A General Framework for Action-Oriented Process Mining

Gyunam Park and Wil M.P. van der Aalst

Process and Data Science Group (PADS), Department of Computer Science,
RWTH Aachen University, Aachen, Germany
{gnpark,wvdaalst}@pads.rwth-aachen.de

Abstract. Process mining provides techniques to extract process-centric knowledge from event data available in information systems. These techniques have been successfully adopted to solve process-related problems in diverse industries. In recent years, the attention of the process mining discipline has shifted to supporting continuous process management and actual process improvement. To this end, techniques for operational support, including predictive process monitoring, have been actively studied to monitor and influence running cases. However, the conversion from insightful diagnostics to actual actions is still left to the user (i.e., the "action part" is missing and outside the scope of today's process mining tools). In this paper, we propose a general framework for *action-oriented process mining* that supports the continuous management of operational processes and the automated execution of actions to improve the process. As proof of concept, the framework is implemented in *ProM*.

Key words: action-oriented process mining, continuous operational management, insights turned into actions, process improvement

1 Introduction

Process mining aims to discover, monitor, and improve business processes by extracting knowledge from event logs available in information systems [1]. Process mining techniques enable business managers to better understand their processes and gather insights to improve these processes. They are successfully deployed by a range of industries, including logistics, healthcare, and production [2].

Nowadays, attention in the process mining discipline is shifting to supporting continuous process management [2]. In order to manage operational processes properly, it is imperative to apply process mining techniques in a repetitive manner, rather than focusing on making a one-time report of process mining diagnostics. This repetitive application enables not only the identification of more relevant problems at stake, but also the continuous improvement of operational processes in a dynamically changing environment. The one-time report is likely to present less relevant problems in the current situation, failing to handle newly-introduced problems.

Online operational support techniques in process mining aim at enabling the continuous management of operational processes [1]. To that end, they continu-

ously monitor and analyze cases that are still running, intended for controlling the problematic process instances [3, 4]. These techniques have been effective in extracting practical diagnostics into performance and compliance problems [1]. However, they do not suggest how the diagnostics are exploited to achieve actual improvements in the operational processes.

For the actual process improvement, it is necessary to convert the insights from process mining diagnostics to management actions. For example, when a bottleneck emerges or is expected to occur, one should take actions, such as assigning more resources, alerting managers, and finding bypassing routes, to mitigate the risk caused by the problem. To fill the gap between the diagnostics and the improvement actions, in this paper, we propose a general framework for action-oriented process mining. This framework supports the continuous monitoring of operational processes and the automated execution of actions to improve the processes based on the monitoring results (i.e., diagnostics).

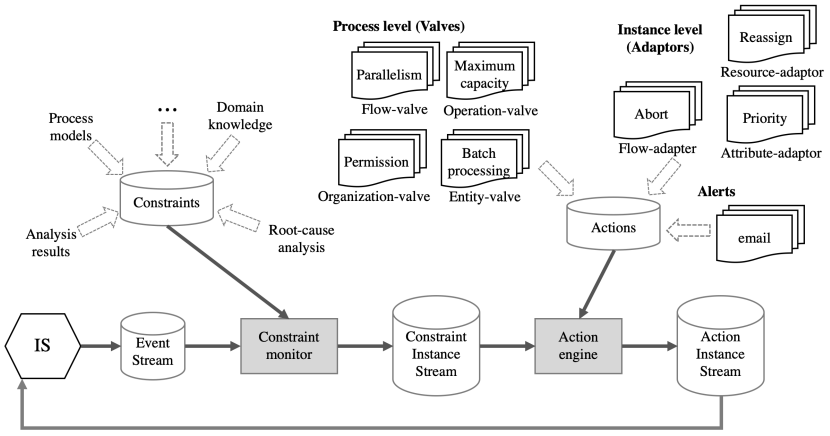


Fig. 1: Overview of the general framework for action-oriented process mining

Figure 1 shows an overview of the proposed framework. By analyzing the continuously updated event data (i.e., event stream), the *constraint monitor* evaluates a set of constraints that are defined with various diagnostics. As a result, it generates a constraint instance stream that is the description of monitoring results. By analyzing this constraint instance stream, the *action engine* assesses the necessity of actions and generates the actions ranging from process-level valves, instance-level adaptors, and alerts, as described in Fig. 1 with representative examples.

In order to advocate the effectiveness of the proposed framework on the continuous process management and the actual process improvement, it has been instantiated as a *ProM* plug-in. In addition, we have tested the implementation on an information system that supports a simulated order handling process. The

details of implementation and the information system are publicly available via `https://github.com/gyunamister/ActionOrientedProcessMining`.

The remainder is organized as follows. We first present a motivating example in Sect. 2. Next, we explain the preliminaries and the general framework for action-oriented process mining in Sect. 3 and Sect. 4. Afterward, Sect. 5 and Sect. 6 present the implementation of the framework and experiments as a proof of concept. Sect. 6 discusses the related work, and Sect. 7 concludes the paper.

2 Motivating example

Suppose we are operation managers in an e-commerce company like Amazon, responsible for an order handling process, where four main object types (i.e., *order, item, package,* and *route*) exist in the process as shown in Fig. 2a. Note that we do not assume a single case notion as in traditional process mining in the proposed framework. Instead, using the principles of object-centric process mining [5], we consider multiple object types and interacting processes. It is indispensable for acquiring precise diagnostics and deploying the framework at the enterprise level where multiple processes with different object types interact with each other.

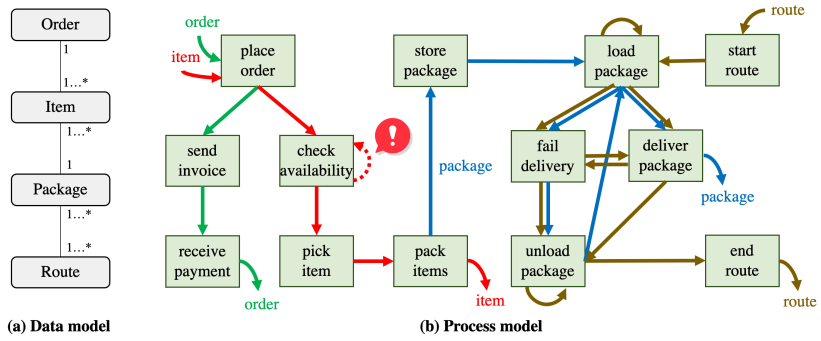


Fig. 2: Data model and the discovered process model of the order handling process. The discovered process model shows that *check availability* happens redundantly.

As operation managers, we analyze the event data using different process mining techniques. As an example, we discovered the process model shown in Fig. 2b where the arcs correspond to the specific object types. Interpreting the discovered process model, we observe that the activity, *check availability*, is redundantly repeated for some items, which should not happen according to a business rule and its negative effects on the overall operational performance. For the continuous management of this problem, we define a constraint *C1* as follows:

– *C1*: there must be no more than one "check availability" for each item.

Afterward, we put it into the repository of constraints and let the *constraint monitor* evaluate if any item violates or is predicted to violate the constraint every morning (e.g., at 9 AM).

We consider that it is highly risky to have more than 10 items violating or predicted to violate *C1* at any point in time, and in case this situation happens, it is most efficient to notify a case manager. Thus, we analyze the monitoring results every morning and take the following action to mitigate the risk:

– *A1*: If there exist more than 10 (possibly) violated items, send an e-mail to the case manager to warn for bad consequences.

This example shows how the insights from process discovery transform into mitigating actions (i.e., alerting a case manager). The proposed framework supports this process of insights turned into actions by continuously monitoring the violations and automatically generating proactive actions. In the following sections, we explain the major components of the framework, i.e., *constraint monitor* and *action engine*, using the above example as a running example.

3 Event data

In this section, we introduce the basic concepts, including event streams, time windows, and time moments.

Real-life processes often have multiple candidate identifiers, as shown in Sect. 2. To enable precise analysis and enterprise-wide adoption of the proposed framework, we use a more realistic event data notion where multiple case notions (e.g., order, item, etc) may coexist. Each event may refer to different objects from different object classes. Note that a conventional event log is a special case of this event data notion; hence one can use the proposed framework with the conventional event logs.

Definition 1 (Universes). *We define the following universes to be used in this paper:*

– $\mathcal{U}_{ei}$ *is the universe of event identifiers*
– $\mathcal{U}_{proc}$ *is the universe of process identifiers,*
– $\mathcal{U}_{act}$ *is the universe of activities,*
– $\mathcal{U}_{res}$ *is the universe of resources,*
– $\mathcal{U}_{time}$ *is the universe of timestamps,*
– $\mathcal{U}_{oc}$ *is the universe of object classes,*
– $\mathcal{U}_{oi}$ *is the universe of object identifiers,*
– $\mathcal{U}_{omap} = \mathcal{U}_{oc} \nrightarrow \mathcal{P}(\mathcal{U}_{oi})$ *is the universe of object mappings where, for* $omap \in \mathcal{U}_{omap}$*, we define* $omap(oc) = \emptyset$ *if* $oc \notin dom(omap)$*,*
– $\mathcal{U}_{attr}$ *be the universe of attribute names,*
– $\mathcal{U}_{val}$ *the universe of attribute values,*
– $\mathcal{U}_{vmap} = \mathcal{U}_{attr} \nrightarrow \mathcal{U}_{val}$ *is the universe of value mappings where, for* $vmap \in \mathcal{U}_{vmap}$*, we define* $vmap(attr) = \perp$ *if* $attr \notin dom(vmap)$*.*

– $\mathcal{U}_{event} = \mathcal{U}_{ei} \times \mathcal{U}_{proc} \times \mathcal{U}_{act} \times \mathcal{U}_{res} \times \mathcal{U}_{time} \times \mathcal{U}_{omap} \times \mathcal{U}_{vmap}$ *is the universe of events.*

We assume these universes are pairwise disjoint, e.g., $\mathcal{U}_{ei} \cap \mathcal{U}_{proc} = \emptyset$.

Each row in Table 1 shows an event of the order handling process introduced in Sect.2.

Event identifier	Process identifier	Activity name	Resource name	Timestamp	Objects involved				Attribute
					Order	Item	Package	Route	Type
...	...	...	...	...	...	...	...	...	...
746	*OH*	*place order*	*Jane*	01-01-2020 09:55	$\{o_7\}$	$\{i_8, i_9\}$	$\emptyset$	$\emptyset$	Gold
747	*OH*	*check availability*	*Jansen*	01-01-2020 10:15	$\{o_7\}$	$\{i_8\}$	$\emptyset$	$\emptyset$	
748	*OH*	*pick item*	*Kevin*	01-01-2020 11:55	$\{o_7\}$	$\{i_8\}$	$\emptyset$	$\emptyset$	
749	*OH*	*check availability*	*Matthias*	01-01-2020 17:55	$\{o_7\}$	$\{i_9\}$	$\emptyset$	$\emptyset$	
750	*OH*	*check availability*	*Jansen*	01-01-2020 19:05	$\{o_7\}$	$\{i_9\}$	$\emptyset$	$\emptyset$	
751	*OH*	*pick item*	*Kevin*	01-01-2020 19:55	$\{o_7\}$	$\{i_9\}$	$\emptyset$	$\emptyset$	
752	*OH*	*place order*	*System*	02-01-2020 09:15	$\{o_8\}$	$\{i_{10}\}$	$\emptyset$	$\emptyset$	Silver
753	*OH*	*pack items*	*Robin*	02-01-2020 15:05	$\{o_7\}$	$\{i_8, i_9\}$	$\{p_8\}$	$\emptyset$	
...	...	...	...	...	...	...	...	...	...

Table 1: A fragment of event data where each line corresponds to an event

Definition 2 (Event Projection). *Given an event* $e = (ei, proc, act, res, time, omap, vmap) \in \mathcal{U}_{event}$, $\pi_{ei}(e) = ei, \pi_{proc}(e) = proc, \pi_{act}(e) = act, \pi_{res}(e) = res, \pi_{time}(e) = time, \pi_{omap}(e) = omap$, *and* $\pi_{vmap}(e) = vmap$.

Let e_{746} be the first event depicted in Table 1. $\pi_{ei}(e_{746}) = 746$, $\pi_{proc}(e_{746}) = OH$, $\pi_{act}(e_{746}) = place\ order$, $\pi_{res}(e_{746}) = Jane$, $\pi_{time}(e_{746}) =$ 01-01-2020 09:55, $\pi_{omap}(e_{746})(Order) = \{o_7\}$, $\pi_{omap}(e_{746})(Item) = \{i_8, i_9\}$, and $\pi_{vmap}(e_{746})(Type) =$ Gold.

We adopt the notion of online event stream-based process mining, in which the data are assumed to be an infinite collection of unique events. An event stream is a collection of unique events that are ordered by time.

Definition 3 (Event Stream). *An event stream* S *is a (possibly infinite) set of events, i.e.,* $S \subseteq \mathcal{U}_{event}$ *such that* $\forall_{e_1, e_2 \in S}\ \pi_{ei}(e_1) = \pi_{ei}(e_2) \implies e_1 = e_2$. *We let* $\mathcal{U}_{stream}$ *denote the set of all possible event streams.*

A time window indicates the range of time to be analyzed.

Definition 4 (Time Window). *A time window* $tw = (t_s, t_e) \in \mathcal{U}_{time} \times \mathcal{U}_{time}$ *is a pair of timestamps such that* $t_s \leq t_e$. *Given a time window* $tw = (t_s, t_e)$, $\pi_{start}(tw) = t_s$ *and* $\pi_{end}(tw) = t_e$. $\mathcal{U}_{tw}$ *is the set of all possible time windows.*

A time moment represents the time when we start analyzing processes and the time window that the analysis addresses.

Definition 5 (Time Moment). *A time moment* $tm = (t, tw) \in \mathcal{U}_{time} \times \mathcal{U}_{tw}$ *is a pair of a timestamp* t *and a time window* tw *such that* $t \geq \pi_{end}(tw)$. *Given* $tm = (t, tw)$, *we indicate* $\pi_t(tm) = t$ *and* $\pi_{tw}(tm) = tw$. $\mathcal{U}_{tm}$ *is the set of all possible time moments.*

4 A General Framework for Action-oriented Process Mining

The proposed framework is mainly composed of two components. Firstly, the *constraint monitor* converts an event stream into a *constraint instance stream* where each *constraint instance* describes the (non) violation of a constraint. Second, the *action engine* transforms the constraint instance stream into an *action instance stream* where each *action instance* depicts a *transaction* to be executed by the information system to mitigate the risks caused by the violations.

4.1 Constraint Monitor

Each (non) violation of a constraint has a context where it occurs. For instance, *C1* in Sect. 2 could be violated by item i_9, which is a part of an order by a Gold customer in the process *OH*, when processed by *Joe* for the activity *check availability*.

Definition 6 (Context). *A context* $ctx \in \mathcal{P}(\mathcal{U}_{proc}) \times \mathcal{P}(\mathcal{U}_{act}) \times \mathcal{P}(\mathcal{U}_{res}) \times \mathcal{U}_{omap} \times \mathcal{U}_{vmap}$ *is a tuple of a set of process identifiers Proc, a set of activities Act, a set of resources Res, an object mapping omap, and a value mapping vmap.* $\mathcal{U}_{ctx}$ *is the set of all possible contexts.*

The above context is denoted as $ctx_1 = (\{OH\}, \{\textit{check availability}\}, \{Joe\}, omap_1, vmap_1)$, where $omap_1(Item) = \{i_9\}$ and $vmap_1(Type) = \text{Gold}$.

Given an event stream and a time window, the constraint formula evaluates if violations happen in a specific context by analyzing the events in the event stream, which are relevant to the time window.

Definition 7 (Constraint Formula). *We define* $\mathcal{U}_{outc} = \{OK, NOK\}$ *to be the universe of outcomes.* $cf \in (\mathcal{U}_{stream} \times \mathcal{U}_{tw}) \rightarrow \mathcal{P}(\mathcal{U}_{ctx} \times \mathcal{U}_{outc})$ *is a constraint formula.* $\mathcal{U}_{cf}$ *is the set of all possible constraint formulas.*

Suppose cf_1 is instantiated to evaluate the constraint described in *C1* of the motivating example. Given the event stream S that contains events listed in Table 1 and time window $tw_1 = (\text{01-01-2020 09:00}, \text{02-01-2020 09:00})$, it evaluates if any item in the time window experience more than one *check availability*. Since there are two *check availability* for item i_9, $(ctx_1, NOK) \in cf_1(S, tw_1)$.

In this paper, we do not assume specific approaches to instantiate the constraint formula. Several approaches are proposed in the field of process mining, including conformance checking techniques [6] and rule-driven approaches based on Petri-net patterns [7] and Linear Temporal Logic [8] (see also Sect.7)

A constraint consists of a constraint formula and a set of time moments, where the former explains what to monitor, and the latter specifies when to monitor.

Definition 8 (Constraint). *A constraint* $c = (cf, TM) \in \mathcal{U}_{cf} \times \mathcal{P}(\mathcal{U}_{tm})$ *is a pair of a constraint formula* cf *and a set of time moments* TM. $\mathcal{U}_c$ *is the set of all possible constraints.*

Suppose $c_1 = (cf_1, TM_1)$ where $(\text{02-01-2020 09:00}, (\text{01-01-2020 09:00}, \text{02-01-2020 09:00})) \in TM_1$. For instance, we evaluate cf_1 at 02-01-2020 09:00 with the events related to time window (01-01-2020 09:00,02-01-2020 09:00).

A constraint instance specifies when and whether a violation happens in a certain context by a constraint formula.

Definition 9 (Constraint Instance). *A constraint instance $ci \in \mathcal{U}_{cf} \times \mathcal{U}_{ctx} \times \mathcal{U}_{time} \times \mathcal{U}_{outc}$ is a tuple of a constraint formula cf, a context ctx, a timestamp $time$, and an outcome $outc$. $\mathcal{U}_{ci}$ is the set of all possible constraint instances.*

For instance, a constraint instance $ci_1 = (cf_1, ctx_1, \text{02-01-2020 09:00}, NOK)$ denotes that cf_1 is violated at 02-01-2020 09:00. in context ctx_1.

A constraint instance stream is a collection of unique constraint instances.

Definition 10 (Constraint Instance Stream). *A constraint instance stream CIS is a (possibly infinite) set of constraint instances, i.e., $CIS \subseteq \mathcal{U}_{ci}$. $\mathcal{U}_{CIS}$ is the set of all possible constraint instance streams.*

Given an event stream, a constraint monitor evaluates a set of constraints and generates a constraint instance stream.

Definition 11 (Constraint Monitor). *Let $C \subseteq \mathcal{U}_c$ be a set of constraints to be used for monitoring. $cm_C \in \mathcal{U}_{stream} \rightarrow \mathcal{U}_{CIS}$ is the constraint monitor such that, for any $S \in \mathcal{U}_{stream}$, $cm_C(S) = \{(cf, ctx, time, outc) \in \mathcal{U}_{ci} | \exists_{TM,tm}\ (cf, TM) \in C \wedge tm \in TM \wedge time = \pi_t(tm) \wedge (ctx, outc) \in cf(S, \pi_{tw}(tm))\}$.*

Note that the definition of a constraint monitor is abstracted in a way that we are able to analyze future events. In reality, it analyzes only the historical events from an event stream and outputs the constraint instance stream relevant to them.

4.2 Action Engine

The action engine aims at producing an action instance stream describing transactions that source information systems need to execute to mitigate the risk incurred by the constraint violations.

Definition 12 (Transaction). *Let $\mathcal{U}_{op}$ be the universe of operations that are executed by information systems (e.g., send emails). A transaction $tr = (op, vmap) \in \mathcal{U}_{op} \times \mathcal{U}_{vmap}$ is a pair of an operation op and a parameter mapping $vmap$. $\mathcal{U}_{tr} \subseteq \mathcal{U}_{op} \times \mathcal{U}_{vmap}$ denotes the set of all possible transactions.*

For instance, the action description *A1* in Sect. 2 represents a transaction, tr_1*=(send-an-email,vmap′)* where *vmap′(recipient)="case manager"* and *vmap′(message) = "Frequent violations of C1"*.

Given a constraint instance stream and a time window, the action formula produces required transactions by analyzing the constraint instances in the constraint instance stream, which are relevant to the time window.

Definition 13 (Action Formula). *An action formula* $af \in (\mathcal{U}_{CIS} \times \mathcal{U}_{tw}) \rightarrow \mathcal{P}(\mathcal{U}_{tr})$ *is a function that maps a constraint instance stream and time window to a set of transactions.* $\mathcal{U}_{af}$ *is the set of all possible action formulas.*

Assume af_1 to assess the condition that is specified by the action description *A1* in Sect. 2, and to produce the corresponding transaction. Given constraint instance stream CIS and time window $tw_1 = (\text{01-01-2020 09:00}, \text{02-01-2020 09:00})$, it assesses if there exist more than 10 constraint instances whose outcomes are "NOK" in the time window. If so, $tr_1 = (\textit{send-an-email}, vmap') \in af(CIS, tw_1)$.

An action consists of an action formula and a set of time moments. The action formula specifies which transactions to generate in which conditions, and the set of time moments indicates when to assess the conditions and to generate transactions.

Definition 14 (Action). *An action* $a = (af, TM) \in \mathcal{U}_{af} \times \mathcal{P}(\mathcal{U}_{tm})$ *is a pair of an action formula* af *and a set of time moments* TM. $\mathcal{U}_a$ *denotes the set of all possible actions.*

Suppose $a_1 = (af_1, TM_1)$ where $(\text{02-01-2020 09:00}, (\text{01-01-2020 09:00}, \text{02-01-2020 09:00})) \in TM_1$. We implement af_1 at 02-01-2020 09:00 with the constraint instances related to time window (01-01-2020 09:00,02-01-2020 09:00).

An action instance indicates when and which transaction is required.

Definition 15 (Action Instance). *An action instance* $ai = (af, tr, time) \in \mathcal{U}_{af} \times \mathcal{U}_{tr} \times \mathcal{U}_{time}$ *is a tuple of an action formula* af*, a transaction* tr*, and a timestamp* $time$. $\mathcal{U}_{ai}$ *is the set of all possible action instances.*

For instance, an action instance $ai_1 = (af_1, tr_1, \text{02-01-2020 09:00})$ denotes that the transaction tr_1 needs to be executed at 02-01-2020 09:00 according to af_1.

An action instance stream is a collection of unique action instances.

Definition 16 (Action Instance Stream). *An action instance stream* AIS *is a (possibly infinite) set of action instances, i.e.,* $AIS \subseteq \mathcal{U}_{ai}$. $\mathcal{U}_{AIS}$ *is the set of all possible action instance streams.*

Given a constraint instance stream, an action engine continuously assesses the necessity of transactions by analyzing the action formulas in predefined actions at their appointed times.

Definition 17 (Action Engine). *Let* $A \subseteq \mathcal{U}_a$ *be a set of actions used by the action engine.* $ae_A \in \mathcal{U}_{CIS} \rightarrow \mathcal{U}_{AIS}$ *is the action engine such that, for any* $CIS \in \mathcal{U}_{CIS}$*,* $ae_A(CIS) = \{(af, tr, time) \in \mathcal{U}_{ai} | \exists_{TM,tm} (af, TM) \in A \wedge tm \in TM \wedge time = \pi_t(tm) \wedge tr \in af(CIS, \pi_{tw}(tm))\}$.

We abstract that the action engine is able to assess future constraint instances. In fact, it analyzes the historical constraint instance stream and produces transactions which mitigate risks caused by the past constraint violations.

5 Implementation

The general framework discussed above is implemented as a plug-in of *ProM*[1], an open-source framework for the implementation of process mining tools in a standardized environment. Our new plug-in is available in a new package named *ActionOrientedProcessMining*. The main input objects of our plug-in are an event stream, a constraint formula definition, and an action formula definition, whereas the output is an action instance stream.

The input event stream is in an XML-based *Object-Centric Log (OCL)* format storing events along with their related objects, while the constraint formula and action formula are defined by *Constraint Formula Language (CFL)* and *Action Formula Language (AFL)*, respectively. The schema of *OCL* and the syntax of *CFL* and *AFL* are explained in the tool manual[2] with examples.

The output action instance stream is in an XML-based *Action Instance Stream (AIS)* format[2] storing action instances describing the transactions that need to be applied by source information systems. A dedicated gateway implemented in the source system parses the resulting *AIS* file and translates it into the system-readable transactions.

6 Proof-of-concept

Based on the implementation, we conducted experiments with the artificial information system supporting a simulated process to evaluate the feasibility of the framework. Specifically, we are interested in answering the following research questions.

- RQ1: Does the constraint monitor effectively detect violations?
- RQ2: Does the action engine effectively generate corresponding transactions?
- RQ3: Does the application of the transactions improve operational processes?

Experimental design The information system used for the evaluation supports the order handling process described in Sect. 2. There are 16 available resources in total at any point in time, and each of them is responsible for multiple activities in the process. Orders are randomly placed and queued for the resource allocation after each activity. The resource is allocated according to the *First-in First-out* rule.

To answer RQ1-RQ3, we carried out the following steps repeatedly:

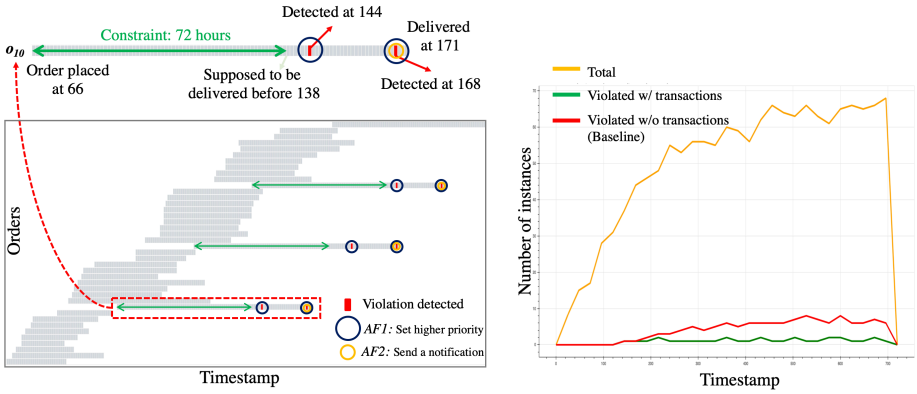
1. The information system generates events and updates an event stream.
2. The constraint monitor evaluates the constraint formula, which formulates "An order must be delivered in 72 hours", every 24 hours.
3. The action engine assesses the following action formulas every 24 hours:
   - "If there is a violated order in the last 24 hours, set a higher priority for it" (*AF1*), and

[1] `http://www.promtools.org`

[2] `https://github.com/gyunamister/ActionOrientedProcessMining`

– "If there is a violation that lasted longer than 24 hours, send a notification to a case manager". (*AF2*)

4. The dedicated gateway for the information system translates the action instance stream into transactions that are executed by the information system.



(a) Results of constraint monitor and action engine **(b) Number of violated instances**

Fig. 3: Experimental results: (a) The results of constraint monitor and action engine on 40 selected orders. (b) Number of violated instances for 30 days.

Experimental Results Fig. 3a reports the results related to RQ1 and RQ2. The figure shows the history of 40 orders by time, where the gray box indicates the delivery time (i.e. from order placement to delivery) and the green arrow denotes allowable delivery time (i.e., 72 hours). The red box represents the time when the violation happens. As shown in Fig. 3a, every order whose delivery time is outside the green arrow is detected by the constraint monitor every 24 hours. Moreover, *AF1* is generated for every violation and *AF2* is generated if the violation lasts longer.

Fig. 3b reports an experimental result related to RQ3. The figure shows the number of violated instances for 30 days. The yellow line indicates the total number of instances, while the red/green lines represent the number of violated instances with/without applying mitigating transactions. The number of violated instances decreases when the transactions are applied.

7 Related Work

A constraint formula is a core component of the proposed framework, enabling the constraint monitor. The process mining discipline provides many techniques to be deployed to instantiate it. Conformance checking [6] can be deployed to find discrepancies between the modeled and the observed behavior of running instances. More tailor-made rules can be formalized into Petri-net patterns [7] or Linear Temporal Logic (LTL) [8] to evaluate whether process executions comply with them.

In addition, deviation detection techniques detect deviations in process executions with the user-defined compliance rules not being given. Instead, the model-based [9] and clustering-based [10] approaches learn the rules by analyzing event data and evaluate the violation of them in the execution of processes.

Furthermore, a constraint formula can be extended by more forward-looking techniques that are able to predict what will happen to individual cases and where bottlenecks are likely to develop [3, 4]. The resulting predictions can be incorporated into the compliance rules [7, 8] to evaluate future violations.

A commercial process mining tool, *Celonis Action Engine* [11], is a representative effort to turn analysis results into actions. It generates signals by analyzing the event data, and executes the actions corresponding to these signals to the source system. However, it does not support processing streaming data, which limits the continuous process management, analyzing signals (i.e., monitoring results), which inhibits the generation of relevant actions, and executing actions at process levels.

Several methods have been developed to generate proactive actions from the process mining diagnostics. In [12], the resource allocation is proactively optimized with the risk predictions of running instances. A prescriptive alarm system [13] generates alarms for the process instances that are predicted to be problematic with the aid of a cost model to capture the trade-off between different interventions. These approaches focus on improving the process by dealing with specific problems, mostly at the instance level. Instead, our proposed framework supports the management of comprehensive process-related problems and the execution of actions at both instance and process levels.

Robotic Process Automation (RPA) also aims at improving operational processes by automating repetitive tasks performed by humans by mimicking the execution on the user interface (UI) level [14]. While having the shared goal, the proposed framework has more emphasis on effectively managing the process in a continuous manner by identifying the problems based on diagnostics and executing proactive actions. Automating the problematic part of process executions with RPA techniques is one of those effective management actions.

8 Conclusion

In this paper, we proposed the general framework for action-oriented process mining, which continuously transforms process diagnostics into proactive actions for the process improvement. It is mainly composed of two parts: the constraint monitor and the action engine. The constraint monitor supports continuous monitoring of constraints, and the action engine generates the necessary transactions to mitigate the risks caused by the constraint violations.

The framework is instantiated in a *ProM* plug-in and tested on an information system. In fact, this paper is the starting point for a new branch of research in process mining. As future works, we plan to develop a concrete technique to support the efficient analysis of constraint instance streams by incorporating the concept of data cube. We also plan to validate the effectiveness of the

proposed framework in real-life processes supported by information systems like SAP, Salesforce, Microsoft Dynamics, etc. Another important direction of future work is to provide a comprehensive taxonomy of constraints and actions to support the elicitation of relevant constraints and actions.

Acknowledgements We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

References

1. van der Aalst, W.M.P.: Data Science in Action. In: Process Mining. Springer, Heidelberg (2016)
2. Reinkemeyer, L., ed.: Process Mining in Action: Principles, Use Cases and Outlook. Springer International Publishing, Cham (2020)
3. de Leoni, M., van der Aalst, W.M.P., Dees, M.: A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Information Systems **56** (2016) 235–257
4. Marquez-Chamorro, A.E., Resinas, M., Ruiz-Cortes, A.: Predictive Monitoring of Business Processes: A Survey. IEEE Transactions on Services Computing **11**(6) (2018) 962–977
5. van der Aalst, W.M.P.: Object-Centric Process Mining: Dealing with Divergence and Convergence in Event Data. In Ölveczky, P.C., Salaün, G., eds.: Software Engineering and Formal Methods. Volume 11724. Springer International Publishing, Cham (2019) 3–25
6. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer (2018)
7. Ramezani, E., Fahland, D., van der Aalst, W.M.P.: Where Did I Misbehave? Diagnostic Information in Compliance Checking. In Hutchison, D., et al., eds.: Business Process Management. Volume 7481. Springer, Heidelberg (2012) 262–278
8. van der Aalst, W.M.P., de Beer, H.T., van Dongen, B.F.: Process Mining and Verification of Properties: An Approach Based on Temporal Logic. In Hutchison, D., Kanade, T., et al., eds.: OTM 2005: CoopIS, DOA, and ODBASE. Volume 3760. Springer, Heidelberg (2005) 130–147
9. Bezerra, F., Wainer, J.: Algorithms for anomaly detection of traces in logs of process aware information systems. Information Systems **38**(1) (2013) 33–44
10. Ghionna, L., Greco, G., Guzzo, A., Pontieri, L.: Outlier Detection Techniques for Process Mining Applications. In An, A., Matwin, S., et al., eds.: Foundations of Intelligent Systems. Volume 4994. Springer, Heidelberg (2008) 150–159
11. Badakhshan, P., Bernhart, G., Geyer-Klingeberg, J., Nakladal, J., Schenk, S., Vogelgesang, T.: The Action Engine – Turning Process Insights into Action. In: 2019 ICPM Demo Track, Aachen, Germany, ceur-ws (2019) 28–31
12. Conforti, R., de Leoni, M., La Rosa, M., van der Aalst, W.M.P., ter Hofstede, A.H.: A recommendation system for predicting risks across multiple business process instances. Decision Support Systems **69** (2015) 1–19
13. Fahrenkrog-Petersen, S.A., Tax, N., Teinemaa, I., Dumas, M., de Leoni, M., Maggi, F.M., Weidlich, M.: Fire Now, Fire Later: Alarm-Based Systems for Prescriptive Process Monitoring. arXiv:1905.09568 [cs, stat] (2019) arXiv: 1905.09568.
14. Agostinelli, S., Marrella, A., Mecella, M.: Towards Intelligent Robotic Process Automation for BPMers. arXiv:2001.00804 [cs] (2020) arXiv: 2001.00804.