Realizing A Digital Twin of An Organization Using Action-oriented Process Mining

Gyunam Park and Wil M.P. van der Aalst
Process and Data Science Group (PADS)
Department of Computer Science, RWTH Aachen University, Aachen, Germany
{gnpark, wvdaalst}@pads.rwth-aachen.de

***Abstract*—A Digital Twin of an Organization (DTO) is a mirrored representation of an organization, aiming to improve the business process of the organization by providing a transparent view over the process and automating management actions to deal with existing and potential risks. Unlike wide applications of digital twins to product design and predictive maintenance, no concrete realizations of DTOs for business process improvement have been studied. In this work, we aim to realize DTOs using action-oriented process mining, a collection of techniques to evaluate violations of *constraints* and produce the required *actions*. To this end, we suggest a *digital twin interface model* as a transparent representation of an organization describing the current state of business processes and possible configurations in underlying information systems. By interacting with the representation, process analysts can elicit constraints and actions that will be continuously monitored and triggered by an *action engine* to improve business processes. We have implemented a web service to support it and evaluated the feasibility of the proposed approach by conducting a case study using an artificial information system supporting an order handling process. *Keywords*—Digital Twin, Action-Oriented Process Mining, Process Improvement, Hybrid Intelligence, Action Patterns**

I. Introduction

Organizations are required to continuously improve their business processes to react to dynamically changing situations in business environments [1]. While the necessity of continuous process improvement is hardly questioned, it is non-trivial to effectively analyze business processes and identify improvement points due to the complexity of business processes and information systems supporting them [2]. For instance, a standard business process of an organization supported by an SAP ERP system (e.g., Order-to-Cash and Procure-to-Pay) may contain 80,000 variants of process instances, making analysis of the process challenging.

A digital twin is a digitally mirrored, transparent representation of a complex real-life object or process [3]. A Digital Twin of an Organization (DTO) is a digital twin extended to the whole organization where such objects and processes belong [4]. DTOs are used to facilitate analysis of business processes in the organization and continuously improve the process with the automated execution of management actions. Despite its tremendous benefits for continuous process improvement, a concrete realization and implementation of DTOs are missing both in research and in practice [5].

In this paper, we aim to realize DTOs using action-oriented process mining, a collection of techniques to continuously monitor violations of *constraints* in operational processes and automatically trigger *actions* based on the monitoring results in such a way that they update the configuration of information systems [6]. If delays (are expected to) happen for approving orders (i.e., constraint), the approval can be temporarily skipped for upcoming orders by adjusting the system configuration (i.e., action). The techniques include conformance checking [7] and predictive process monitoring [8].

To that end, we suggest a *digital twin interface model* as a representation of a business process and its supporting information systems in an organization. The representation provides transparent views of the current state of the process (e.g., a bottleneck in an activity) and possible configurations in the system (e.g., skipping the activity). Interacting with it, process analysts can elicit constraints by analyzing the state of business processes and define actions based on the possible configurations. Subsequently, an *action engine* continuously monitors the constraints and automates corresponding actions.

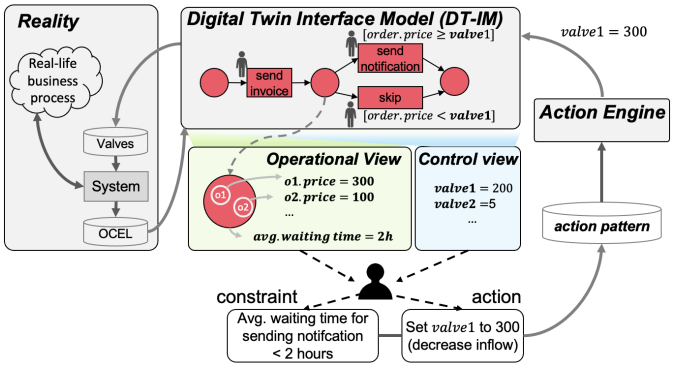

Figure 1: An overview of the approach to realize DTOs based on action-oriented process mining

Fig. 1 describes the realization more in detail. A digital twin interface model replicates business processes supported by information systems, using Object-Centric Petri Nets (OCPN) [9] as formal representations. Event data, recorded by the system during the execution of the process (e.g., Object-Centric Event Log (OCEL) [10], are used to construct a digital twin interface model along with some expert knowledge. An operational view describes the current status of the process, being updated using streaming event data. It describes which objects reside in which phase of the process (e.g., *o1* and *o2* waiting for *send notification*) and various diagnostics (e.g., the average waiting time for two days is 2 hours). A control view specifies the current configuration of the system. For instance, *valve1*

is currently set to 200. Based on the operational and control view, process analysts can define *action patterns* specifying the common pattern between constraints and actions. Using the pattern, an *action engine* monitors the process and produces necessary actions that update the configuration of the system.

In order to evaluate the feasibility of realizing DTOs, we have implemented a web application to support the digital twin interface model and action engine. In addition, we have tested the implementation with an artificial information system that supports an order handling process. The details and sources of the implementation and the information system are publicly available via https://github.com/gyunamister/dtween.

The remainder is organized as follows. We discuss the related work in Sec. II. Next, we present the preliminaries in Sec. III. In Sec. IV, we define a digital twin interface model along with its operational and control views. Next, Sec. V presents action patterns and an action engine. Afterward, Sec. VI introduces the implementation of the web application and Sec. VII provides a case study based on a simulated business process. Finally, Sec. VIII concludes the paper.

II. Related Work

A. Digital Twin

A digital twin is a digital replication of a real system, e.g., a production process or even a whole organization. Conceptual frameworks of various digital twins have been widely studied. In [11], a conceptual framework for product design, product manufacturing, and product service has been suggested, while [12] providing a conceptual model of digital twins for smart manufacturing service. [4] and [5] propose conceptual frameworks of DTOs by extending a conventional digital twin in a manufacturing domain to an entire organization.

Contrary to various conceptualizations of digital twins in extensive fields, concrete realizations only exist for product designs, product developments, and productions, which focus on digitalizing physical assets using IoT sensors [4]. For instance, NASA applies the digital twin in developing vehicles to predict the future status of vehicles using simulation models [3]. However, applications of DTOs for continuous improvements of business processes have not been studied. In this work, we realize DTOs aiming at improving business processes by providing transparent views to the process and automating necessary actions.

B. Action-Oriented Process Mining

We deploy techniques for action-oriented process mining to realize DTOs, including 1) ones for monitoring violations of constraints and 2) ones for triggering actions based on monitoring results. First, extensive literature exists for monitoring techniques [7]. [13] suggests a technique to monitor (temporarily) satisfied, (temporarily) violated constraints based on Linear Temporal Logic (LTL) and colored automata. In [14], constraints are formulated into Petri net patterns, which are then evaluated by alignment-based conformance checking.

Predictive monitoring techniques enable violations to be detected before they actually happen [8]. In [15], a scenario-based predictive approach is proposed to predict the future behavior of a business process by deploying system dynamics. Recently, techniques based on deep neural networks have been widely studied, e.g., [16] suggests a predictive method based on Long-Short Term Memory (LSTM) networks.

Next, several techniques have been proposed to automatically generate specific types of actions to improve business processes. In [17], the resource allocation is optimized by predicting the risk of process instances. In [18], a prescriptive alarm system has been proposed to generate alarms by computing trade-offs among different interventions.

Instead of focusing on a specific type of action, [19] provides support to turn diagnostics into extensive actions. It generates signals by analyzing event data and executes different actions corresponding to the signals to source systems. In [6], a more systematic approach is proposed to transform the process-centric diagnostics into management actions.

III. Preliminaries

In this work, we use object-centric Petri nets as a core formal representation of a digital twin interface model. First, a Petri net is a directed graph having places and transitions as nodes, and flow relations as edges. A labeled Petri net is a Petri net where the transitions are labeled.

Definition 1 (Labeled Petri Net)**.** Let $\mathbb{U}_{act}$ be the universe of activity names. A labeled Petri net is a tuple $N{=}(P, T, F, l)$ with P the set of places, T the set of transitions, $P \cap T{=}\emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ the flow relation, and $l \in T \not\rightarrow \mathbb{U}_{act}$ a labeling function.

A marking $M_N \in \mathcal{B}(P)$ represents the state of a Petri net as a multiset of places. A transition $t \in T$ is enabled in marking M_N if its input places contain at least one token, and it may *fire* by removing one token from each of the input places and producing one token for each of the output places.

In an OCPN, each place is associated with an object type, enabling it to represent interactions among different object types. Moreover, variable arcs are used to represent the consumption/production of a variable amount of tokens in one step.

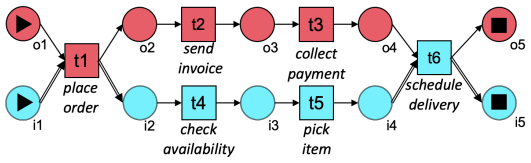

Figure 2: An example of object-centric Petri nets

Definition 2 (Object-Centric Petri Net)**.** Let $\mathbb{U}_{ot}$ be the universe of object types. An *object-centric Petri net* is a tuple $ON{=}(N, pt, F_{var})$ where $N{=}(P, T, F, l)$ is a labeled Petri net, $pt \in P \rightarrow \mathbb{U}_{ot}$ maps places onto object types, and $F_{var} \subseteq F$ is the subset of variable arcs.

Fig. 2 shows an OCPN: $P{=}\{o1, \ldots, o5, i1, \ldots, i5\}$, $T{=}\{t1, t2, \ldots, t6\}$, $F{=}\{(o1, t1), (i1, t1), (o2, t2), \ldots\}$, $l(t1){=}place\ \ order$, $l(t2){=}send\ \ invoice$, etc., $pt(o1){=}Order$, $pt(i1){=}Item$, etc., and $F_{var}{=}\{(i1, t1), (t1, i2), (i4, t6), (t6, i5)\}$, where $Order, Item \in \mathbb{U}_{ot}$.

A marking represents the state of an OCPN.

Definition 3 (Marking)**.** Let $\mathbb{U}_{oi}$ be the universe of object identifiers. Let $ON{=}(N, pt, F_{var})$ be an object-centric Petri net, where $N{=}(P, T, F, l)$. $otyp \in \mathbb{U}_{oi} \to \mathbb{U}_{ot}$ assigns object types to object identifiers. $Q_{ON}{=}\{(p, oi) \in P \times \mathbb{U}_{oi} \mid otyp(oi){=}pt(p)\}$ is the set of possible tokens. A marking M of ON is a multiset of tokens, i.e., $M \in \mathcal{B}(Q_{ON})$.

For instance, marking $M1{=}[(o1, 0092), (i1, 10085), (i1, 10086), (i1, 10087)]$ denotes four tokens among which place *o1* has one token referring to object *0092* and $i1$ has three tokens referring to objects *10085*, *10086*, and *10087*.

The concept of binding is used to explain the semantics of an OCPN. A binding describes the execution of a transition consuming objects from its input places and producing objects for its output places. A binding (t, b) is a tuple of transition t and function b mapping the object types of the surrounding places to sets of object identifiers. For instance, $(t1, b1)$ describes the execution of transition $t1$ with $b1$ where $b1(Order){=}\{0092\}$ and $b1(Item){=}\{10085, 10086, 10087\}$, where *Order* and *Item* are the object types of the surrounding places of $t1$ (i.e., $o1, i1, o2, i2$).

A binding (t, b) is *enabled* in marking M if all the objects specified by b exist in the input places of t. For instance, $(t1, b1)$ is enabled in marking $M1$ since 0092, 10085, 10086, and 10087 exist in its input places, i.e., $o1, i1$.

If $t1$ *fires* with $(t1, b1)$, *0092* is removed from $o1$ and added to $o2$. Besides, 10085, 10086, and 10087 are removed from $i1$ and added to $i2$, resulting in new marking $M2{=}[(o2, 0092), (i2, 10085), (i2, 10086), (i2, 10087)]$. For more details of OCPNs, we refer readers to [9].

IV. Digital Twin Interface Model

In this section, we first introduce a Digital Twin Interface Model (DT-IM) with its core components. Next, we explain an operational view of the DT-IM describing the current states of business processes. Afterward, we introduce a control view of the DT-IM depicting the current configurations and possible actions defined over target information systems.

A DT-IM represents process behaviors using OCPNs as formal representations. Moreover, it describes the routing/resource allocation rules of the process using *guards*. A guard is a formula defined over attributes using relational operators ($\leq$,$\geq$,$=$) as well as logical operators such as conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$). Associated with a transition, a guard extends the semantics of OCPNs by allowing the transition to fire not only if the transition is enabled in marking M, but also if the guard evaluates to $true$. We denote with $F(X)$ the set of such formulas defined over a set X of attributes. *Valves* are configuration and options in information systems and can be used in guards of the OCPN (e.g., minimum threshold to send notifications).

Definition 4 (Digital Twin Interface Model (DT-IM))**.** Let $\mathbb{U}_{attr}$ be the universe of attribute names. A digital twin interface model, denoted as DT, is a tuple (ON, V, G) where

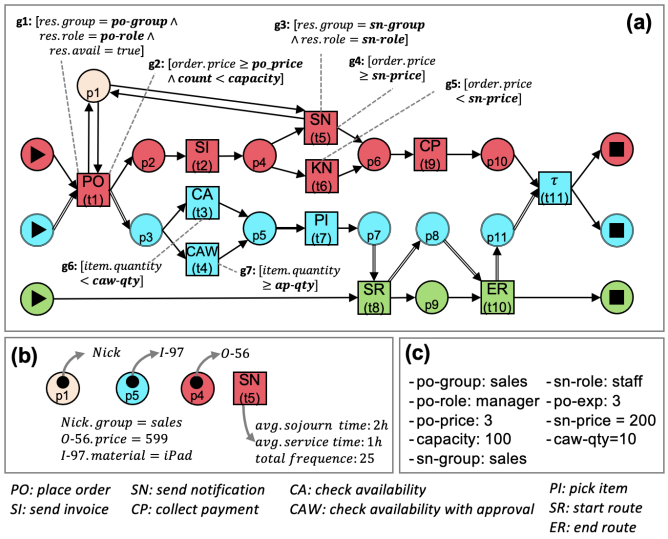

Figure 3: (a) a DT-IM replicating an order handling process (b) an operational state of the DT-IM (c) a configuration of the DT-IM

- $ON{=}(N, pt, F_{var})$ is an object-centric Petri net, where $N = (P, T, F, l)$,
- $V \subseteq \mathbb{U}_{attr}$ is a set of valves, and
- $G \in T \to (F(\mathbb{U}_{attr}) \cup \{true\})$ associates transitions with guards.

Fig. 3(a) is an example of DT-IMs (DT_1) replicating the information system that supports an order handling process. The OCPN in Fig. 3(a) describes the control flow of the order handling process where four different types of objects (i.e., *order* in red, *item* in blue, *route* in green, and *resource* in ivory) interact. Note that some arcs connecting $p1$ and transactions, e.g., $(p1, t2)$ and $(t2, p1)$, are omitted for better representations.

DT_1 uses valves, such as *po-group*, *po-price*, and *capacity* to define guards. For instance, *g1* describes a rule that a resource from the *sales* department with the role of valve *po-group* is eligible to perform the activity. *g2* describes that only orders with a price higher than or equal to valve *po-price* can be further processed if the current capacity is below valve *capacity*.

Conceptually, a guard $G(t)$ may consist of two parts: one formulating the resource allocation rules, $G_{res}(t)$, and the other formulating the routing rules, $G_{rout}(t)$, i.e., $G(t){=}G_{res}(t) \wedge G_{rout}(t)$. In Fig. 3(a), *g1* describes a resource allocation rule for *place order*, whereas *g2* describes the routing rule. The resource allocation and routing rules form guard $G(t1)$, i.e., $G(t1){=}g1 \wedge g2$. In other words, $t1$, when enabled at marking M, fires only if the corresponding resource is from *po-group*, the price is higher than *po-price*, and the capacity of the process is below *capacity*.

A. Operational View of A Digital Twin Interface Model

Using DT-IMs, we can describe the current operational state of business processes. The state is represented by *marking* and *diagnostics*. A marking represents which objects reside in which parts of business processes using OCPNs, while

diagnostics represent performance/compliance of business processes, e.g., average waiting time for sending notifications in the last two days (*avg-wt-sn-2d*). The diagnostics are computed using data-driven, model-based analysis such as token-based replay [20]. Many diagnostics are available in a DT-IM, but we leave them deliberately vague, denoting Δ_{DT} to be the set of all possible diagnostics of digital twin interface model DT.

Definition 5 (Operational State of A DT-IM)**.** Let $\mathbb{U}_{val}$ be the universe of attribute values. Let $\mathbb{U}_{vmap}=\mathbb{U}_{attr} \nrightarrow \mathbb{U}_{val}$ be the set of all partial functions mapping a subset of attribute names onto the corresponding values. Let $DT=(ON, V, G)$ be a DT-IM, where $ON=(N, pt, F_{var})$. An operational state of DT is a tuple $OS=(M, ovmap, dmap)$ where

- $M \in \mathcal{B}(Q_{ON})$ is a marking of ON,
- $ovmap \in O \rightarrow \mathbb{U}_{vmap}$ is an object value assignment where $O=\{oi|(p, oi) \in M\}$, and
- $dmap \in \Delta_{DT} \nrightarrow \mathbb{R}$ is a diagnostics assignment such that, for any $diag \in \Delta_{DT}$, $dmap(diag) = \perp$ if $diag \notin dom(dmap)$.

We denote Ω_{DT} to be the set of all possible states of DT.

Fig. 3(b) describes state $OS_1=(M_1, ovmap_1, dmap_1)$ where $M_1=[(p1, \textit{Nick}), (p4, \textit{O-56}), (p5, \textit{I-97})]$ denotes resource token *Nick*, order token *O-56*, and item token *I-97* residing in $p1$, $p4$ and $p5$, respectively. Besides, $ovmap_1(\textit{Nick})(\textit{group})=$ *sales*, $ovmap_1(\textit{O-56})(\textit{price})=599$, $ovmap_1(\textit{I-97})(\textit{material})=$ *iPad*, etc., and $dmap(\textit{avg-wt-sn-2d})=2$ *(hour)* where *avg-wt-sn-2d* $\in \Delta_{DT_1}$.

Operational states of a DT-IM change according to events from information systems. Here, we abstract from the mechanism to update states based on events. In Sec. VI, we introduce an implementation based on replaying events on OCPNs.

Definition 6 (Operation Engine)**.** Let $DT=(ON, V, G)$ be a DT-IM. An operation engine of DT, op_{DT}, updates the operational states of DT, i.e., $op_{DT} \in \Omega_{DT} \rightarrow \Omega_{DT}$.

For instance, an event of *send notification* about order *O-56* updates OS_1 to $OS_2=(M_2, ovmap_2, dmap_2) \in \Omega_{DT_1}$ where $M_2=[(p1, \textit{Nick}), (p6, \textit{O-56}), (p5, \textit{I-97})]$, $ovmap_2(\textit{O-56})(\textit{notified})=true$, and $dmap_2(\textit{avg-wt-sn-2d})=2.5$ *(hour)*.

B. Control View of A Digital Twin Interface Model

A DT-IM is also used to describe controls of the information system. The control is characterized by the value assignment of valves, called *configuration*. Note that the configuration does not enforce the behaviors of business processes since deviations may happen in business processes.

Definition 7 (Configuration)**.** Let $DT=(ON, V, G)$ be a DT-IM. A configuration $conf \in V \rightarrow \mathbb{U}_{val}$ assigns values to valves. Σ_{DT} is the set of all possible configurations of DT.

Fig. 3(c) describes the configuration, $conf_1 \in \Sigma_{DT_1}$ where $conf_1(\textit{sn-price})=200$, $conf_1(\textit{caw-qty})=10$, etc.

Definition 8 (Action)**.** Let $DT=(ON, V, G)$ be a DT-IM. An action $act \in \Sigma_{DT} \rightarrow \Sigma_{DT}$ updates the configuration. A_{DT} denotes the set of all possible actions defined over DT.

For instance, *skip-sn* $\in A_{DT_1}$ increases valve *sn-price* to infinity so that no orders go through *send notification*, i.e., $\textit{skip-sn}(conf_1)=conf_2$ where $conf_2 \in \Sigma_{DT_1}$ and $conf_2(\textit{sn-price})=\infty$.

V. Action Patterns and Action Engines

Using operational and control views of DT-IMs, process analysts can define *action patterns*. An action pattern describes the automated execution of actions to recurring problems in business processes. Each pattern consists of a *constraint* that specifies an undesired situation and a corresponding action to resolve the situation. A constraint is a formula defined over diagnostics using relational operators and logical operators.

Definition 9 (Action Pattern)**.** Let DT be a DT-IM. $\Pi_{DT}=F(\Delta_{DT}) \times A_{DT}$ is the set of all possible action patterns of DT.

Given DT_1 in Fig. 3, we can define action pattern ap_1 that skips *send-notification* if the average waiting time for the activity in the last two days is higher than 2 hours, i.e., $ap_1=([\textit{avg-wt-sn-2d}>2h], \textit{skip-sn}) \in \Pi_{DT_1}$ where *avg-wt-sn-2d* $\in \Delta_{DT_1}$ *skip-sn* $\in A_{DT_1}$.

Definition 10 (Action Engine)**.** Let $DT=(ON, V, G)$ be a DT-IM. An action engine $ae_{DT} \in (\Pi_{DT} \times \Omega_{DT} \times \Sigma_{DT}) \rightarrow A_{DT}$ generates actions given action patterns, operational states, and configurations, i.e., for any $ap=(constr, act) \in \Pi_{DT}$, $OS \in \Omega_{DT}$, and $conf \in \Sigma_{DT}$, $ae_{DT}(ap, OS, conf)=act$ if $constr$ evaluates to *true* w.r.t. OS and $conf$. $ae_{DT}(ap, OS, conf)= \perp$ otherwise.

In the remainder, we introduce the taxonomy of constraint violations and management actions. Note that our goal is not to provide completely exhaustive taxonomies, but to facilitate the elicitation of various action patterns by providing possible constraints defined over operational states of a DT-IM and actions based on its configuration.

A. A Taxonomy of Constraint Violations

Fig. 4 shows the taxonomy of constraint violations in UML class diagram. First, violations are categorized by their orientations: *compliance* and *performance* [21]. Compliance-oriented violations refer to the violations of rules during the execution of business processes. In the process shown in Fig. 3(a), skipping *place order* is considered as compliance-oriented violations. Performance-oriented violations concern undesired performances that are measured with various process performance metrics. The high average waiting time for the notification to customers in the order handling process is a performance-oriented diagnostic.

We further distinguish the compliance-oriented violations based on their types, i.e., existence and non-existence. The existence concerns unnecessary executions of activities in business processes, while the non-existence refers to skipping necessary executions of activities. In Fig. 3, notifications are sent for the orders higher than \$200. Sending notifications for orders lower than \$200 is concerned with the existence,

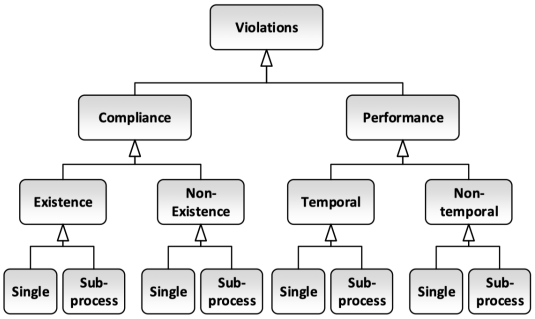

Figure 4: A taxonomy of constraint violations

whereas skipping such notifications for orders higher than $200 is related to the non-existence. Furthermore, the violations can consider either a single activity or a set of activities (i.e., sub-process). In Fig. 3, a violation of the existence can be defined not only for sending notifications, but also together with collecting payments.

Performance-oriented violations are further divided into temporal and non-temporal ones. The former involves absolute/relative time information in its definition, whereas the latter is not concerned with time information. For instance, the high average sojourn time for sending notifications relates to the time information, whereas the high frequency of sending notifications does not involve time information.

B. A Taxonomy of Management Actions

Fig. 5 shows the taxonomy of management actions to mitigate risks resulting from violations of constraints in UML class diagram. First, *inflow* controls the execution of certain activities in business processes by blocking or allowing the execution. In Fig. 3, we can block all executions of sending notifications by increasing the minimum threshold to infinity. Besides, we can block some of the executions by configuring the threshold to the value higher than the current one, e.g., $300. In contrast, we may allow the execution of activities by augmenting valves. Suppose that *ap-quantity* in Fig. 3(a) is set to 0, which means all items need approvals for checking availability. We can allow some skips by increasing *ap-quantity*.

Second, *routing* is to control the routing of objects in business processes, including *parallelism*, *extra*, *alternative*, and *skipping*. First, parallel routing is to parallelize the execution of activities. In case that a parallel execution exists in the process model (e.g., a choice between parallelizing *send invoice* and *send notification* and sequentially processing them as described in Fig. 3), we can adjust the routing to enable parallel executions for some/all objects. Next, extra routing is to execute extra activities for certain objects (e.g., further approval steps for specific types of items), while alternative routing is to adopt alternative control flows for certain objects (e.g., customer pick-ups of items without routes). Finally, skip routing is to deliberately bypass the execution of certain activities. For instance, we can skip the notification for the orders having a price higher than $200 by setting *sn-price* to 200 in Fig. 3.

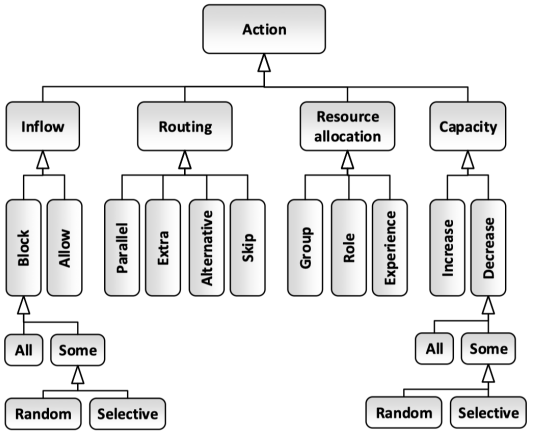

Figure 5: A taxonomy of management actions. Note that we only specify *All*/*Some* for *Block*/*Decrease*, but they exist for all the classes in the same level.

Third, resource allocation is to control the assignment of resources to activities based on different attributes of resources, e.g., *group*, *role*, and *experience*. First, we can change the resource allocation by adjusting group information. In Fig. 3, we can limit the execution of *place order* only by resources from sales department. Moreover, the assignment rule can be changed according to the role and experience of resources. In Fig. 3, by changing *po-role* to *staff*, the resource allocation for *place order* can be updated. Likewise, by increasing the level of *po-exp*, the allocation is accordingly updated.

Finally, one can control the capacity of business processes either by increasing or decreasing the maximum amount of process instances, e.g., manufacturing products, based on the situation of the process. For instance, we can increase the capacity of the order handling process by decreasing valve *capacity*. Contrary to that, we can decrease the capacity by increasing valve *capacity*.

VI. Implementation

A cloud-based web service[1] is implemented to support the digital twin interface model and action engine with a dedicated user interface. The implementation consists of four major functional components: (1) building DT-IMs, (2) updating states of DT-IMs, (3) visualizing DT-IMs, and (4) evaluating action patterns. All four functional components are implemented as Python packages. Being containerized as a Docker container, the functionality of the framework and its functional components is structured into a coherent set of microservices that are deployable on any platform.

In the following, we briefly explain each component. We refer readers to the tool manual[1] for more details.

A. Building Digital Twin Interface Models: This component aims at building DT-IMs given event data and user inputs. The main inputs of this component are event data, guards, and valves, whereas the output is a DT-IM. The event data are stored as standard OCEL format [10], while the guards

[1] sources & manuals available at https://github.com/gyunamister/dtween

and valves are stored as JSON-based formats[1]. The discovery technique introduced in [9] is deployed to discover an OCPN from the event data. User-provided guards and valves are used to enrich the discovered OCPN, completing a DT-IM.

B. Updating States of Digital Twin Interface Models: The goal of this component is to update states of a DT-IM accordingly to the updates of information systems. The main inputs include streaming event data in the standard OCEL format and a DT-IM built using the component in Sec. VI. The token-based replay [20] technique is used to compute diagnostics using the streaming event data in user-defined intervals (e.g., every 24 hours). Likewise, markings and object value assignments are computed, completing operational states of the DT-IM.

C. Visualizing Digital Twin Interface Models: This component provides visual information to users to facilitate the elicitation of action patterns. First, it provides the control view of a DT-IM, describing the routing and resource allocation rules and possible controls. Second, it provides the operational view showing the current states of a DT-IM. Fig. 6 is a screenshot of the web service showing the operational state. The OCPN describes the process model and the green box specifies the marking at place *order3* where four objects reside at 11:22:54, 17.June.2021. The table on the right hand side describes the object value assignment, e.g., object *o22* is an order having price $174. Using operational states in the operational view and configurations in the control view, users can define constraints and actions, which are used to compose action patterns.

D. Action Engine: This component aims to continuously evaluate action patterns by monitoring the business process. The main inputs include states of a DT-IM computed by the component in Sec. VI, current configurations, and user-defined action patterns. Based on the states, action engine evaluates the constraint specified in the action pattern and produce the corresponding action. The generated actions are applied to update the configuration of the DT-IM, which in turn is consumed by the target system to update its configuration.

VII. Proof of Concept

To evaluate the feasibility of realizing DTOs, we conduct a case study using an artificial information system that supports the order handling process introduced in Sec. IV. The implementation of the information system is publicly available[1] to be used for demonstrations. 14 resources are available in total at any point in time, each of them being responsible for multiple activities in the process. Orders are randomly placed and the resources process the orders and the corresponding items and routes based on *First-in-First-out* rule.

Multiple configurations are available in the information system. For instance, *sn-price* determines the minimum price of the orders that require the notification for payments. Moreover, *caw-price* determines the minimum price of the items requiring approvals for checking availability. The information system generates event logs recording the execution of the process in the standard OCEL format.

Using the implementation, we 1) construct a DT-IM for the information system, 2) define action patterns, and 3) evaluate action patterns by monitoring the process.

A. Constructing A Digital Twin Interface Model

We first build a DT-IM (DT') that replicates the behavior of the information system. The OCEL containing events from the information system is used to discover an OCPN. Fig. 7(a) shows a screenshot of the control view of the DT-IM. As described in Fig. 3(a), guards are defined with the valves of the information system. Fig. 8(a) is a screenshot of the operational view showing diagnostics of the process at 23:59:59, 02.May.2021. The average/median sojourn time of *send notification* for the last two days (*avg-st-sn-2d* $\in \Delta_{DT'}$), i.e., between 00:00:00, 01.May.2021 and 23:59:59, 02.May.2021 is 16 hours. The routing probability to *check availability with approval* (*rout-prob-caw-2d* $\in \Delta_{DT'}$) for the last two days is 0.3.

B. Defining Action Patterns

We define constraints and actions based on the operational and control view, respectively. By analyzing the diagnostics from the operational view, we conclude that it is problematic for the average sojourn time for the previous two days to be higher than 16 hours, whereas the problematic situation is considered to be relieved if the average sojourn time reaches 10 hours. Based on it, the following constraints are defined:

- *delay-on-notification (C1)* evaluates if the average sojourn time for two days is higher than 16 hours, i.e., [*avg-st-sn-2d*$>$ 16 (hours)]. Fig. 8(b) illustrates how we define *C1* using the user-interface of the implementation.
- *relaxed-delay-on-notification (C2)* evaluates if the average sojourn time for two days is lower than 10 hours, i.e., [*avg-st-sn-2d* $<$ 10 (hours)]

Next, using the valves in the control view, we define the following actions:

- *skip-more-notification (A1)* $\in A_{DT'}$ sets *sn-price* to 220, i.e., for any $conf, conf' \in \Sigma_{DT'}$, $A1(conf)$=$conf'$ such that $conf'$(*sn-price*)=220 and $\forall_{v \in dom(conf) \setminus \{\textit{sn-price}\}}$ $conf'(v)$=$conf(v)$. Fig. 8(a) illustrates how we define *A1* using the user-interface of the implementation.
- *skip-less-notification (A2)* $\in A_{DT'}$ sets *sn-price* to 180, i.e., for any $conf, conf' \in \Sigma_{DT'}$, $A2(conf)$=$conf'$ such that $conf'$(*sn-price*)=180 and $\forall_{v \in dom(conf) \setminus \{\textit{sn-price}\}}$ $conf'(v)$=$conf(v)$.

Based on the constraints and actions, we define following action patterns: *AP1*=(*C1*, *A1*), *AP2*=(*C2*, *A2*) $\in \Pi_{DT'}$. *AP1* evaluates if the average sojourn time for two days is higher than 16 hours and generates actions to skip more notifications, whereas *AP2* generates actions to skip fewer notifications by evaluating if the average sojourn time is lower than 10 hours.

C. Monitoring

Using action patterns and the action engine of the implementation, we analyze the event stream, which is continuously generated by the information system. *AP1* and *AP2* are

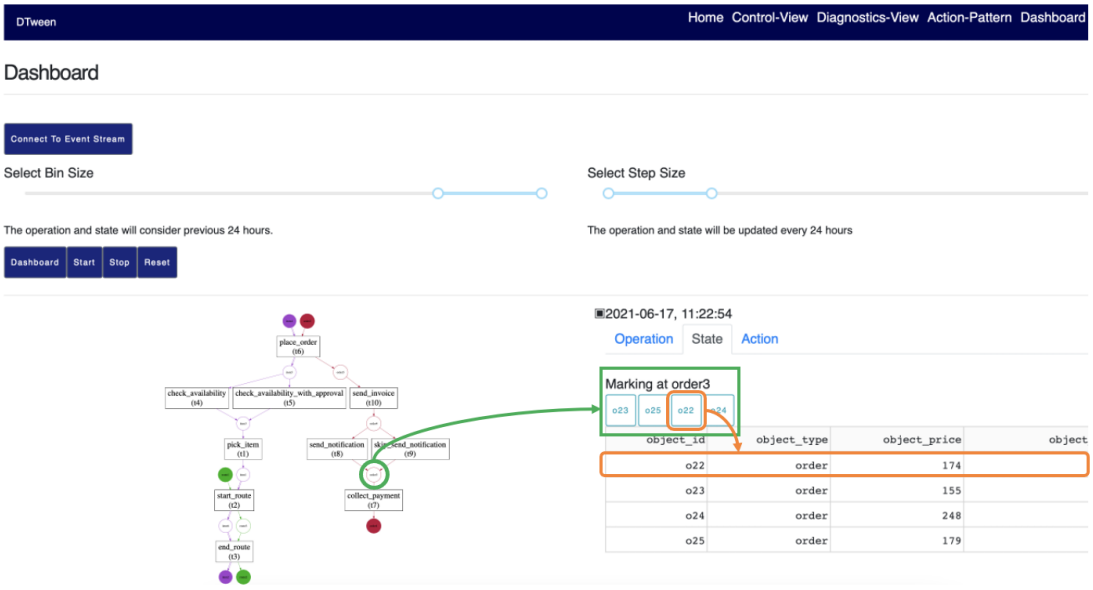

Figure 6: Operational view of the DT-IM replicating the artificial information system. The green box denotes the objects residing in place *order3* and the orange box indicates the value assignment of each object.

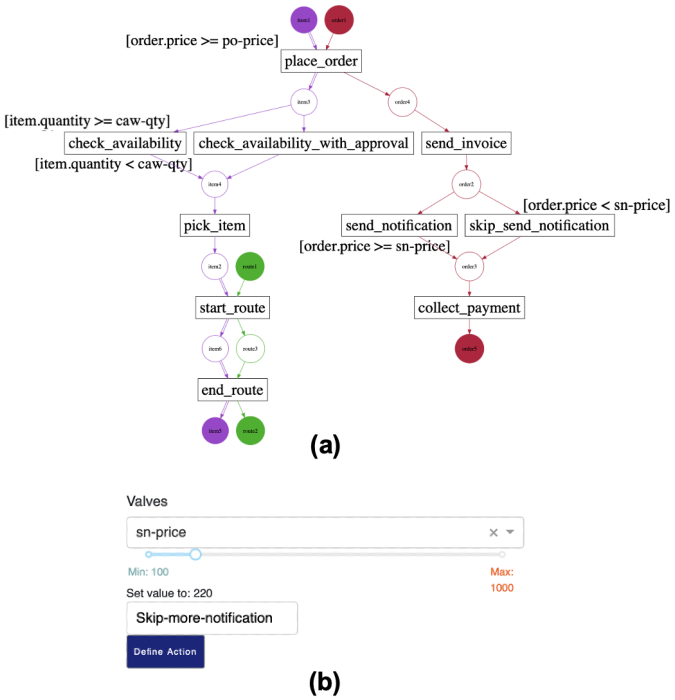

Figure 7: (a) Control view of the DT-IM replicating the artificial information system. Note that it only describes a part of the guards for a better representation. (b) Defining *skip-more-notification (A1)* using the user interface of the implementation.

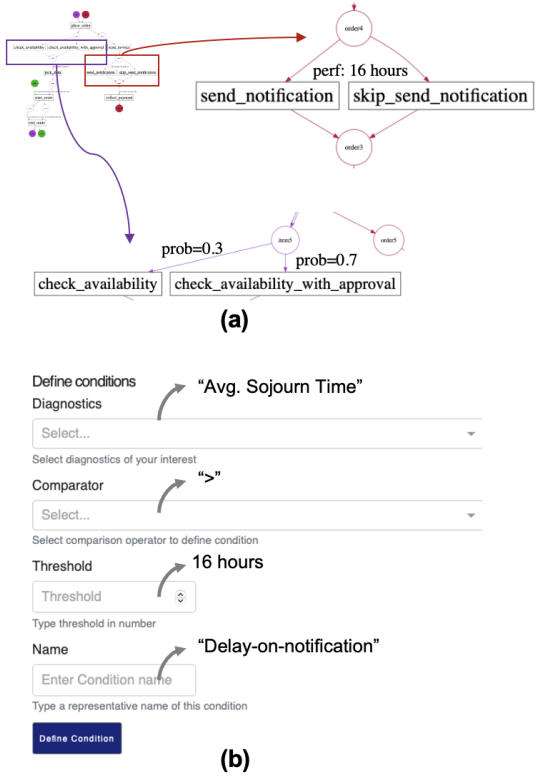

Figure 8: (a) Diagnostics between 00:00:00 01.May.2021 and 23:59:59 02.May.2021. (b) Defining *delay-on-notification (C1)* using the user interface of the user interface of the implementation.

evaluated every 12 hours for 24 days, i.e., from 03.May.2021 to 26.May.2021. Fig. 9 reports the average sojourn time of sending a notification for the last two days by time. The red line denotes the average sojourn time when we trigger the actions defined by the action patterns. At 12:00:00 on 06.May.2021, *A1* was triggered since *C1* was satisfied (i.e., the average sojourn time was higher than 16). As a result of the action, *sn-price* of the DT-IM changed to 220, which in turn updated configurations of the information system. After *A1*, the average sojourn time had decreased until *A2* was triggered

after the average sojourn time reached below 10. *A1* was taken again on 18.May.2021 and 25.May.2021 to deal with delays in sending notifications.

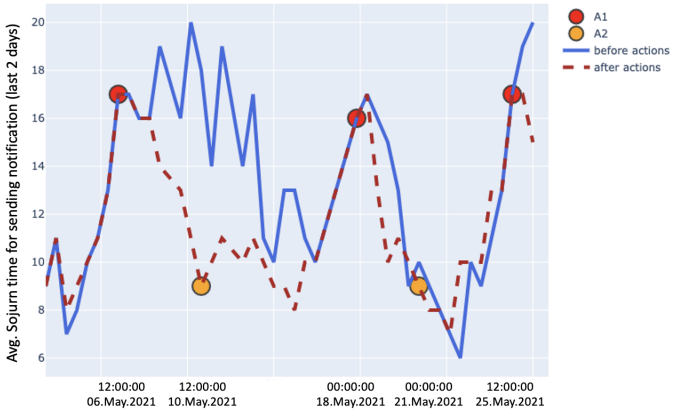

Figure 9: The red dotted line depicts the average sojourn time when actions are triggered, whereas the blue line shows one when actions are not taken. Red/orange dots indicate the execution of *A1*/*A2*.

To evaluate the effectiveness of the action, we measure the average sojourn time when actions are not taken. The blue line depicts the average sojourn time, showing that the execution of actions incredibly reduces the average sojourn time after the first execution of *A1*. For instance, at 12:00:00 10.May.2021, the average sojourn time with actions is 9, whereas the average sojourn time without actions is 20.

VIII. Conclusion

In this paper, we proposed a digital twin interface model to realize DTOs based on action-oriented process mining. An operational view of the interface model reflects the current states of business processes with various diagnostics, while a control view provides routing and resource allocation rules of the process. Process analysts can define action patterns based on the operational and control views, which are evaluated by an action engine to produce the required actions. We have implemented a web service that supports building digital twins, updating states, visualizing the digital twin, and monitoring action patterns. To evaluate the feasibility of the proposed concept, we conducted a case study using an artificial information system supporting an order handling process.

However, given the practical relevance of the proposed approach, the applicability to real-life information systems needs to be extensively evaluated. The evaluation should include the construction of DT-IMs for real-life business processes, real-time updates of operational states, and automated execution of actions. As future work, we plan to conduct case studies with real-life information systems. Moreover, we plan to develop a feedback loop that provides suggestions to improve the existing action patterns. Finally, we will evaluate the inter-effects of different actions to provide recommendations for optimizing existing action patterns.

References

[1] L. Reinkemeyer, "Business view: Towards a digital enabled organization," in *Process Mining in Action: Principles, Use Cases and Outlook*, 2020, pp. 197–206.

[2] Y. Alotaibi and F. Liu, "Survey of business process management: challenges and solutions," *Enterprise Information Systems*, vol. 11, no. 8, pp. 1119–1153, 2017.

[3] E. Glaessgen and D. Stargel, "The digital twin paradigm for future nasa and u.s. air force vehicles," in *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, 2012.

[4] R. Parmar, A. Leiponen, and L. D. Thomas, "Building an organizational digital twin," *Business Horizons*, vol. 63, no. 6, pp. 725–736, 2020.

[5] M. Caporuscio, F. Edrisi, M. Hallberg, A. Johannesson, C. Kopf, and D. Perez-Palacin, "Architectural concerns for digital twin of the organization," in *Software Architecture*, A. Jansen, I. Malavolta, H. Muccini, I. Ozkaya, and O. Zimmermann, Eds., 2020, pp. 265–280.

[6] G. Park and W. M. P. van der Aalst, "A general framework for action-oriented process mining," in *BPM 2020 Workshops*, A. Del Río Ortega, H. Leopold, and F. M. Santoro, Eds., 2020, pp. 206–218.

[7] L. T. Ly, F. M. Maggi, M. Montali, S. Rinderle-Ma, and W. M. P. van der Aalst, "Compliance monitoring in business processes: Functionalities, application, and tool-support," *Information Systems*, vol. 54, pp. 209–234, 2015.

[8] A. E. Marquez-Chamorro, M. Resinas, and A. Ruiz-Cortes, "Predictive Monitoring of Business Processes: A Survey," *IEEE Transactions on Services Computing*, vol. 11, no. 6, pp. 962–977, 2018.

[9] W. M. P. van der Aalst and A. Berti, "Discovering Object-centric Petri Nets," *Fundam. Informaticae*, vol. 175, no. 1-4, pp. 1–40, 2020.

[10] A. F. Ghahfarokhi, G. Park, A. Berti, and W. M. P. van der Aalst, "Ocel: A standard for object-centric event logs," in *New Trends in Database and Information Systems*, L. Bellatreche, M. Dumas, P. Karras, R. Matulevičius, A. Awad, M. Weidlich, M. Ivanović, and O. Hartig, Eds., 2021, pp. 169–175.

[11] F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, and F. Sui, "Digital twin-driven product design, manufacturing and service with big data," *The International Journal of Advanced Manufacturing Technology*, vol. 94, no. 9-12, pp. 3563–3576, 2018.

[12] Q. Qi, F. Tao, Y. Zuo, and D. Zhao, "Digital Twin Service towards Smart Manufacturing," *Procedia CIRP*, vol. 72, pp. 237–242, 2018.

[13] F. M. Maggi, M. Montali, M. Westergaard, and W. M. P. van der Aalst, "Monitoring business constraints with linear temporal logic: An approach based on colored automata," in *BPM 2011*, S. Rinderle-Ma, F. Toumani, and K. Wolf, Eds., vol. 6896, 2011, pp. 132–147.

[14] E. Ramezani, D. Fahland, and W. M. P. van der Aalst, "Where did I misbehave? diagnostic information in compliance checking," in *BPM 2012*, A. Barros, A. Gal, and E. Kindler, Eds., vol. 7481, 2012, pp. 262–278.

[15] M. Pourbafrani, S. J. van Zelst, and W. M. P. van der Aalst, "Scenario-based prediction of business processes using system dynamics," in *OTM 2019*, H. Panetto, C. Debruyne, M. Hepp, D. Lewis, C. A. Ardagna, and R. Meersman, Eds., 2019, pp. 422–439.

[16] J. Evermann, J.-R. Rehse, and P. Fettke, "Predicting process behaviour using deep learning," *Decision Support Systems*, vol. 100, pp. 129–140, 2017.

[17] R. Conforti, M. de Leoni, M. La Rosa, W. M. P. van der Aalst, and A. H. ter Hofstede, "A recommendation system for predicting risks across multiple business process instances," *Decision Support Systems*, vol. 69, pp. 1–19, 2015.

[18] S. A. Fahrenkrog-Petersen, N. Tax, I. Teinemaa, M. Dumas, M. de Leoni, F. M. Maggi, and M. Weidlich, "Fire Now, Fire Later: Alarm-Based Systems for Prescriptive Process Monitoring," *arXiv:1905.09568 [cs, stat]*, 2019.

[19] P. Badakhshan, G. Bernhart, J. Geyer-Klingeberg, J. Nakladal, S. Schenk, and T. Vogelgesang, "The Action Engine – Turning Process Insights into Action," in *2019 ICPM Demo Track*, 2019, pp. 28–31.

[20] A. Berti and W. M. P. van der Aalst, "A novel token-based replay technique to speed up conformance checking and process enhancement," *Trans. Petri Nets Other Model. Concurr.*, vol. 15, pp. 1–26, 2021.

[21] W. M. P. van der Aalst, *Process Mining - Data Science in Action*, 2016.