Progress in Artificial Intelligence
https://doi.org/10.1007/s13748-022-00281-7

REGULAR PAPER

Action-oriented process mining: bridging the gap between insights and actions

Gyunam Park[1] · **Wil M. P. van der Aalst**[1]

Received: 13 March 2021 / Accepted: 25 March 2022

Abstract
As business environments become more dynamic and complex, it becomes indispensable for organizations to objectively analyze business processes, monitor the existing and potential operational frictions, and take proactive actions to mitigate risks and improve performances. Process mining provides techniques to extract insightful knowledge of business processes from event data collected during the execution of the processes. Besides, various approaches have been suggested to support the real-time (predictive) monitoring of the process-related problems. However, the link between the insights from the continuous monitoring and the concrete management actions for the actual process improvement is missing. *Action-oriented process mining* aims at connecting the knowledge extracted from event data to actions. In this work, we propose a general framework for action-oriented process mining covering the continuous monitoring of operational processes and the automated execution of management actions. Based on the framework, we suggest a cube-based action engine where actions are generated by analyzing monitoring results in a multi-dimensional way. The framework is implemented as a *ProM* plug-in and evaluated by conducting experiments on both artificial and real-life information systems.

Keywords Action-oriented process mining · Continuous operational management · Turning events into actions · Continuous process improvement

1 Introduction

In [1], the term "business process hygiene" was coined. In the same manner that individuals do regular check-ups to find potential health issues before they become serious problems, organizations should objectively analyze key processes to identify existing and potential problems and improve performance and productivity.

Indeed, many efforts have been made to ensure the overall health of organizations by redesigning business processes [2]. Process redesign often entails a comprehensive and extensive analysis of business processes and requires fundamental changes to the process. Despite the efforts to prevent inefficiencies in design-time, many operational frictions still arise in the execution of the business process, making a variety of exceptions. For instance, an order-to-cash process, which is standardized with known best practices, often shows thousands of variants in the real-life execution, generating various problems in organizations.

In order to deal with the unanticipated operational frictions that may arise during the execution of business processes, it is imperative to manage business processes in a continuous manner. To this end, business managers need to continuously identify problems, monitor the occurrence of the problems, and take proactive actions to deal with possible risks to the business process. This continuous management of business processes enables to deal with relevant operational frictions that may happen in the dynamically changing environments in a responsive and proactive manner.

Process mining has provided many useful techniques to support the continuous management of business processes. First, process discovery, conformance checking, and process enhancement have enabled the business managers to identify problems by making the business processes transparent [3]. Moreover, process monitoring techniques have effectively detected and predicted problems in an online manner [4–6].

✉ Gyunam Park
gnpark@pads.rwth-aachen.de

Wil M. P. van der Aalst
wvdaalst@pads.rwth-aachen.de

[1] Department of Computer Science, Process and Data Science Group (PADS), RWTH Aachen University, Aachen, Germany

Published online: 02 July 2022

However, the selection of actions to address such problems is still unstructured and ad-hoc, i.e., the "action part" is still missing and outside the scope of today's process mining tools. Indeed, for the actual process improvement, it is necessary to turn the insights from process mining diagnostics to management actions. For instance, when a bottleneck emerges or is forecasted to arise, business managers should take actions, such as alerting responsible employees, bypassing the activity, and assigning more resources, alongside the detection and prediction of it.

Action-oriented process mining aims at addressing such problems by systematically combining process mining results and domain knowledge, and also automating management actions to improve business processes. Figure 1 presents the overview of the action-oriented process mining. Process mining techniques for diagnostics (cf. Sect. 2.1) are used to extract process knowledge from event data. The *constraint monitor* analyzes a continuous stream of event data and evaluates a set of constraints designed using the process knowledge combined with domain knowledge. As a result, it generates constraint instances describing the monitoring results. Note that events in event data have temporal relationships and constraints over the event data often entail temporal nature. Thus, they cannot be monitored using techniques for *automated data quality verification* [7] with non-temporal declarative constraints. Next, the *action engine* analyzes the constraint instances and produces action instances describing needed actions to deal with the existing and potential threats to the business processes. The action instances are automatically triggered in the underlying information system to make changes in system configurations, or generate alerts through its messaging systems.

In this paper, we provide following contributions:

- We propose a general framework for action-oriented process mining to support the continuous monitoring of operational processes and the automated execution of actions by extending our earlier work presented in [8].
- We instantiate the action engine of the framework using *constraint cubes* that stores the continuous stream of constraint instances and analyzes it to produce relevant actions.
- We have implemented the cube-based action engine as a ProM plug-in.
- We have evaluated the effectiveness of the framework based both on artificial and real-life information systems.

The remainder is organized as follows. We present the related work in Sect. 2. Next, we explain the background covering a motivating example, basic notation, and event data in Sect. 3. Afterward, we present the general framework for action-oriented process mining and the constraint cube-based instantiation of the action engine in Sects. 4 and 5. Section 6 presents the implementation of the framework and experiments both in artificial and real-life information systems. In Sect. 8, we discuss the implications and limitations and conclude the paper.

2 Related work

In this section, we first introduce process mining techniques to provide diagnostics used for the identification of improvement points. Afterward, we present techniques for the operational support, showing the missing gap between insights from the detection and prediction of problems and actual actions to improve business processes. Finally, we demonstrate the need for a systematic approach to support continuous process improvement.

2.1 Process mining diagnostics

Action-oriented process mining begins by defining improvement points in business processes based on process knowledge. Process mining provides techniques to extract process-centric insights from event data collected by information systems during the execution of business processes.

The three main categories of process mining diagnostics include process discovery, conformance checking, and process enhancement. First, process discovery is to automatically derive a process model from the event log recorded from the execution of business processes [9]. The resulting process model captures the control-flow relations between activities observed in the event log.

Second, conformance checking evaluates to what degree the execution of a process conforms with the reference process model or the discovered process model [10]. It allows business analysts to identify non-conforming process instances and analyze their behaviors that lead to the non-conformity.

Third, process enhancement is to enrich the process model with additional information to enable performance analysis [3]. By using timestamps, one can extend it with time-related measures (e.g., service time, throughput time, and waiting time) and analyze the bottlenecks in business processes. For instance, a performance spectrum plots each process step per case over time, allowing to analyze non-stationarity of performance and synchronization of different cases over time [11]. Focusing on the organizational perspective of business processes, social network analysis provides insights into the relationship among such as handover of works, subcontracting, and working together [3]. In addition, organizational mining analyzes the roles in organizations that execute a similar set of activities [12]. Furthermore, root cause analysis enables to find in-depth explanations of risk incidents.

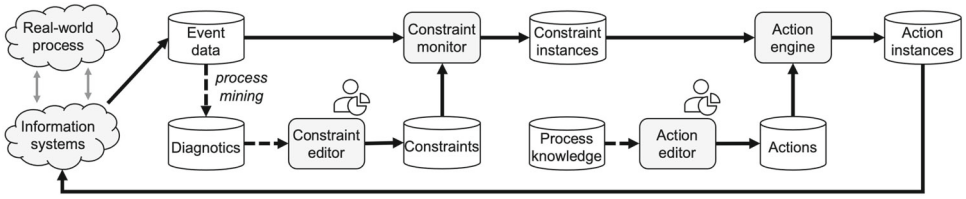

Fig. 1 The objective of action-oriented process mining: continuous process improvement

In [13], the root-cause of long throughput times of process instances is analyzed by focusing on the effect of workloads.

2.2 Operational support

Instead of providing process-centric insights by analyzing historical data, operational support aims at influencing current operational processes to improve the process by analyzing current event data. In [3], three core activities of operational support are described, i.e., *detect*, *predict*, and *recommend*. First, *detect* activity analyzes deviating behaviors by running process instances. Conformance checking is the enabling technology for this activity. For monitoring purposes, the conformance checking techniques are lifted to runtime. For instance, van Zelst et al. [4] propose to compute prefix-alignments to detect deviant behaviors. Burattin et al. [14] suggest a generic framework to compute conformance indicators based on behavioral patterns.

Some approaches exploit Complex Event Processing (CEP) with the abstractions of models such as direct or eventual successions. Weidlich et al. [15] propose a method to derive event queries from behavioral profiles that serve as abstractions of the process model. The event queries are monitored using the CEP engine. Awad et al. [16] suggest a technique to derive anti-patterns based on a predefined generic set of patterns regarding business processes. The patterns describe the rules in terms of the occurrence of tasks, their ordering, and resource assignments. A CEP engine monitors them and detects violations of the rules.

In order to verify properties at runtime, Linear Temporal Logic (LTL) is deployed as a declarative language for describing the properties [17]. For monitoring purposes, more possible truth-value states are defined such as temporarily satisfied, temporarily violated, permanently satisfied, or permanently violated. Maggi et al. [5] suggest a monitoring technique based on LTL and colored automata. The global automaton represents the conjunction of all the imposed constraints represented by the local automata. It enables to identify the possible conflicts among different constraints.

In several approaches, rules are represented as graphical notations such as Petri net. In [6], the constraints are formalized into Petri-net patterns. The patterns are aligned with event logs to evaluate whether the execution of business processes comply with them. In [18], the Petri-net patterns for cloud-based business processes are suggested for certifying compliant cloud-based processes.

Some approaches analyze deviating behaviors of process instances without the user-defined compliance rules given. Bezerra et al. [19] propose a technique to detect the structural deviations in business processes by discovering infrequent process instances in the process. Ghionna et al. [20] compute clusters of process instances using the pattern measures and find deviations by identifying clusters with small sizes. Replacing the time-consuming process of finding clusters, Li et al. [21] suggest a framework to identify deviating process instances based on the profiles that encode the characteristics of normal process instances.

Second, *predict* activity aims to provide timely information to mitigate risks and improve business processes by predicting what will happen to individual cases and where bottlenecks are likely to develop [22]. Polato et al. [23] predict the remaining time of running process instances using support vector regression methods. Senderovich et al. [24] extracts intra- and inter-case features to consider dynamics among different process instances and predict the remaining time using linear regression, random forests, and XGBoost approaches. di Francescomarino et al. [25] suggest a clustering-based approach to predict outcomes of running process instances by mapping them into clusters and estimate the probabilities of possible outcomes. Teinemaa et al. [26] present a predictive process monitoring framework combining text mining with classification techniques to deal with both structured and unstructured data.

In addition, next event prediction is also actively studied. In [27], Hidden Markov Models (HMM) are used to predict the next process steps. Lakshmanan et al. [28] build a decision tree on each activity in the process model to compute the transition probabilities and use a Markov chain model to predict the next step in business processes. Breuker et al. [29] utilize a Probabilistic Finite Automaton (PFA) based on Bayesian regularization, providing the comprehensibility of the predictive model. In [30], a rule-based approach is proposed to predict the next events, which encodes event log properties using a window-based encoding technique.

Recent breakthroughs in deep neural networks have enabled the extensive development of predictive business process monitoring techniques [31]. For instance, Long Short-Term Memory networks (LSTMs) have been adopted to predict the next event, remaining time, and outcome of

process instances in business processes [32]. In [33], the occurrences of activities and their timestamps are encoded using one-hot encoding, and LSTMs are applied to predict the next event and timestamp. Mehdiyev et al. [34] extends the previous deep learning approaches by providing an architecture composed of unsupervised stacked autoencoders and supervised fine-tuning with n-gram features leveraged by feature hashing.

Moreover, several approaches are proposed to explain why a predictive model reports the predictions, facilitating the adoption of the model in practice. For instance, Galanti et al. provide explanations of the predictions using the game theory of Shapley Values [35].

Third, *recommend* activity provides the guidance to achieve the goal of business processes (e.g., minimizing flow time and resource usage). In [36], the resource allocation is proactively optimized with the risk predictions of running instances. A prescriptive alarm system [37] 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. Weinzierl et al. [38] suggest a method to recommend the next best actions by analyzing the next most likely activities in terms of key performance indicators.

Compared to the extensive literature that supports *detect* and *predict* activities, little attention has been paid on *recommend* activity. The decisions for the corrective actions to improve business processes are left to the subjective judgment of process participants. However, this subjective judgment rather than objective facts result in undesired outcomes. Dees et al. [39] show that interventions may lead to unanticipated results, demonstrating the importance of making effective decisions on interventions along with accurate detection and prediction. Moreover, de Leoni et al. [40] reports the effectiveness of purely objective, unbiased recommendations in improving key performance indicators. This triggers the need for a more systematic approach that supports the decision on effective actions and automates interventions by quickly changing, evaluating, and experimenting with the interventions.

2.3 Action-oriented process mining

A commercial process mining tool, *Celonis Action Engine* [41], is a representative effort to achieve the goal of action-oriented process mining, i.e., turning diagnostics 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 provide a systematic approach to transform the process-centric diagnostics into needed management actions, rather generating actions in an ad-hoc manner. In this work, we provide the systematic approach to convert the insights from process diagnostics to automated corrective actions by presenting a general framework for the action-oriented process mining and its realization using constraint cubes.

Recently, a digital twin interface model has been proposed to realize digital twins of an organization using action-oriented process mining [42]. The interface model, on the one hand, represents the current state of a business process including currently processed objects and diagnostics (e.g., bottlenecks) of the process. On the other hand, it describes possible configurations of the process, called *valves*. Using the interface model, a process expert can design constraints by analyzing the current state of the underlying process and define actions to manage the violation of the constraints. Based on the constraints and actions, the process is continuously monitored and necessary actions are automatically executed to the underlying information system.

3 Background

In this section, we introduce a motivating example that is used as a running example to explain the major components of the framework for action-oriented process mining. Also, we present basic preliminary material covering the notion of the time window, time moment, and event stream.

3.1 Motivating example

Suppose we are operation managers in an e-commerce company like Amazon. In the order handling process of the company, four main object types (i.e., *order, item, package,* and *route*) exist as shown in Fig. 2a.

In this work, we do not assume a single case notion as in traditional process mining. Instead, using the principles of object-centric process mining [43], 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.

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 with different colors correspond to different object types. For instance, the green arc represents the order that includes activities such as place order, send invoice, and receive payment. After placing an order, the relevant items of the order undergo check availability, pick item, and pack items, creating packages of the items.

Based on the discovered process model, we observe that *fail delivery* occurs redundantly, decreasing customer satisfaction and increasing the cost for deliveries. For the continuous management of this problem, we define a constraint *C1* as follows:

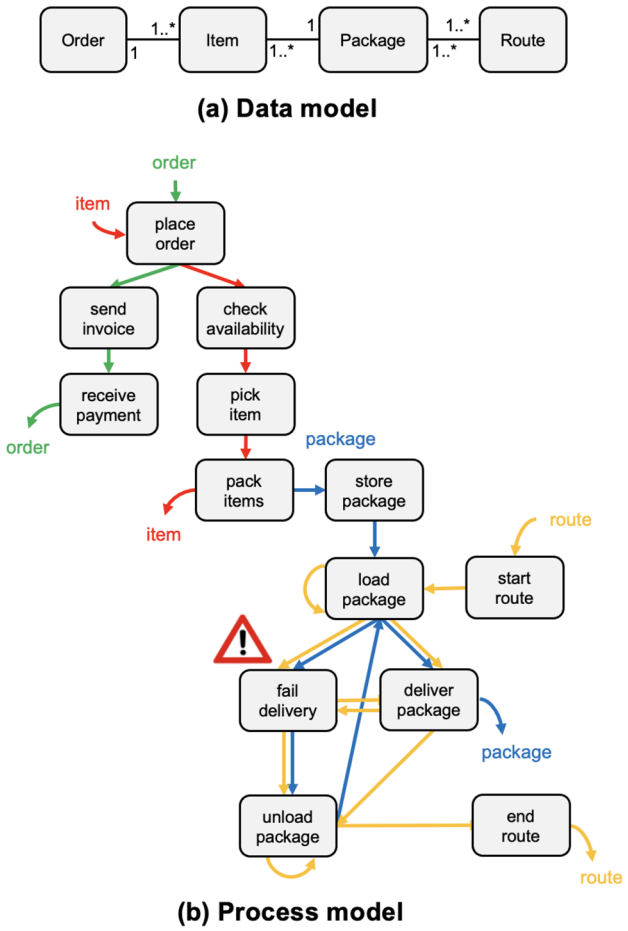

Fig. 2 Data model and the discovered process model of the order handling process. The discovered process model describes the occurrence of *fail delivery*

– *C1*: there must be no "fail delivery" for any package.

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 a.m.).

We consider that it is highly problematic that the same package fails to be delivered more than twice, and in case this situation happens, it is most efficient to ask customers to provide alternative methods to ensure the successful delivery. Thus, we analyze the monitoring results every morning and take the following action to mitigate the risk:

– *A1*: if a package is failed to be delivered more than twice, send a message to the customer to ask for alternative methods to delivery (e.g., deliver to a neighbor and select a pick-up store).

This example shows how the insights from process discovery (i.e., the occurrence of *fail delivery*) transform into mitigating actions (i.e., finding an alternative method for deliveries). The proposed framework supports this process of insights turned into actions by continuously monitoring the violations and automatically generating proactive actions.

3.2 Basic notation

Let X denote an arbitrary set. $\mathcal{P}(X)$ denotes the power set of X, i.e., $\mathcal{P}(X)=\{X' | X' \subseteq X\}$. We let $\mathbb{B}=\{true, false\}$ denote the set of Boolean values.

A directed acyclic graph is a graph with directed edges in which there are no cycles, i.e., $dag=(N, R)$ where N is a set of nodes N and $R \subseteq N \times N$ is a binary relation on N that specifies a directed edge from a node $n \in N$ to another one $m \in N$ if $(n, m) \in R$. The transitive closure R^+ of the relation R is irreflexive, i.e., $(n, n) \notin R^+$ for any $n \in N$.

Let $\mathbb{U}_{time}$ be the universe of timestamps. A time window $tw=(t_s, t_e) \in \mathbb{U}_{time} \times \mathbb{U}_{time}$ 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 $\pi_{end}(tw)=t_e$. $\mathbb{U}_{tw}$ denotes the set of all possible time windows.

A time moment $tm=(t, tw) \in \mathbb{U}_{time} \times \mathbb{U}_{tw}$ 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 $\pi_{tw}(tm)=tw$. Suppose time moment $tm_1=$(03-01-2022 09:00, (03-01-2022 09:00, 03-01-2022 09:00)). In monitoring purposes, it indicates to monitor events in time window (03-01-2022 09:00,03-01-2022 09:00) at 03-01-2022 09:00. We let $\mathbb{U}_{tm}$ denote the set of all possible time moments.

3.3 Event data

Real-life processes often have multiple candidate identifiers, as shown in Sect. 3.1. 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.

Recently, the *OCEL standard*[1] has been proposed to support such event data notion. In the following, we formulate a conceptual abstraction of it and use the abstraction to formally define the proposed framework in Sect. 4.

Definition 3.1 (*Universes*) We define the following universes to be used in this paper:

– $\mathbb{U}_{ei}$ is the universe of event identifiers
– $\mathbb{U}_{proc}$ is the universe of process identifiers,

[1] http://ocel-standard.org.

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

Event identifier	Process identifier	Activity name	Resource name	Timestamp	Objects involved				Attribute
					Order	Item	Package	Route	Type
...	...	...	...	...	...	...	...	...	...
199210	*OH*	*Place order*	*Merlin*	02-01-2022 09:55	$\{o_7\}$	$\{i_8, i_9\}$	$\emptyset$	$\emptyset$	Silver
199211	*OH*	*Check availability*	*Laure*	02-01-2022 10:15	$\{o_7\}$	$\{i_8\}$	$\emptyset$	$\emptyset$	
199212	*OH*	*Pick item*	*Bowie*	02-01-2022 11:55	$\{o_7\}$	$\{i_8\}$	$\emptyset$	$\emptyset$	
199213	*OH*	*Load package*	*Adams*	02-01-2022 17:55	$\emptyset$	$\{i_3, i_4\}$	$\{p_2\}$	$\emptyset$	
199214	*OH*	*Fail delivery*	*Adams*	02-01-2022 17:55	$\emptyset$	$\{i_3, i_4\}$	$\{p_2\}$	$\emptyset$	
199215	*OH*	*Unload package*	*Adams*	02-01-2022 17:55	$\emptyset$	$\{i_3, i_4\}$	$\{p_2\}$	$\emptyset$	
199216	*OH*	*Load package*	*Schuster*	02-01-2022 21:55	$\emptyset$	$\{i_5, i_6\}$	$\{p_3\}$	$\emptyset$	
199217	*OH*	*Place order*	*System*	03-01-2022 09:15	$\{o_8\}$	$\{i_{10}\}$	$\emptyset$	$\emptyset$	Gold
199218	*OH*	*Pack items*	*James*	03-01-2022 15:05	$\{o_7\}$	$\{i_8, i_9\}$	$\{p_8\}$	$\emptyset$	
...	...	...	...	...	...	...	...	...	...

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

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

Each row in Table 1 shows an event of the order handling process introduced in Sect. 3.1. Given an event $e=(ei, proc, act, res, time, omap, vmap) \in \mathbb{U}_{event}$, $\pi_{ei}(e)=ei$, $\pi_{proc}(e)=proc$, $\pi_{act}(e)=act$, $\pi_{res}(e)=res$, $\pi_{time}(e)=time$, $\pi_{omap}(e)=omap$, and $\pi_{vmap}(e)=vmap$. Let e_{199214} be the first event depicted in Table 1. $\pi_{ei}(e_{199214})=199214$, $\pi_{proc}(e_{199214})=OH$, $\pi_{act}(e_{199214})=$*fail delivery*, $\pi_{res}(e_{199214})=Adams$, $\pi_{time}(e_{199214})=$02-01-2022 17:55, $\pi_{omap}(e_{199214})(Item)=\{i_3, i_4\}$, and $\pi_{omap}(e_{199214})(Package)=\{p_2\}$.

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

4 A general framework for action-oriented process mining

In this section, we introduce a general framework for action-oriented process mining. Figure 3 explains the overview of the proposed framework. It is mainly composed of two components. Firstly, the *constraint monitor* converts an event stream into a *constraint instance stream* by evaluating a set of *constraints* entailing *constraint formula* to explain what/how and time moments to specify when. Each *constraint instance* describes the (non) violation of a constraint. Second, the *action engine* transforms the constraint instance stream into an *action instance stream* by assessing the necessity of management actions and generating *action instances*. Each action instance depicts a *transaction/workflow* to be executed by the information system to mitigate the risks caused by the violations.

In the following, we explain the components with formal definitions and examples.

4.1 Constraint monitor

A constraint monitor analyzes an event stream to evaluate (non) violations of constraints. Each (non) violation of a constraint occurs in a certain context.

Definition 4.1 (*Context*) A context $ctx \in \mathcal{P}(\mathbb{U}_{proc}) \times \mathcal{P}(\mathbb{U}_{act}) \times \mathcal{P}(\mathbb{U}_{res}) \times \mathbb{U}_{omap} \times \mathbb{U}_{vmap}$ 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 $vmap$. $\mathbb{U}_{ctx}$ is the set of all possible contexts.

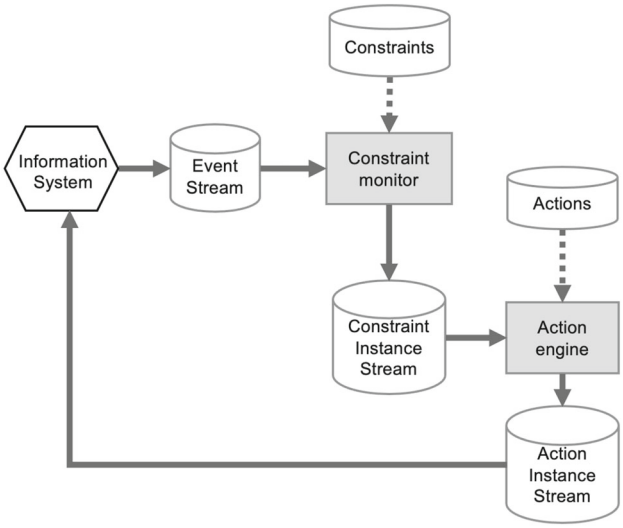

Fig. 3 Overview of the general framework for action-oriented process mining

For instance, *C1* in Sect. 3.1 could be violated by package p_2, which contains item i_3 and i_4 in the process *OH* to which *Adams* execute the activity *fail delivery*. ctx_1 describes the context, i.e., ctx_1=($\{OH\}$, $\{fail\ delivery\}$, $\{Adams\}$, $omap_1$, $vmap_1$), where $omap_1(Package)$=$\{p_2\}$ and $omap_1(Item)$=$\{i_3, i_4\}$.

Given a context ctx=($Proc$, Act, Res, $omap$, $vmap$) $\in \mathbb{U}_{ctx}$, $\pi_{proc}(ctx)$=$Proc$, $\pi_{act}(ctx)$=Act, $\pi_{res}(ctx)$=Res, $\pi_{omap}(ctx)$=$omap$, and $\pi_{vmap}(ctx)$=$vmap$. For instance, $\pi_{proc}(ctx_1)$=$\{OH\}$, $\pi_{act}(ctx_1)$=$\{$*fail delivery*$\}$, $\pi_{res}(ctx_1)$=$\{Adams\}$, $\pi_{omap}(ctx_1)$=$omap_1$, and, $\pi_{vmap}(ctx_1)$=$vmap_1$.

The constraint formula evaluates if violations occur in a specific context by analyzing the events in a given event stream that are relevant to a given time window.

Definition 4.2 (*Constraint formula*) We define $\mathbb{U}_{outc}$=$\{OK, NOK\}$ to be the universe of outcomes. $cf \in (\mathbb{U}_{stream} \times \mathbb{U}_{tw}) \rightarrow \mathcal{P}(\mathbb{U}_{ctx} \times \mathbb{U}_{outc})$ is a constraint formula. $\mathbb{U}_{cf}$ denotes 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=(02-01-2022 09:00, 03-01-2022 09:00), it evaluates if any package in the time window undergoes *fail delivery*. Since there exists *fail delivery* for package p_2, $(ctx_1, NOK) \in cf_1(S, tw_1)$.

In this paper, we do not assume specific approaches to instantiate the constraint formula. As presented in Sect. 2.2, several approaches are proposed in the field of process mining to implement constraint formula, including conformance checking techniques [4], Linear Temporal Logic [5], and rule-driven approaches based on Petri-net patterns [6].

Furthermore, predictive process monitoring techniques can be deployed to instantiate *predictive* constraint formulas. Such constraint formulas evaluate if violations *will* occur in a certain context by analyzing the events in a given event stream with a given time window. Suppose cf_2 is a predictive constraint formula instantiated to predict the violation of the constraint described in *C1* of the motivating example, and it predicts that the package loaded late in the evening fails. Given the event stream S that contains events listed in Table 1 and time window tw_1=(02-01-2022 09:00, 03-01-2022 09:00), $(ctx_2, NOK) \in cf_2(S, tw_1)$, where ctx_2=($\{OH\}$, $\{faildelivery\}$, {}, $omap_1$, $vmap_1$), where $omap_1(Package)$=$\{p_3\}$ and $omap_1(Item)$=$\{i_5, i_6\}$, since p_3 is loaded late in the evening at 21:55.

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

Suppose c_1=(cf_1, TM_1) where (03-01-2022 09:00, (02-01-2022 09:00, 03-01-2022 09:00)) $\in TM_1$. For instance, we evaluate cf_1 at 03-01-2022 09:00 with the events related to time window (02-01-2022 09:00,03-01-2022 09:00).

A constraint instance specifies when and whether a violation happens in a certain context by a constraint formula. A constraint instance stream is a collection of unique constraint instances.

Definition 4.4 (*Constraint instance stream*) A constraint instance $ci \in \mathbb{U}_{cf} \times \mathbb{U}_{ctx} \times \mathbb{U}_{time} \times \mathbb{U}_{outc}$ is a tuple of a constraint formula cf, a context ctx, a timestamp $time$, and an outcome $outc$. We let $\mathbb{U}_{ci}$ be the set of all possible constraint instances. A constraint instance stream CIS is a (possibly infinite) set of constraint instances, i.e., $CIS \subseteq \mathbb{U}_{ci}$. $\mathbb{U}_{cis}$ denotes the set of all possible constraint instance streams.

For instance, a constraint instance ci_1=(cf_1, ctx_1, 03-01-2022 09:00, NOK) denotes that cf_1 is violated at 03-01-2022 09:00 in context ctx_1.

Based on Definition 3-6, we define a constraint monitor. As shown in Fig. 4, it transforms an event stream into a constraint instance stream. Parameterized with a set of constraints, it evaluates the constraint formula in each constraint according to the corresponding set of time moments and generates constraint instances.

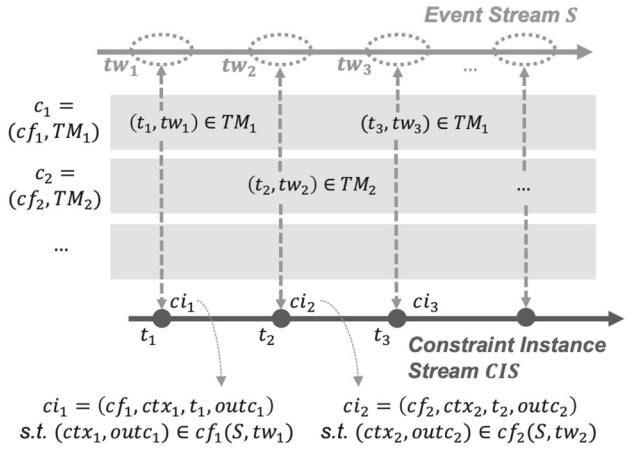

Fig. 4 Constraint monitor cm_C, where $C = (c_1, c_2, \dots)$, transforms an event stream into a constraint instance stream

Definition 4.5 (*Constraint monitor*) Let $C \subseteq \mathbb{U}_c$ be a set of constraints to be used for monitoring. $cm_C \in \mathbb{U}_{stream} \rightarrow \mathbb{U}_{cis}$ is the constraint monitor such that, for any $S \in \mathbb{U}_{stream}$, $cm_C(S)=\{(cf, ctx, time, outc) \in \mathbb{U}_{ci} | \exists_{TM,tm} (cf, TM) \in C \wedge tm \in TM \wedge time=\pi_t(tm) \wedge (ctx, outc) \in cf(S, \pi_{tw}(tm))\}$.

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/workflows that source information systems need to execute to mitigate the risk incurred by the constraint violations.

Definition 4.6 (*Transaction*) Let $\mathbb{U}_{op}$ be the universe of operations that are executed by information systems (e.g., send messages). A transaction $tr=(op, vmap) \in \mathbb{U}_{op} \times \mathbb{U}_{vmap}$ is a pair of an operation op and a parameter mapping $vmap$. $\mathbb{U}_{tr} \subseteq \mathbb{U}_{op} \times \mathbb{U}_{vmap}$ denotes the set of all possible transactions.

For instance, the action description $A1$ in Sect. 3.1 represents a transaction, tr_1=(*send-a-message*,$vmap'$) where $vmap'(recipient)$=*customer* and $vmap'(message)$= *"Select an alternative method for the delivery."*.

Note that an operation may involve multiple fine-grained operations (i.e., child operations). For instance, notifying the sales department may include sending e-mails to all staff, transferring alert signals to responsible staff, etc. We call the transaction with such composite operations as workflows.

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 4.7 (*Action formula*) An action formula $af \in (\mathbb{U}_{cis} \times \mathbb{U}_{tw}) \rightarrow \mathcal{P}(\mathbb{U}_{tr})$ is a function that maps a constraint instance stream and time window to a set of transactions. $\mathbb{U}_{af}$ 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. 3.1, and to produce the corresponding transaction, i.e., sending a message to a customer. Given constraint instance stream CIS and time window tw_1=(02-01-2022 09:00,03-01-2022 09:00), it assesses if there exist more than two constraint instances whose outcomes are NOK in the time window. If so, tr_1=(*send-a-message*, $vmap'$) $\in af(CIS, tw_1)$.

The implementation of action formulas relies on the domain knowledge of process experts. As shown in the motivating example, it is the domain knowledge to ensure that sending a message to a customer is to reduce delivery failures. However, the root-cause analysis of violations could help domain experts to provide useful insights on effective management actions. For instance, if the root-cause of a violation (e.g., failed delivery) is provided (e.g., the delivery time is too late), a domain expert can design an action formula to produce the transaction to change the delivery time to a earlier time. In case that a violation is predicted, the recent development of explainable prediction models can be used to analyze the predicted violation.

An action consists of 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 4.8 (*Action*) An action $a=(af, TM) \in \mathbb{U}_{af} \times \mathcal{P}(\mathbb{U}_{tm})$ is a pair of an action formula af and a set of time moments TM. $\mathbb{U}_a$ denotes the set of all possible actions.

Suppose $a_1=(af_1, TM_1)$ where (03-01-2022 09:00, (02-01-2022 09:00, 03-01-2022 09:00)) $\in TM_1$. We implement af_1 at 03-01-2022 09:00 with the constraint instances related to time window (02-01-2022 09:00,03-01-2022 09:00).

An action instance indicates when and which transaction is required. An action instance stream is a collection of unique action instances.

Definition 4.9 (*Action instance stream*) An action instance $ai=(af, tr, time) \in \mathbb{U}_{af} \times \mathbb{U}_{tr} \times \mathbb{U}_{time}$ is a tuple of an action formula af, a transaction tr, and a timestamp $time$. We let $\mathbb{U}_{ai}$ be the set of all possible action instances. An action instance stream AIS is a (possibly infinite) set of action instances, i.e., $AIS \subseteq \mathbb{U}_{ai}$. $\mathbb{U}_{ais}$ denotes the set of all possible action instance streams.

For instance, an action instance ai_1=(af_1, tr_1, 03-01-2022 09:00) denotes that the transaction tr_1 needs to be executed at 03-01-2022 09:00 according to af_1.

Based on Definition 8–11, we define an action engine. As shown in Fig. 5, it transforms a constraint instance stream into a action instance stream. Being parameterized with a set of actions, it evaluates the action formula in each action to assess the necessities of transactions according to the corresponding set of time moments and produces action instances.

Definition 4.10 (*Action engine*) Let $A \subseteq \mathbb{U}_a$ be a set of actions used by the action engine. $ae_A \in \mathbb{U}_{cis} \rightarrow \mathbb{U}_{ais}$ is the action engine such that, for any $CIS \in \mathbb{U}_{cis}$, $ae_A(CIS)$={$(af, tr, time) \in \mathbb{U}_{ai} | \exists_{TM,tm} (af, TM) \in A \wedge tm \in TM \wedge time$=$\pi_t(tm) \wedge tr \in af(CIS, \pi_{tw}(tm))$}.

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 that mitigate risks caused by the past constraint violations.

Conceptually, the action engine can be extended to incorporate *action recommender systems*. An action recommender system $recommend \in \mathbb{U}_{ais} \rightarrow \mathbb{U}_{ais}$ evaluates candidate action instances to provide efficient action instances. One can implement the recommender system by ranking candidate action instances and filtering ones with higher ranks. Moreover, it can be implemented such that it resolves conflicting action instances (e.g., arranging a meeting of a patient in a doctor's office and simultaneously routing the patient to a blood test) by modifying the action instances (e.g., postponing the arrangement of the meeting until the blood test is completed).

5 Cube-based action engine

A constraint instance indicates when a violation does (not) happen in a specific context in terms of a constraint (cf. Definition 4.4). It is essential to comprehensively analyze these different aspects (i.e., constraint, context, and time) and generate needed actions. Moreover, it is crucial to aggregate constraint instances in a meaningful abstraction level (e.g., violations by an order during a week) to produce appropriate actions.

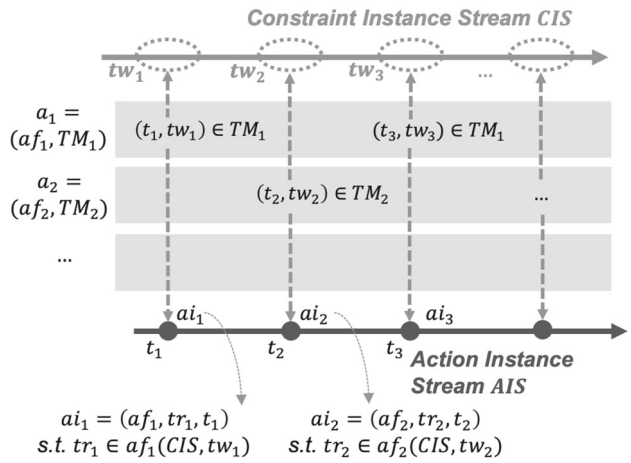

Fig. 5 Action engine ae_A, where $A = (a_1, a_2, \ldots)$, transforms a constraint instance stream into an action instance stream

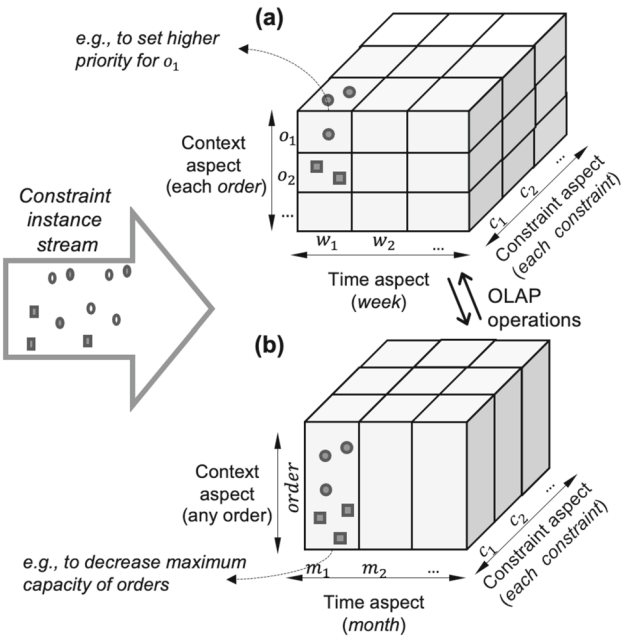

Fig. 6 Examples of structuring a constraint instance stream into constraint cubes and conducting multi-dimensional analysis on constraint instances

Cube-based action engine realizes these requirements by utilizing *constraint cube* and OLAP operations defined over the cube. Figure 6 describes examples of constructing constraint cubes from a constraint instance stream and generating actions by analyzing the constraint instances belonging to each cell of the cube. Each cell is characterized by different aspects of the constraint instance in a proper abstraction level. For instance, each cell in Fig. 6a contains constraint instances of respective orders (i.e., context aspect) and individual constraints (i.e., constraint aspect) in a week level (i.e., time aspect). By analyzing the constraint instances in each cell, the action engine generates actions such as setting higher priority for the corresponding order.

By using OLAP operations, such as slice, dice, roll-up, and drill-down, we can analyze the constraint instances using different selections and granularities. By rolling-up from individual orders to entire order in the context dimension and from week to month in the time dimension, we can analyze the constraint instances relating any orders in a specific month per constraint, as depicted in Fig. 6b. This may result in actions such as decreasing the maximum capacity of orders in the process.

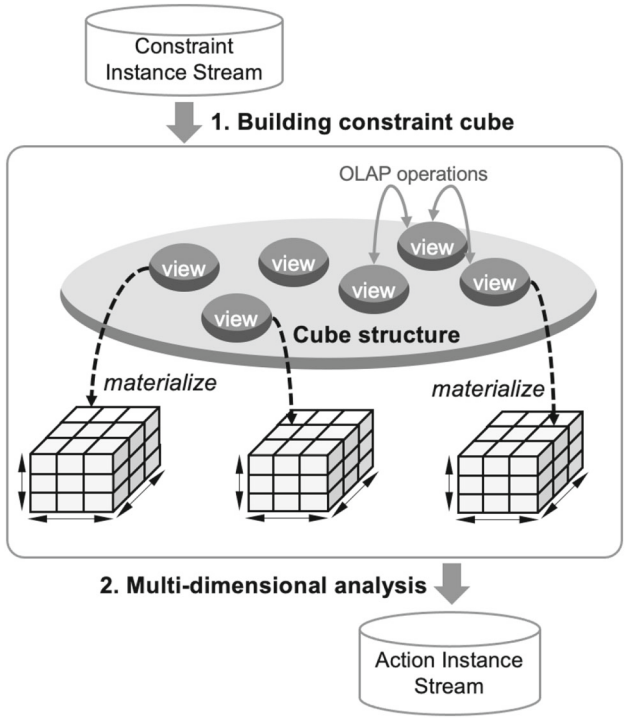

Fig. 7 Core components of a cube-based action engine: (1) constraint cube and (2) multi-dimensional analysis

Figure 7 describes core components of a cube-based action engine, i.e., 1) constraint cube and 2) multi-dimensional analysis. First, it entails building blocks and functionalities of constraint cubes. The cube structure provides a basic scheme upon which a variety of views can be defined. Each view is materialized to enable a multi-dimensional analysis of constraint instances in appropriate abstraction levels. OLAP operations support the flexible conversion between different views. Second, cube-based action formulas, configured with a view on the constraint cube structure, produce transactions by analyzing the constraint instances in a materialized constraint cube.

In the following, we explain the components of a cube-based action engine with formal definitions and examples.

5.1 Constraint cube

5.1.1 Building blocks and functionalities

In a constraint cube, we model three aspects of constraint instances (i.e., constraint, context, and time) into *dimensions*. Each dimension has different abstraction levels (i.e., elements) and the relationship between elements (i.e., hierarchy). As shown in Fig. 8a, the time dimension, representing the time aspect of the constraint instance, may consist of elements such as *date*, *month*, *quarter*, *year*, *season*, and *week*. For constraint instance $ci \in \mathbb{U}_{ci}$, the date element indicates the date of $\pi_{time}(ci)$. The elements have hierarchy, e.g., from date to month and from month to quarter. Each element involves the set of possible values. For instance, *month* element can have values in $\{\ldots, 01.2021, 02.2021, \ldots\}$. We define dimension formally as follows:

Definition 5.1 (*Dimension*) A dimension is a pair $dim=((E, H), vsmap)$ where

- (E, H) is a directed acyclic graph with nodes $E \subseteq \mathbb{U}_{attr}$ and a set of directed edges $H \subseteq E \times E$, and
- $vsmap \subseteq E \rightarrow \mathcal{P}(\mathbb{U}_{val})$ is a function defining the set of possible values for each element.

$\mathbb{U}_{dim}$ denotes the set of all possible dimensions. Given $dim=((E, H, vsmap)) \in \mathbb{U}_{dim}$, E_{dim} and H_{dim} denote the set of elements and hierarchies of the dimension, respectively.

For instance, the time dimension depicted in Fig. 8a is defined as follows: $dim_t=((E_t, H_t), vsmap_t)$ where $E_t=\{date, month, quarter, year, season, week\}$,$H_t=\{(date, month), (month, quarter), \ldots\}$, and $vsmap(date)=\{01.01..2021, 02.01.2021, \ldots\}$.

A constraint cube structure is composed of constraint dimension $dim_{cst} \in \mathbb{U}_{dim}$, context dimension $dim_{ctx} \in \mathbb{U}_{dim}$, and time dimension $dim_t \in \mathbb{U}_{dim}$, each of which represents the constraint, context, and time aspects of constraint instances, respectively. Each dimension is independent of each other, i.e., the dimensions do not have common elements.

Definition 5.2 (*Constraint cube structure*) A constraint cube structure $CCS = \{dim_{cst}, dim_{ctx}, dim_t\} \subseteq \mathbb{U}_{dim}$ such that $\forall_{\{dim_1, dim_2\} \subseteq CCS}\ E_{dim_1} \cap E_{dim_2}=\emptyset$. E_{ccs} denotes the set of all element defined over CCS, i.e., $E_{ccs}=\bigcup_{dim \in CCS} E_{dim}$.

Depending on business processes and purposes, each dimension has different elements, hierarchies, and possible element values, thus resulting in a different constraint cube structure. As an example, we realize the constraint cube structure based on the order handling process in Sect. 3.1, as described in Fig. 8.

First, we assume that constraints are classified to cost, time, and quality categories, according to their influences on the process. For instance, *C1*—there must be no "fail delivery" for any package— has influences in the cost perspective, thus belonging to *cost* category. In addition, depending on the severity of the violation of the constraint, different levels are given to each category, ranging from 1 to 5, e.g., *C1* belongs to *cost-3*, forming the sub-category element. Consequently, $dim_{cst}=((E_{cst}, H_{cst}), vsmap_{cst})$ where $E_{cst}=\{individual, sub\text{-}category, category\}$, $H_{cst}=\{(individual, sub\text{-}category), (sub\text{-}category, category)\}$, and $vsmap_{cst}(category)=\{cost, time, quality\}$.

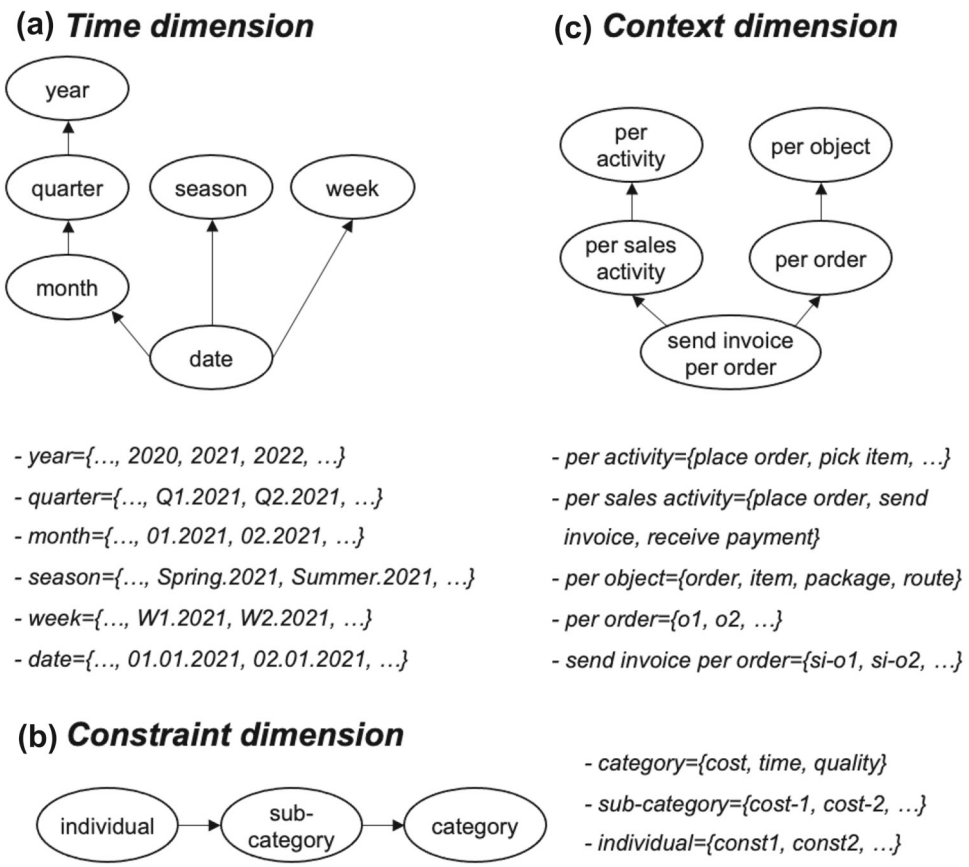

Fig. 8 An example of dimensions

The context dimension includes several useful entities defined for the process. First, *send-invoice-per-order* indicates the context where the send invoice activity is performed for each order. The *per-sales-activity* denotes the context where the activities related to sales occur such as place order, send invoice, and receive payment. The *per-activity* denotes the context of executing respective activities in the process, while the *per-order* means the context of individual orders. Finally, *per-object* includes order, item, etc. A hierarchy among the elements is also described in Fig. 8, e.g., *per-object* is in higher hierarchy than *per-order* since orders belong to the order object.

An element matching assigns element values to constraint instances.

Definition 5.3 (*Element matching*) Let CCS be a constraint cube structure and $CIS \in \mathbb{U}_{cis}$ be a constraint instance stream. $em_e \in CIS \rightarrow \mathcal{P}(\mathbb{U}_{val})$ relates values of $e \in E_{ccs}$ to constraint instances such that, for any $ci \in CIS$, $em_e(ci) \subseteq vsmap(e)$.

For instance, $em_{per\text{-}order}$ assigns values of $per\text{-}order \in E_{dim_{ctx}}$ to constraint instances, i.e., for any constraint instance ci, $em_{per\text{-}order}(ci)=\pi_{omap}(\pi_{ctx}(ci))(order)$. Some element matchings may require additional functions. For instance, $em_{category}$ assigns values of $category \in dim_{cst}$ to constraint instances, i.e., for any constraint instance ci, $em_{category}(ci)=cat(\pi_{cf}(ci))$ where cat is a function that relates constraint formulas to their categories.

While conducting analysis using a constraint cube, its structure remains the same, whereas its view changes to support the multi-dimensional analysis. A constraint cube view determines which dimensions to select in which granularity of elements with what values. For instance, one possible view is to select the constraint dimension with the granularity of *individual*, the context dimension with *object*, and time dimension with *month*. It enables to analyze the constraint instances for each constraint and object type (e.g., order, item, etc.) in each month.

Definition 5.4 (*Constraint cube view*) Let CCS be a constraint cube structure. A constraint cube view of CCS is a tuple $ccv=(D_{sel}, gran, sel)$ such that:

- $D_{sel} \subseteq CCS$ are the selected dimensions,
- $gran \in D_{sel} \rightarrow E_{ccs}$ is a function defining the granularity for each selected dimensions, and
- $sel \in E_{ccs} \rightarrow \mathcal{P}(\mathbb{U}_{val})$ is a function selecting a set of the values of the elements in each dimension such that, for any $e \in E_{ccs}$, $sel(e) \subseteq vsmap(e)$.

The example view described above can be defined as $ccv_1=\{D_{sel}, gran, sel\}$ where $D_{sel}=\{dim_{cst}, dim_{ctx}, dim_t\}$, $gran(dim_{cst})=individual$, $gran(dim_{ctx})=object$, $gran(dim_t)=month$, $sel(individual)=\{const_1, const_2\}$, $sel(object)=\{order\}$, and $sel(month)=\{01.2021, 02.2021, \ldots, 12.2021\}$.

A constraint cube view constitutes a collection of cells. Each cell is characterized by specific values for different dimensions defined in the constraint cube view. When materialized with a constraint instance stream, each cell involves the constraint instances associated with the values.

Definition 5.5 (*Cell set*) Let CCS be a constraint cube structure and $ccv=(D_{sel}=\{dim_1, \ldots, dim_n\}, gran, sel)$ be a constraint cube view over CCS. $CS_{ccv}=(EV_{dim_1} \times \cdots \times EV_{dim_n})$ is a cell set of ccv, where for any $dim_i \in D_{sel}$, $EV_{dim_i}=gran(dim_i) \times sel(gran(dim_i))$ is a set of element-value sets.

For instance, $CS_{ccv_1}=\{$

$((individual, const_1), (object, order), (month, 01.2021)),$
$((individual, const_2), (object, order), (month, 01.2021)),$
$((individual, const_1), (object, order), (month, 02.2021)),$
$((individual, const_2), (object, order), (month, 02.2021)),$
$\ldots,$
$((individual, const_2), (object, order), (month, 12.2021))\}$

Next, we fill each cell in the cell set of the constraint cube view with the constraint instances from a constraint instance stream.

Definition 5.6 (*Materialized constraint cube*) Let a constraint cube structure CCS and $CIS \in \mathbb{U}_{cis}$ a constraint instance stream. Let $ccv=(D_{sel}, gran, sel)$ be a constraint cube view of CCS. The materialized constraint cube view of ccv on CIS is $mv_{ccv,cis} \in CS_{ccv} \to \mathcal{P}(CIS)$ such that, for any $c \in CS_{ccv}$, $mv_{ccv,cis}(c)=\{ci \in CIS | \forall_{dim \in D_{sel}} \exists_{v \in em_{gran(dim)}(ci)} (gran(dim), v) \in c \wedge \forall_{e \in E_{ccs}} em_{gran(dim)}(ci) \subseteq sel(e)\}$.

Each constraint instance belonging to a cell must correspond to the (element, value) pair of the cell. For instance, for a cell $c=\{(individual, const_1), (object, order), (month, 01.2021)\}$ in cell set CS_{ccv_1}, only constraint instances related to $const_1$, $order$ and 01.2021 are considered. In addition, the resulting constraint instances must not be filtered out by the constraint cube view.

Figure 9 describes an example of materializing constraint cube view ccv_1. Each cell in the materialized constraint cube contains the constraint instances related to an individual constraint for the entire order in a month.

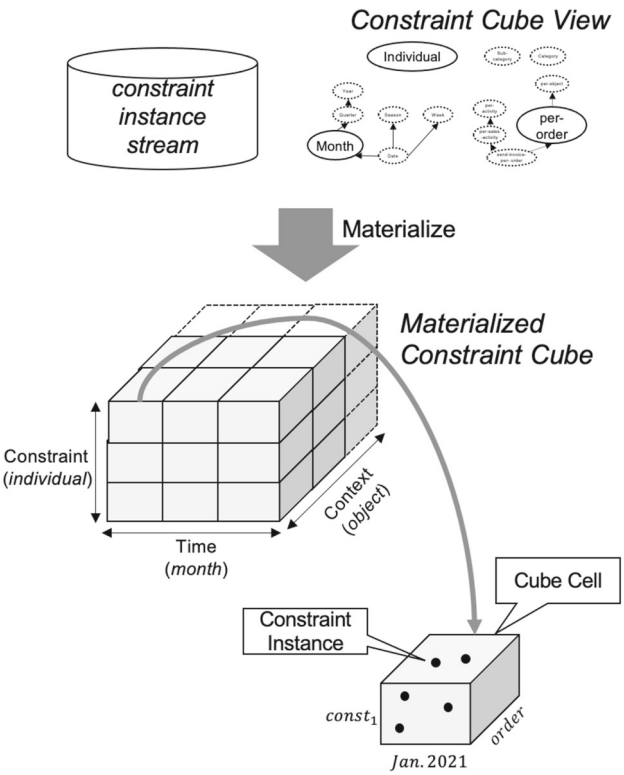

Fig. 9 An example of materialized constraint cubes

5.1.2 OLAP operation

Classical OLAP operations are defined for the constraint cube to support multi-dimensional analysis on constraint instances.

5.1.2.1 Slice operation

This operation enables to select specific values of one dimension. For instance, one can select 01.2021 from the time dimension by removing the dimension from constraint cube view ccv_1 and only considering the constraint instances in 01.2021.

Let CCS be a constraint cube structure and $ccv=(D_{sel}, gran, sel)$ a view of CCS. Let $dim=((E, H), vsmap) \in D_{sel}$ be a dimension. A filtering function $filt \in (E \to \mathcal{P}(\mathbb{U}_{val}))$ relates sets of values to the elements such that, for any $e \in E$, $filt(e) \subseteq vsmap(e)$. A slice operation $slice_{dim,filt}$ selects specific values in dim using $filt$, i.e., $slice_{dim,filt}(ccv)=(D'_{sel}, sel', gran)$ such that

- $D'_{sel}=D_{sel} \setminus \{dim\}$ is the new set of selected dimensions, and
- $sel' \in E \to \mathcal{P}(\mathbb{U}_{val})$ is the new selection function such that

$$sel'(e)=\begin{cases} filt(e) & \text{if } e \in E_{dim} \\ sel(e) & \text{if } e \in (E_{ccs} \setminus E_{dim}) \end{cases}$$

5.1.2.2 Dice operation

This operation allows to select specific values of multiple dimensions. For instance, one can select 01.2021 and 02.2021 for the time dimension and $const_1$ for the constraint dimension from constraint cube view ccv_1. Note that no dimensions are removed, but the constraint instances that are related to 01.2021 or 01.2021 and $const_1$ are considered.

Let $filt_{dim}$ denote the filtering function defined over dimension dim. We let F_D denote the collection of filtering functions defined over $D{=}\{dim_1, \ldots, dim_n\} \subseteq D_{sel}$, i.e., $F_D{=}\{filt_{dim_1}, \ldots, filt_{dim_n}\}$. A dice operation $dice_{D,F_D}$ select specific values in $dim \in D$ using the filtering function $filt_{dim} \in F_D$, i.e., $dice_{D,F_D}(ccv){=}(D_{sel}, sel', gran)$ such that $sel' \in E_{ccs} \to \mathcal{P}(\mathbb{U}_{pval})$ is the new selection function such that, for any $dim \in D$,

$$sel'(e){=}\begin{cases} filt(e) & \text{if } e \in E_{dim} \\ sel(e) & \text{if } e \in (E_{ccs} \setminus E_{dim}) \end{cases}$$

5.1.2.3 Roll-up and drill-down

These operations change the granularity of dimensions. *roll-up* changes the granularity of a dimension to the higher hierarchy, e.g., from *date* to *month*, whereas *drill-down* changes it to the lower hierarchy, e.g., from *month* to *date*.

Given a constraint cube view $ccv{=}(D_{sel}, gran, sel)$, a granularity change function $chgr_{dim,e}$ changes the granularity of $dim{=}((E, H), vsmap) \in D_{sel}$ to $e \in E$, i.e., $chgr_{dim,e}(ccv){=}(D_{sel}, gran', sel)$ such that $gran'(dim){=}e$ and, for any $dim' \in D_{sel} \setminus \{dim\}$, $gran'(dim'){=}gran(dim')$. A roll-up operation $rlup_{dim,e}$ climbs up the hierarchy in dim to e, i.e., $rlup_{dim,e}(ccv){=}chgr_{dim,e}(ccv)$ such that $\exists_{e \in E_{dim}}\ (gran(dim), e) \in H_{dim}$. A drill-down operation $drdown_{dim,e}$ climbs down the hierarchy in dim to e, i.e., $drdown_{dim,e}(ccv){=}chgr_{dim,e}(ccv)$ such that $\exists_{e \in E_{dim}}\ (e, gran(dim)) \in H_{dim}$.

5.2 Multi-dimensional analysis

In a cube-based action engine, *cube-based action formulas* produce transactions by conducting multi-dimensional analysis using constraint cubes. The action formula consists of three components: 1) constraint cube view to materialize constraint cubes in appropriate abstraction levels, 2) assessment function to evaluate the necessity of actions, and 3) transaction mapping to assign transactions.

First, an assessment function analyzes the constraint instances collected in a proper abstraction level to discover critical violations of constraints requiring management actions. Different measures can be adopted for the analysis, such as basic aggregation (e.g., sum and count) and more sophisticated measures (e.g., regression and classification).

Definition 5.7 (*Assessment*) An assessment function $as \in \mathcal{P}(\mathbb{U}_{ci}) \to \mathbb{B}$ relates Boolean values to sets of constraints instances such that, for any $CI \subseteq \mathcal{P}(\mathbb{U}_{ci})$, $as(CI) = true$ if an action is necessary, $as(CI) = false$ otherwise.

Suppose that any order needs predefined management actions if it violates a service level agreement more than 10 times in the last three days. An assessment function as_1 assess the necessity of actions, i.e., for any relevant set of constraints instances $CI \subseteq \mathbb{U}_{ci}$, $as_1(CI){=}true$ if $|\{ci \in CI | \pi_{outc}(ci){=}NOK\}| > 10$, $false$ otherwise.

In case that the necessity for actions arises, a transaction mapping relates management transactions to cells. If as_1 is evaluated to be true for the constraint instances belonging to a cell, it produces the action of sending an email to the order manager warning frequent violation of the service level agreement by the corresponding order.

Definition 5.8 (*Transaction mapping*) Let ccv be a constraint cube view and CS the cell set of ccv. A transaction mapping $tmap \in CS \nrightarrow \mathbb{U}_{tr}$ is a partial function that assigns transactions to cells. We denote $tmap(c){=}\perp$ if $c \notin dom(tmap)$.

Given the constraint cube cell $c_1{=}((individual, const_1), (order, o_1), (month, 01.2021)) \in CS_{ccv_1}$, a transaction mapping $tmap_1$ generates a transaction $tr_1 \in \mathbb{U}_{tr}$ as follows: $tmap_1(c_1){=}tr_1{=}(op, pmap)$ such that op is *"send an email to manager"*, $pmap(violation){=}const_1$, and $pmap(subject){=}o_1$, where $violation, subject \in dom(pmap)$.

A cube-based action formula materializes the constraint cube based on a constraint cube view, and analyze the necessity of actions in each cell using an assessment function, and produce corresponding actions using a transaction mapping.

Definition 5.9 (*Cube-based action formula*) Let $CIS \in \mathbb{U}_{cis}$ be a constraint instance stream and CCS a constraint cube structure that is compatible with CIS. A cube-based action formula $af_{cube} \in (\mathbb{U}_{cis} \times \mathbb{U}_{tw}) \to \mathcal{P}(\mathbb{U}_{tr})$ consists of

- a constraint cube view ccv of CCS,
- an assessment as, and
- a transaction mapping $tmap$.

Given any constraint instance stream $CIS \in \mathbb{U}_{cis}$ and a time window $tw \in \mathbb{U}_{tw}$, it computes the required transactions, i.e., $af_{cube}(CIS, tw){=}\{tmap(c) | c \in CS_{ccv} \wedge as(mv_{ccv,cis}(c)){=}true\}$.

6 Evaluation

In this section, we first present the implementation of the proposed framework using the cube-based action engine and evaluate the effectiveness of the framework in improving business processes by conducting experiments with an artificial information system and a real-life SAP ERP system.

6.1 Implementation

The general framework is implemented as a plug-in of *ProM*,[2] 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* in the nightly build of ProM. Moreover, the technical manual of the plug-in is available at https://github.com/gyunamister/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 a JSON-based *Object-Centric Event Log (OCEL)*[3] format [44], storing events along with their related objects. Below is an example of the OCEL format representing the event:

```
''ocel:events'': {
  ''199210'': {
    ''ocel:activity'': ''place_order'',
    ''ocel:timestamp'': "2021–01–01 09:55:00.000+01:00'',
    ''ocel:omap'': [
      ''o7'',
      ''i8'',
      ''i9''
    ],
    ''ocel:vmap'': {
      ''resource'': ''Louis'',
      ''process'': ''OH'',
      ''type'': ''Gold''
    }
  },
}
```

The constraint formula and action formula are defined by *Constraint Formula Language (CFL)* and *Action Formula Language (AFL)*, respectively. The CFL specifies an in-built function of the plug-in and its required parameters, and the AFL specifies the constraint cube view, assessment function, and transaction mapping that compose the cube-based action formula. For the syntax and examples of the CFL and AFL, we refer readers to the tool manual[4].

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. Below is an example of an action stored in AIS format:

```
<?xml version=''1.0'' encoding=''UTF–8''?>
<action–instance–stream>
  <action–instance>
    <action–formula>ask–alternatives</action–formula>
    <operation>send–message</operation>
    <parameter–mapping>
      <parameter name=''customer''>
        <value>order</value>
      </parameter>
    </parameter–mapping>
    <timestamp>09:55 01–01–2021</timestamp>
  </action–instance>
  ...
</action–instance–stream>
```

The action-instance-stream tag corresponds to $ais \in \mathbb{U}_{ais}$. The action-instance tag relates $ai = (af, (op, vmap), time) \in ais$, whereas action-formula , operation , parameter-mapping , and timestamp corresponds to af, op, $vmap$, and $time$, respectively.

6.2 Experiments

In order to evaluate the effectiveness of the proposed framework, we conduct experiments using the implementation. Below are the research questions that we aim to answer in the experiments:

- 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?

We conduct experiments using two business processes: one that is supported by an artificial information system and the other supported by real-life SAP systems. In the following, we explain each of them.

6.2.1 Artificial information system

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.

6.2.1.1 Experimental design

We assume that, based on a service level agreement, an order must be delivered within 72 hours after its placement. However, it has been reported that some orders violate the agreement, generating additional costs. To deal with the delayed orders, we can take several actions such as setting higher priorities for the orders to assign resources earlier than normal cases and sending notifications to the corresponding order manager to promote the intensive care of the orders.

In this regard, we define a constraint formula formulating that an order must be delivered in 72 hours ($cf_{oh,1}$). Besides, we formulate two action formulas: 1) setting a higher priority for any order that violates $cf_{oh,1}$ in the last 24 hours ($af_{oh,1}$) and

[2] http://www.promtools.org.

[3] http://ocel-standard.org.

[4] https://github.com/gyunamister/ActionOrientedProcessMining.

2) sending a warning message to a case manager for the order that violates $cf_{oh,1}$ twice in the last 48 hours ($af_{oh,2}$). The former allows the delayed order to be allocated to resources earlier than others in the system, reducing the waiting time for its activities. The latter reduces the time taken for resources to process the activities of the delayed order in the system, decreasing the processing time of the order.

The information system continuously generates events and updates an event stream. The constraint monitor analyzes the event stream using $cf_{oh,1}$ every 24 hours and produces constraint instances that are added to the constraint instance stream. The action engine analyzes the constraint instance stream using $af_{oh,1}$ and $af_{oh,2}$ every 24 hours in accordance with the constraint monitor and generates necessary actions. A dedicated gateway for the information system translates the actions and apply them to the information system.

6.2.1.2 Experimental Results

Figure 10 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 of each order (i.e., from the placement of an order to its delivery) and the green oval arrow denotes allowable delivery time (i.e., 72 hours). The red rectangle indicates the time when the violation of $cf_{oh,1}$ happens. Any order whose delivery time is outside the green arrow is detected by the constraint monitor every 24 hours.

For each delayed order, the higher priority is set in line with its detection ($af_{oh,1}$). Furthermore, the order with longer delays are notified to the corresponding case manager ($af_{oh,2}$). For instance, o_{10} was placed at 66 and, according to the agreement, was supposed to be delivered until 138. The constraint monitoring at 144 detected the violation of the agreement by o_{10} and higher priority was given to it. In the next monitoring (i.e., after 24 hours), the order was still not completed, thus requiring a notification to the case manager.

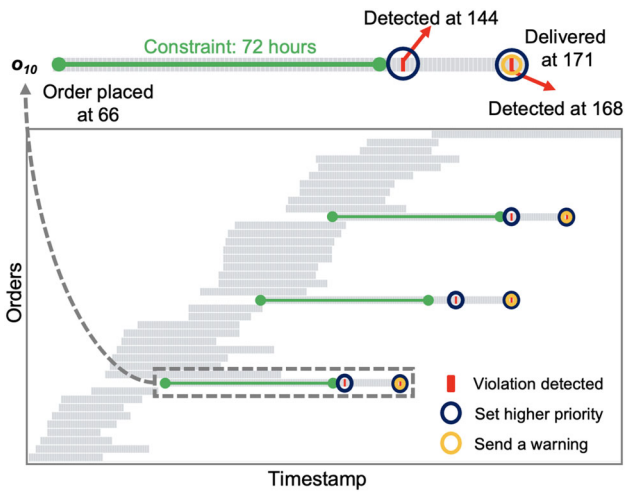

Fig. 10 The results of constraint monitor and action engine on 40 selected orders

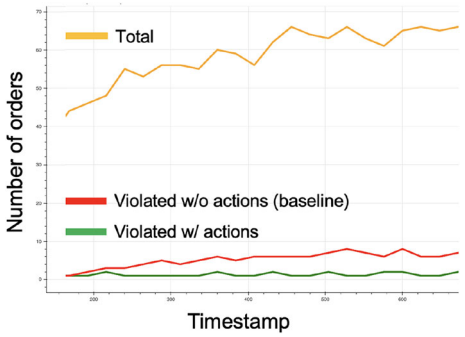

Fig. 11 Number of total/violated orders for 30 days

Figure 11 reports the experimental result related to RQ3. The figure shows the number of violated orders for 30 days. The yellow line indicates the total number of orders by time. The red line represents the number of violated orders when the actions are not applied, whereas the green line indicates the number of violations after the actions are applied to the information system. The number of violated orders is always lower when the actions are applied, validating the effectiveness of executing the actions in improving the performances of the business process.

6.2.2 SAP ERP system

The SAP Enterprise Resource Planning system (SAP ERP) supports the business processes of organizations by incorporating the key business functions such as sales and distribution, financial accounting, supply chain management, human resource management, etc. The order-to-cash (O2C) process is one of the core business processes in the sales and distribution supported by SAP ERP.

The O2C process deals with customer orders. Customers send inquiries to the company and sales corresponding quotations are sent to the customers in response. Sales managers convert the sales quotations into sales orders if customers accept the quotations. Next, corresponding deliveries are prepared, and the warehouse staff pick up and pack the items for the deliveries. After shipping the deliveries, invoices are prepared and sent to customers. Finally, the payments are collected and the invoices are cleared.

In this experiment, we utilize the SAP ERP system (SAP ERP ECC 6.0) supporting the O2C process of Global Bike Inc., which is a multinational enterprise producing and distributing bicycle products. The O2C process of the company consists of multiple objects (called documents) such as inquiry, quotation, order, item, delivery, shipment, and invoice. Employees involved in the O2C process interact with the system to deal with the orders from various customers. For instance, sales staff places orders by inserting the order information such as customers, sales organizations, sales divisions, materials, etc., into the system, as shown in Fig. 12. This, in turn, triggers

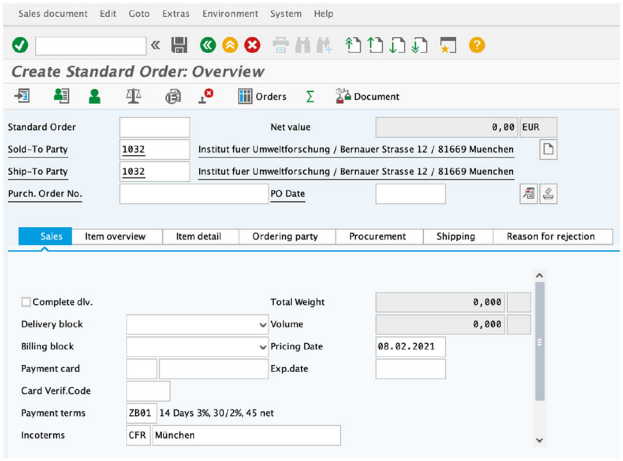

Fig. 12 A screenshot of SAP ERP ECC 6.0: placing order using transaction *VA01*

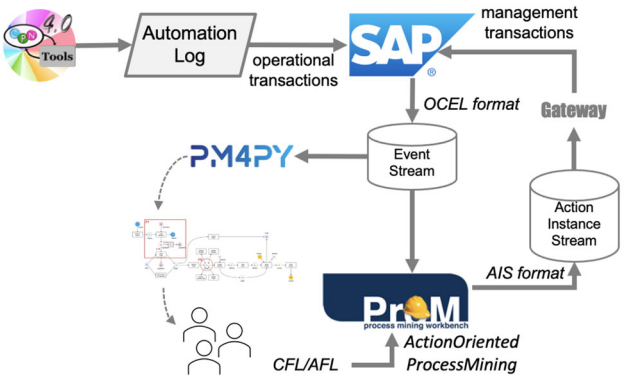

Fig. 13 Overview of the experimental design using a real-life SAP ERP system

transactions in the system (e.g., transaction code *VA01* in SAP ERP), thus supporting the business goals of the company.

6.2.2.1 Experimental Design

Figure 13 describes the overview of the experimental design based on the simulation approach that incorporates ERP systems [45]. First, we simulate the O2C process using *CPN Tools*[5] and generate the automation log where each record clones the behaviors of the employees in the company (e.g., *Adams* is supposed to place an order at 3 p.m.). Based on the automation log, we automate the execution of transactional operational transactions in the SAP ERP system. For instance, our automation tool directly executes *VA01* at 3 p.m. on behalf of *Adams* with necessary inputs such as customer information, materials, quantity, etc.

As such, we continuously generate events in the SAP ERP system and update the event stream. To scale the experiment, we use the scaling factor in time when automating the transactions. For instance, we automate the transactions of 30 days in 30 hours, by using the scaling factor of 24 (i.e., 1 hour in the experiment represents 24 hours in real-life business). Note that we still maintain the relative time difference between different transactions.

We analyze the event stream to identify improvement points in the process using process mining diagnostics supported in process mining tools such as *PM4Py*[6] and *ProM*, and define constraints and actions using *CFL* and *AFL*. The ProM plug-in generates an action instance stream in *AIS* format and the gateway for the SAP ERP system translates the management transactions described in action instances and executes them in the system.

Figure 14 shows the discovered process model of the O2C process represented as an object-centric Petri net [46]. Note that, for better representation, we only consider five objects (i.e., inquiry, quotation, order, delivery, and invoice) in the process and the one-to-one relationship between the objects. Based on the discovered process model, we identify two problems residing in the process. First, the conversion rate of quotations to orders is low (*P1*). 36 percent of the quotations are successfully converted to the orders, while 64 percent of the quotations are ending without positive responses from the customers. Second, we identify another problem that orders from customers are frequently changed after their confirmation, e.g., change price, change the quantity, etc., (*P2*). The frequently changed orders tend to have higher throughput time, i.e., frequent changes in orders are positively correlated to the throughput time of the orders.

Based on the identified problems, we define two constraint formulas, i.e., $cf_{o2c,1}$ and $cf_{o2c,2}$. First, $cf_{o2c,1}$ formulates that the inquiry from customers must be responded to in 24 hours to monitor late responses. It comes from the observation that the response time from receiving inquiry to sending quotation has a negative correlation to the likelihood of the acceptance of the quotations, i.e., the longer the response time is, the lower the likelihood is. Second, $cf_{o2c,2}$ formulates that an order should not be changed after the confirmation.

To deal with the constraint instances resulting from constraint formula $cf_{o2c,1}$, we define two action formulas: 1) alerting a sales manager whenever an inquiry with a long response time is detected ($af_{o2c,1}$) and 2) adding a temporary resource to deal with the late response when the ratio of late responses to normal responses is higher than 10 percent ($af_{o2c,2}$). The former reduces the waiting time for sending quotations to inquiries that are not responded for a long time. The latter reduces the waiting time for sending quotations to inquiries, in general, regardless of their response time.

To reduce the changes in orders, we define two action formulas: 1) notifying customers that the changes in the order after its confirmation may result in the delayed deliveries for any order entailing changes ($af_{o2c,3}$) and 2) adding an activity *approval by a sales manager* in the control flow of the process before confirming orders to avoid the changes caused by incorrect handling of orders if more than 20 changes occur in the last 24 hours ($af_{o2c,4}$). The former reduces the number

[5] http://cpntools.org/.

[6] https://pm4py.fit.fraunhofer.de/.

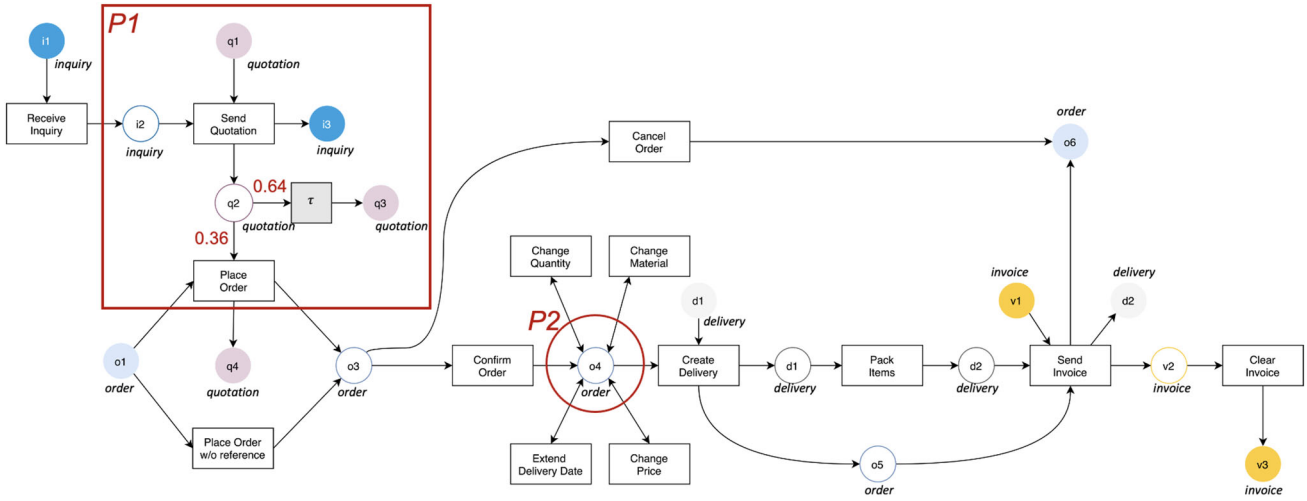

Fig. 14 An object-centric Petri net [46] describing the process model of the O2C process supported by SAP ERP system. The process has two problems: (1) low conversion rate (*P1*) and (2) changes in orders after the confirmation (*P2*)

of unnecessary changes in an order by changing the behavior of customers. The latter changes the occurrence of necessary changes in an order by ensuring that all incorrect information is corrected during the newly introduced activity.

The constraint monitor analyzes the event stream using $cf_{o2c,1}$ every 9 a.m., 12 p.m., and 3 p.m. and $cf_{o2c,2}$ every 9 a.m., and produces constraint instances. The action engine analyzes the constraint cube, materialized from the constraint instance stream, using $af_{o2c,1}$ and $af_{o2c,2}$ every 9 a.m., 12 p.m., and 3 p.m. to generate necessary actions. Besides, it evaluates $af_{o2c,3}$ and $af_{o2c,4}$ every 9 a.m. The generated actions are translated into SAP-executable transactions by a dedicated gateway and executed by the SAP ERP system.

6.2.2.2 Experimental Results

Figure 15 describes the responses to 151 inquiries for products—*precision pipes*—by multinational customers, to which quotations were sent between 5.12.2020 and 7.12.2020. Each row in the upper chart depicts the timeline of the response to an inquiry (i.e., the leftmost part indicates the time when the inquiry is received and the rightmost part is the time when the quotation is sent). The gray cell represents the legal part of the response (i.e., within 24 hours), whereas the rest indicates the lateness (i.e., beyond 24 hours). The vertical line represents the time when the constraint monitor evaluates constraint formula $cf_{o2c,1}$ and also the action engine evaluates action formula $af_{o2c,1}$ and $af_{o2c,2}$.

The yellow square box indicates the time when the constraint instances with non-violated outcomes are produced, whereas the red square box denotes the time when the violating constraint instances are generated. As indicated by the monitoring vertical line and the (il)legal part of the responses, the constraint monitor successfully produces (non) violating constraint instances. For instance, on 5.12.2020 at 9 a.m., the constraint monitor generates five violating constraint instances ab-out the inquiries reported as late responses and 41 non-violating constraint instances.

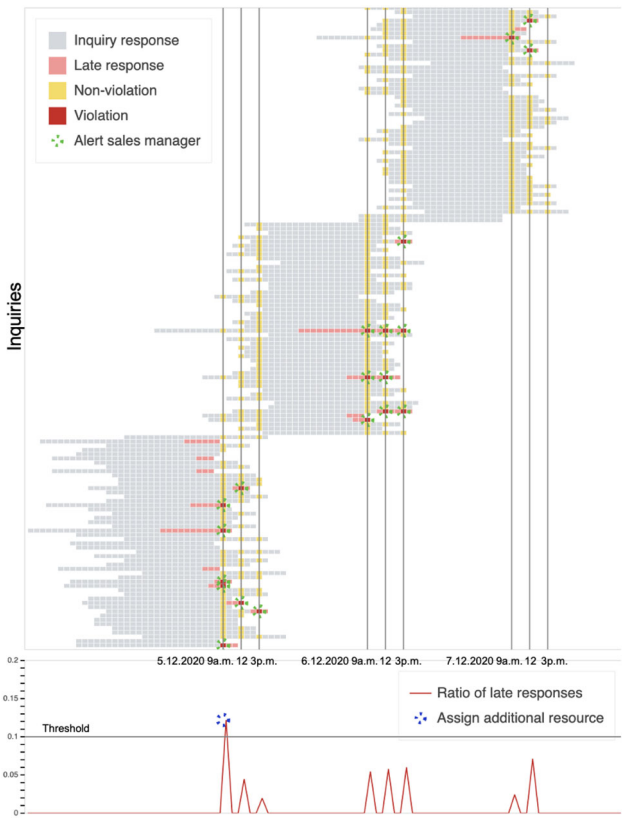

Fig. 15 The upper part of the chart describes 151 inquiries and the evaluation results by the constraint monitor using $cf_{o2c,1}$ and the actions by the action engine using $af_{o2c,1}$. The chart on the lower side describes the ratio of detected late responses and the generation of actions by the action engine using $af_{o2c,2}$

The green dotted circle represents the execution of action $af_{o2c,1}$, i.e., alerting the sales manager. For each occurrence of the late response, the action was properly executed. For instance, on 5.12.2020 at 9 a.m., for the five inquiries reported for their late responses, alerting messages were sent to the corresponding sales manager. The lower part of the chart describes the ratio of late responses compared to legal responses. As defined in $af_{o2c,2}$, we assign additional resources to handle the late responses when the ratio is higher than 10 percent. For instance, on 5.12.2020 at 9 a.m., the ratio was 12 percent, thus an additional resource being dispatched to handle the late responses.

Table 2 describes the mean, median, and standard deviation of response times on each day before and after taking management actions. The mean response time decreases after sending alerts to sales managers and assigning more resources to handle delayed responses. For instance, the mean response time to the inquiries on 02. Dec. is 30.10 hours, whereas the mean response time on 07. Dec. is 23.35 after taking the actions. In addition, the standard deviation remarkably decreases after the actions, showing the efficiency of the actions in handling the inquiries taking a long time without mitigating measures. For instance, the standard deviation of the response times on 04. Dec. is 17.88, whereas the one on 07. Dec. is 1.25, representing that the long-delayed responses are prevented by the actions.

Figure 16 depicts the 53 orders that had changes from 9 a.m., 8.12.2020 to 9 a.m., 10.12.2020. Each row (yellow line) in the upper chart of the figure represents the lifecycle of each order, and the red circle indicates the occurrence of the change activity in the order. Note that the orders were under processing for the three days and only change activities in the orders are highlighted. The gray vertical lines represent the points in time when the constraint monitor evaluates constraint $cf_{o2c,2}$ and the action engine produces the actions based on $af_{o2c,3}$ and $af_{o2c,4}$.

The red square box represents the time when the constraint instances violating $cf_{o2c,2}$ are produced. For each detection of the violation, a notification is sent to the customer (as shown in the dotted green circle) to warn of the possible delay of the order in case of frequent changes in the order. For instance, on 8.12.2020 at 9 a.m., 23 notifications were sent to the corresponding customers. Since the frequency of the changes in the last 24 hours at 9 a.m., 8.12.2020 is higher than 20, the new activity, i.e., *approve by a sales manager*, is added to the control flow of the process for the next 24 hours.

Before taking mitigating actions, 18 percent of all changed orders entail multiple changes in their specification, thus generating delays in the delivery. After sending notifications to customers and adding new activities in the control flow of the business process, only four orders out of 53 orders reporting changes (i.e., 7.5 percent) showed multiple changes, validating the effectiveness of the actions in preventing possible additional changes.

Table 2 Statistics of the response times (hours) to inquiries in each day (*before* and *after* taking management actions)

Before				After		
02.Dec.	03.Dec.	04.Dec.	*Date*	05.Dec.	06.Dec.	07.Dec.
30.10	27.05	30.36	*Mean*	24.48	23.88	23.35
34.00	29.00	27.00	*Med.*	23.00	23.00	23.00
13.93	14.89	17.88	*Std.*	2.73	3.53	1.25

7 Discussion

In this section, we explain academic and practical implications of this work. Next, we discuss limitations and future work.

7.1 Implications

This paper suggests the starting point for a new branch of research in process mining discipline. The action-oriented process mining bridges the gap between the valuable insights from process mining techniques and the needed management actions to improve business processes. The proposed general frame-

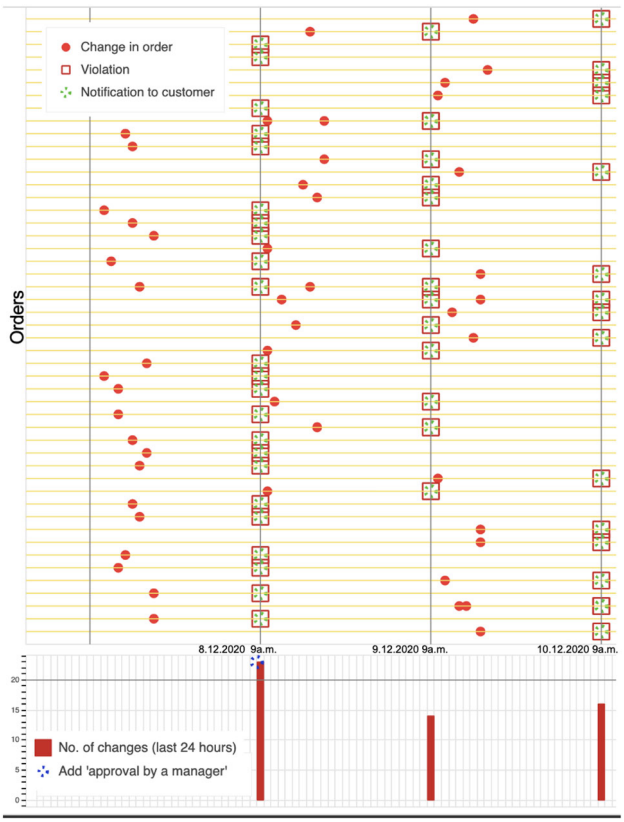

Fig. 16 The upper part of the chart describes 53 orders and the evaluation results by the constraint monitor using $cf_{o2c,2}$ and the actions by the action engine using $af_{o2c,3}$. The chart on the lower side describes the frequency of the changes in the last 24 hours and the generation of actions by the action engine using $af_{o2c,4}$

work incorporates the traditional process mining techniques for diagnostics and process monitoring and connects them to the generation of proactive actions.

Our work has important implications in both academic and practical standpoints. From an academic research standpoint, this work provides the formal architecture upon which novel techniques in business process monitoring can be deployed for the ultimate goal of the process improvement. For instance, techniques for predictive business process monitoring can be deployed using the framework for the continuous monitoring of the target business processes. Moreover, the proposed framework provides the foundation for developing novel approaches to generate proactive actions. In this work, we suggest the cube-based action engine as an instantiation of the action engine. New approaches can be developed to produce relevant management actions by analyzing the constraint instance stream and deployed to the action engine to enable the continuous management of business processes.

Furthermore, the constraint cube facilitates the development of methods to analyze constraint instances. Each cell in the constraint cube contains the constraint instances that contain attributes (e.g., context). Along with the aggregation measures, such as sum, count, and average, more advanced techniques (e.g., statistical analysis and machine learning) can be applied to find meaningful patterns in the constraint instances, enabling to generate more effective management actions.

Besides, in real-life business processes, it is common to observe variations and changes in process behavior, i.e., concept drifts [47]. For instance, due to Covid-19 pandemic, organizations augment their business processes, resulting in varying process behavior. In order to implement effective techniques for action-oriented process mining, we need to take such concept drifts into account. In particular, the definition of constraints and actions should be adapted accordingly such that they remain relevant in a specific situation. To this end, opportunities exist for fellow scholars to connect the detection, characterization, and explanation of concept drifts in business processes [48–50] to the dynamic and adaptive definition of constraints and actions. A possible direction is to deploy incremental learning and model adaptation [51] to ensure that the definition of constraints and actions are up-to-date with respect to evolving business processes.

In terms of implications for practice, our proposed framework allows practitioners to incorporate diverse monitoring techniques in the organization into the constraint monitor and maintain the monitoring results in the consistent format of the constraint instance. The single source of monitoring results is continuously analyzed and utilized to generate necessary management actions. The framework also provides the general structure based upon which practitioners can define domain-specific definitions of management actions and their continuous deployments. This continuous management of business processes using the constraint monitor and action engine enables them to maintain competitive advantages in fast-evolving and dynamic business environments by continuously improving the process.

7.2 Limitations and future work

This work has some limitations. Firstly, even though the proposed framework was evaluated using the real-life information system (i.e., SAP ERP system), it was not evaluated with the real-life human resources in the organization. Indeed, it is essential to evaluate the effects of actions in the real-life organization to take diverse influencing factors into account. For instance, as described in [52], business processes are affected by social factors such as organizational frictions. The action of adding a new resource to a task, in principle, is expected to increase the productivity of the task. However, considering the possible organizational frictions of the action, it can also result in negative impacts on productivity.

Second, the proposed framework does not provide a feedback mechanism. The actions produced from the action engine may have undesired impacts on business processes or even have conflicts among them. For instance, in a healthcare process, two actions may coexist where one of them assigns a patient to in-depth tests, whereas the other asks for a doctor to provide immediate reports to the patient. To deal with this conflict, a method to provide feedback on the possible conflicts is required, e.g., using what-if analysis based on discrete event simulation techniques to recommend valid actions, conducting A/B testing to evaluate the efficiency of actions, and optimizing the generation of actions using reinforcement learning. Next, in this work, we do not provide the set of common and effective actions in business processes for process improvement, rather focusing on providing the framework to produce the actions in an efficient manner with the given definitions of actions by domain experts. The taxonomy of generic management actions will help the elicitation of the available management actions in different organizations, facilitating the adoption of the proposed framework.

Finally, the proposed framework defines the outcome of the constraint instances as a binary value, i.e., OK and NOK. However, it is practically more relevant to have multiple different states of the outcome such as potentially violated and potentially non-violated. It will enable a more abundant analysis of constraint instances, leading to the generation of more effective management actions. Moreover, a constraint instance may also include attributes such as priorities to promote the efficient analysis. For instance, considering that the violation of the constraint with higher priority is more problematic to business processes, one may define an action formula to take the priority into account when analyzing the necessity of actions.

As future work, we plan to validate the effectiveness of the proposed framework by conducting case studies with real-life organizations. The experiments with the real-life organization will provide valuable insights to improve the framework and facilitate the deployment of the framework to organizations. Another important direction of future work is to develop

a feedback mechanism to proactively evaluate the effects of management actions and resolve conflicts among them. Moreover, it will empower the framework to be adaptive to dynamic and complex business environments. We also plan to develop action patterns that describe recurring problems in business processes and effective actions to deal with them. The action patterns are transformed into constraint formulas and action formulas to support the continuous management of business processes.

8 Conclusion

In this paper, we proposed the general framework for action-oriented process mining, which continuously transforms process diagnostics into proactive actions for 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. Also, we suggested a concrete instantiation of the action engine using the constraint cube. The constraint cube stores a continuous stream of constraint instances in a multi-dimensional structure, and the cube-based action engine analyzes them at a proper abstraction level to produce necessary actions.

The framework is implemented as a *ProM* plug-in and evaluated on an artificial information system and a real-life SAP ERP system. The experiments show the effectiveness of our proposed framework to monitor business processes and provide timely corrective actions to improve the process by removing existing and potential threats in the process and enhancing performances.

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

Funding Open Access funding enabled and organized by Projekt DEAL. This research is funded by the Alexander von Humboldt(AvH) Stiftung.

Availability of data and material The proposed framework is implemented as a plug-in of *ProM*, an open-source framework for the implementation of process mining tools. It is available in a package named *ActionOrientedProcessMining* in the nightly build of *ProM*. The user manual is available via https://github.com/gyunamister/ActionOrientedProcessMining/blob/master/README.md, along with the simulated data used for the evaluation.

Declarations

Conflicts of interest The authors declare no conflict of interest.

Code availability The source code for the implementation is publicly available both via the SVN repository (https://svn.win.tue.nl/repos/prom/Packages/ActionOrientedProcessMining) and Github repository (https://github.com/gyunamister/ActionOrientedProcessMining).

References

1. van der Aalst, W.M.P.: Academic View: Development of the Process Mining Discipline, pp. 181–196. Springer International Publishing, Cham (2020)
2. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer, Berlin (2018)
3. van der Wil, M.P.: Aalst. Data Science in Action. In Process Mining. Springer, Heidelberg (2016)
4. van Zelst, S.J., Bolt, A., Hassani, M., van Dongen, B.F., van der Aalst, W.M.P.: Online conformance checking: relating event streams to process models using prefix-alignments. Int. J. Data Sci. Anal. **8**(3), 269–284 (2019)
5. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: An approach based on colored automata. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) Business Process Management, vol. 6896, pp. 132–147. Springer, Berlin (2011)
6. Ramezani, E., Fahland, D., van der Aalst, W.M.P.: Where did i misbehave? Diagnostic information in compliance checking. In: Hutchison, D., Kanade, T., et al. (eds.) Business Process Management. volume 7481, pp. 262–278. Springer, Heidelberg (2012)
7. Schelter, S., Lange, D., Schmidt, P., Celikel, M., Biessmann, F., Grafberger, A.: Automating large-scale data quality verification. Proc. VLDB Endow. **11**(12), 1781–1794 (2018)
8. Park, G., van der Aalst, W.M.P.: A general framework for action-oriented process mining. In: Adela D.R.O., Henrik, L., Flávia M.S. (eds), *Business Process Management Workshops*, vol. 397, pp. 206–218. Springer International Publishing (2020)
9. Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Maggi, F. M., Marrella, A., Soo, A.: Automated discovery of process models from event logs: review and benchmark. arXiv:1705.02288 [cs] (2018)
10. Carmona, J., van Dongen, B., Solti, A., Matthias, W.: Relating processes and models. Springer International Publishing, Conformance Checking (2018)
11. Denisov, V., Fahland, D., van der Aalst, W.M.P.: Unbiased, fine-grained description of processes performance from event data. In: Mathias, W., Marco, M., Ingo, W. (eds) Business Process Management, vol. 11080, pp. 139–157. Springer International Publishing (2018)
12. Burattin, A., Sperduti, A., Veluscek, M.: Business models enhancement through discovery of roles. In: 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), pp. 103–110. IEEE (2013)
13. Suriadi, S., Ouyang, C., van der Aalst, W.M.P., ter Hofstede, A.H.: Root cause analysis with enriched process logs. In: La Rosa, M., Soffer, P. (eds.) Business Process Management Workshops, vol. 132, pp. 174–186. Springer, Berlin (2013)
14. Burattin, A., van Zelst, S.J., Armas-Cervantes, A., Dongen, B. F. V., Carmona, J.: Online conformance checking using behavioural pat-

terns. In: Mathias, W., Marco, M., Ingo, W., Jan, B. (eds) *Business Process Management*, vol. 11080, pp. 250–267. Springer International Publishing (2018)
15. Weidlich, M., Ziekow, H., Mendling, J., Günther, O., Weske, M., Desai, N.: Event-based monitoring of process execution violations. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) Business Process Management. volume 6896, pp. 182–198. Springer, Berlin (2011)
16. Awad, A., Barnawi, A., Elgammal, A., Elshawi, R., Almalaise, A., Sakr, S.: Runtime detection of business process compliance violations: an approach based on anti patterns. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing, pp. 1203–1210. ACM (2015)
17. Maggi, F.M., Westergaard, M., Montali, M., van der Aalst, W.M.P.: Runtime verification of LTL-based declarative process models. In: Khurshid, S., Sen, K. (eds.) Runtime Verification, vol. 7186, pp. 131–146. Springer, Berlin (2012)
18. Accorsi, R., Lowis, L., Sato, Y.: Automated certification for compliant cloud-based business processes. Bus. Inf. Syst. Eng. **3**(3), 145–154 (2011)
19. Bezerra, F., Wainer, J.: Algorithms for anomaly detection of traces in logs of process aware information systems. Inf. Syst. **38**(1), 33–44 (2013)
20. 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, pp. 150–159. Springer, Heidelberg (2008)
21. Li, G., van der Aalst, W.M.P.: A framework for detecting deviations in complex event logs. Intell. Data Anal. **21**(4), 759–779 (2017)
22. de Leoni, M., . van der Aalst, W.M.P.: A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Inf. Syst. **56**, 235–257 (2016)
23. Polato, M., Sperduti, A., Burattin, A., de Leoni, M.: Time and activity sequence prediction of business process instances. Computing **100**(9), 1005–1031 (2018)
24. Senderovich, A., Francescomarino, C.D., Maggi, F.M.: From knowledge-driven to data-driven inter-case feature encoding in predictive process monitoring. Inf. Syst. **84**, 255–264 (2019)
25. Francescomarino, C.D., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. IEEE Trans. Serv. Comput. **12**(6), 896–909 (2019)
26. Teinemaa, I., Dumas, M., Maggi, F.M., Francescomarino, C. D.: Predictive business process monitoring with structured and unstructured data. In: Marcello, L.R., Peter, L., Oscar, P. (eds) *Business Process Management*, vol. 9850, pp. 401–417. Springer International Publishing (2016)
27. Le, M., Nauck, D., Gabrys, B., Martin, T.: Sequential clustering for event sequences and its impact on next process step prediction. In: Anne, L., Olivier, S., Bernadette, B.-M., Ronald, R.Y. (eds.) Information Processing and Management of Uncertainty in Knowledge-Based Systems, vol. 442, pp. 168–178. Springer International Publishing, Berlin (2014)
28. Lakshmanan, G.T., Shamsi, D., Doganata, Y.N., Unuvar, M., Khalaf, R.: A markov prediction model for data-driven semi-structured business processes. Knowl. Inf. Syst. **42**(1), 97–126 (2015)
29. Breuker, D., Matzner, M., Delfmann, P., Becker, J.: Comprehensible predictive models for business processes. MIS Q. **40**(4), 1009–1034 (2016)
30. Márquez-Chamorro, A.E., Resinas, M., Antonio, R.-C., Miguel, T.: Run-time prediction of business process indicators using evolutionary decision rules. Expert Syst. Appl. **87**, 1–14 (2017)
31. Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortes, A.: Predictive monitoring of business processes: a survey. IEEE Trans. Serv. Comput. **11**(6), 962–977 (2018)
32. Evermann, J., Rehse, J.-R., Fettke, P.: Predicting process behaviour using deep learning. Decis. Supp. Syst. **100**, 129–140 (2017)
33. Tax, N., Verenich, I., Rosa, M. L., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Eric, D., Klaus, P. (eds) Advanced Information Systems Engineering, vol. 10253, pp. 477–492. Springer International Publishing (2017)
34. Mehdiyev, N., Evermann, J., Fettke, P.: A novel business process prediction model using a deep learning method. Bus. Inf. Syst. Eng. **62**(2), 143–157 (2020)
35. Galanti, R., Coma-Puig, B., Leoni, M.D., Carmona, J., Navarin, N.: Explainable predictive process monitoring. In: 2020 2nd International Conference on Process Mining (ICPM), pp. 1–8. IEEE (2020)
36. Conforti, R., de Leoni, M., La, R., Marcello, A., van der Aalst, W.M.P., Hofstede, A.H.M.: A recommendation system for predicting risks across multiple business process instances. Decis. Supp. Syst. **69**, 1–19 (2015)
37. Fahrenkrog-Petersen, S.A., Tax, N., Teinemaa, I., Dumas, M., Leoni, M.D., Maggi, F.M., Weidlich, M.: Fire now, fire later: alarm-based systems for prescriptive process monitoring. arXiv:1905.09568 [cs, stat] (2019)
38. Weinzierl, S., Dunzer, S., Zilker, S., Matzner, M.: Prescriptive business process monitoring for recommending next best actions. In: Dirk, F., Chiara, G., Jörg, B., Marlon, D. (eds.) Business Process Management Forum, pp. 193–209. Springer International Publishing, Cham (2020)
39. Dees, M., de Leoni, M., van der Aalst, W.M.P., Reijers, H.A.: What if process predictions are not followed by good recommendations? In: Jan, B., Jan, M., Michael, R. (eds) *Proceedings of the Industry Forum at BPM 2019*, vol. 2428, pp. 61–72. CEUR-WS.org (2019)
40. de Leoni, M., Dees, M., Reulink L.: Design and evaluation of a process-aware recommender system based on prescriptive analytics. In: 2020 2nd International Conference on Process Mining (ICPM), pp. 9–16 (2020)
41. 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, pp. 28–31, Aachen, Germany (2019)
42. Park, G., van der Aalst, W.M.P.: Realizing a digital twin of an organization using action-oriented process mining. In: 2021 3rd International Conference on Process Mining (ICPM), pp. 104–111 (2021)
43. van der Aalst, W.M.P.: Object-centric process mining: dealing with divergence and convergence in event data. In: Peter, C.Ö., Gwen, S. (eds.) Software Engineering and Formal Methods, vol. 11724, pp. 3–25. Springer International Publishing, Cham (2019)
44. Ghahfarokhi, A.F., Park, G., Berti, A., van der Aalst, W.M.P.: A standard for object-centric event logs. In: Ladjel, B., Marlon, D., Panagiotis, K., Raimundas, M., Ahmed, A., Matthias, W., Mirjana, I., Olaf, H. (eds.) New Trends in Database and Information Systems, pp. 169–175. Springer International Publishing, Cham (2021)
45. Park, G., van der Aalst, W.M.P.: Towards reliable business process simulation: A framework to integrate erp systems. In: Adriano, A., Asif, G., Selmin, N., Iris, R.-B., Rainer, S., Jelena, Z. (eds.) Enterprise, Business-Process and Information Systems Modeling, pp. 112–127. Springer International Publishing, Cham (2021)
46. van der Aalst, W.M.P.: Aalst and Alessandro Berti. Discovering object-centric petri nets. Fundamenta Informaticae **175**(1), 1–40 (2020)
47. Burattin, A.: Process Mining for Stream Data Sources, pp. 177–204. Springer International Publishing, Cham (2015)
48. Carmona, J., Ricard, G.: Online techniques for dealing with concept drift in process mining. In: Jaakko, H., Frank, K., Allan, T. (eds.) Adva. Intell. Data Anal. XI, pp. 90–102. Springer, Berlin (2012)
49. Omori, N.J., Tavares, G.M., Ceravolo, P., Barbon Jr, S.: Comparing concept drift detection with process mining tools. In: Proceedings of the XV Brazilian Symposium on Information Systems, SBSI'19, New York, NY, USA, 2019. Association for Computing Machinery

50. Adams, J.N., van Zelst, S.J., Quack, L., Hausmann, K., van der Aalst, W.M.P., Rose, T.: A framework for explainable concept drift detection in process mining. In: Artem, P., Moe, T.W., Amy, V.L., Manfred, R. (eds.) Business Process Management, pp. 400–416. Springer International Publishing, Cham (2021)
51. Maisenbacher, M., Weidlich, M.: Handling concept drift in predictive process monitoring. In: 2017 IEEE International Conference on Services Computing (SCC), pp. 1–8 (2017)
52. van der Aalst, W.M.P., Dustdar, S.: Process mining put into context. IEEE Internet Comput. **16**(1), 82–86 (2012)