times. A recommendation service that uses historic information for guiding the user to select the next work item has been implemented in ProM [19] and it is related to case-based reasoning [21]. A recommender for execution of business processes based on the Product Data Model (PDM) is presented in [20].

The framework has been tested using a set of plug-ins related to time-based operational support. This approach is most related to the flexible mining approach in [3] and the prediction approach in [4]. However, in this paper we present an overarching approach, and a generic implementation that does not just support prediction, but also time-based conformance checking and time-based recommendations.

There are various approaches to run time support in the context of world wide web. Some examples are monitoring based on business rules [13], BPEL [7], event calculus [14], etc. Other examples are the various types of recommender systems that support users in their decision-making [15, 22]. These systems generate recommendations based on the user's preferences/behavior and are becoming an essential part of e-commerce and information seeking activities. Similar questions are also studied in the context of intrusion detection [11].

The main contribution of this paper is that it provides a framework for positioning the various types of process mining (cf. Figure 2 on page 3) and details the aspect of operational support for running processes in a generic manner. This view is supported in the new version of ProM.

5 Conclusions

In this paper, we focus on the application of process mining to operational decision making. We presented a generic framework and described a set of ProM plug-ins for time-based operational support. The approaches are based on transition systems annotated with time information. These are used to check the timely execution of cases, predict the completion time of cases, and recommend the best steps to minimize the overall flow time. This serves as an example for a much larger set of possible techniques for operational support. In the future, we would like to add more techniques (not only related to time, but also costs, quality, compliance, etc.) and apply them actively in selected domains (most likely hospitals and municipalities). Note that the application of new techniques requires a tight integration with existing information systems.

References

1. W.M.P. van der Aalst, B.F. van Dongen et al. ProM 4.0: Comprehensive Support for Real Process Analysis. In J. Kleijn and A. Yakovlev, editors, *Petri Nets 2007*, volume 4546 of *Lecture Notes in Computer Science*, pages 484–494. Springer-Verlag, Berlin, 2007.
2. W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, and H.M.W. Verbeek. Business Process Mining: An Industrial Application. *Information Systems*, 32(5):713–732, 2007.
3. W.M.P. van der Aalst, V. Rubin, B.F. van Dongen, E. Kindler, and C.W. Günther. Process Mining: A Two-Step Approach to Balance Between Underfitting and Overfitting. *Software and Systems Modeling*, 9(1):87–111, 2010.
4. W.M.P. van der Aalst, M.H. Schonenberg, and M. Song. Time Prediction Based on Process Mining. BPM Center Report BPM-09-04, BPMcenter.org, 2009.
5. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
6. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.

7. L. Baresi, C. Ghezzi, and S. Guinea. Smart Monitors for Composed Services. In *ICSOC '04: Proceedings of the 2nd International Conference on Service Oriented Computing*, pages 193–202, New York, NY, USA, 2004. ACM Press.
8. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
9. A. Datta. Automating the Discovery of As-Is Business Process Models: Probabilistic and Algorithmic Approaches. *Information Systems Research*, 9(3):275–301, 1998.
10. B.F. van Dongen, R.A. Crooy, and W.M.P. van der Aalst. Cycle Time Prediction: When Will This Case Finally Be Finished? In R. Meersman and Z. Tari, editors, *CoopIS 2008*, volume 5331 of *Lecture Notes in Computer Science*, pages 319–336. Springer-Verlag, Berlin, 2008.
11. L. Feng, X. Guan, S. Guo, Y. Gao, and P. Liu. Predicting the Intrusion Intentions by Observing System Call Sequences. *Computers and Security*, 23(3):241–252, 2004.
12. D.R. Ferreira and D. Gillblad. Discovering Process Models from Unlabelled Event Logs. In *Business Process Management (BPM 2009)*, volume 5701 of *Lecture Notes in Computer Science*, pages 143–158. Springer-Verlag, Berlin, 2009.
13. A. Lazovik, M. Aiello, and M. Papazoglou. Associating Assertions with Business Processes and Monitoring their Execution. In *ICSOC 2004*, pages 94–104, New York, NY, USA, 2004. ACM Press.
14. K. Mahbub and G. Spanoudakis. A Framework for Requirents Monitoring of Service Based Systems. In *ICSOC 2004*, pages 84–93, New York, NY, USA, 2004. ACM Press.
15. P. Resnick and H.R. Varian. Recommender Systems. *Communications of the ACM*, 40(3):56–58, 1997.
16. A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008.
17. A. Rozinat, R.S. Mans, M. Song, and W.M.P. van der Aalst. Discovering Simulation Models. *Information Systems*, 34(3):305–327, 2009.
18. A. Rozinat, M. Wynn, W.M.P. van der Aalst, A.H.M. ter Hofstede, and C. Fidge. Workflow Simulation for Operational Decision Support. *Data and Knowledge Engineering*, 68(9):834–850, 2009.
19. H. Schonenberg, B. Weber, B.F. van Dongen, and W.M.P. van der Aalst. Supporting Flexible Processes Through Recommendations Based on History. In *International Conference on Business Process Management (BPM 2008)*, volume 5240 of *Lecture Notes in Computer Science*, pages 51–66. Springer-Verlag, Berlin, 2008.
20. I.T.P. Vanderfeesten, H.A. Reijers, and W.M.P. van der Aalst. Product Based Workflow Support: Dynamic Workflow Execution. In Z. Bellahsene and M. Léonard, editors, *CAiSE'08*, volume 5074 of *Lecture Notes in Computer Science*, pages 571–574. Springer-Verlag, Berlin, 2008.
21. B. Weber, W. Wild, and R. Breu. CBRFlow: Enabling Adaptive Workflow Management Through Conversational Case-Based Reasoning. In *Advances in Case-Based Reasoning*, volume 3155 of *Lecture Notes in Computer Science*, pages 434–448. Springer-Verlag, Berlin, 2004.
22. B. Zhou, S.C. Hui, and K. Chang. An Intelligent Recommender System Using Sequential Web Access Patterns. In *IEEE Conference on Cybernetics and Intelligent Systems*, pages 393–398, 2004.