Checking Constraints for Object-Centric Process Executions

Tian Li, Gyunam Park and Wil M. P. van der Aalst

Process and Data Science Group (PADS), RWTH Aachen University, Germany
tian.li, gnpark, wvdaalst@pads.rwth-aachen.de

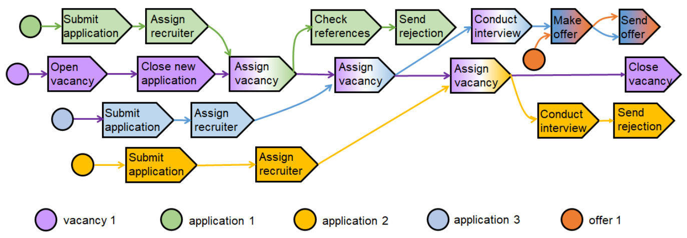
Abstract. Conformance-checking techniques reveal the deviations between event data and the desired process specification, which can be expressed as a process model or a set of rules. State-of-the-art approaches assume a single case identifier, i.e., each case in the business process is associated with only one object. In contrast, processes in real life usually involve multiple object types. For instance, an order management process involves object types such as orders, items, and packages. These objects interact with one another, e.g., packing multiple items from the inventory to create a package. Existing techniques may provide misleading insights when applied to such object-centric event data. We address the issue by extracting process executions (cases) from the object-centric event log and representing constraints using *Object-Centric Constraint Models* (OCCMs). In this way, we handle cardinality, temporal, and performance constraints. Compared to procedural languages like Petri nets, the declarative nature of OCCMs provides more flexibility in modeling constraints, and constraint checking delivers more comprehensive *diagnostics that go beyond isolated cases*. The proposed method has been implemented as a ProM plug-in that supports the extraction of process executions, user-defined OCCMs, and constraint-checking. The feasibility of the proposed approach has been evaluated with other state-of-the-art approaches.

Keywords: Process Mining · Conformance Checking · Constraint Checking· Object-Centric.

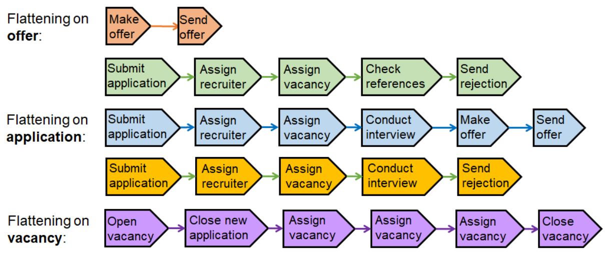
1 Introduction

Process mining techniques aim to analyze event data recorded in the information systems and gain insights into business processes. Identifying unexpected or undesired deviations in processes is critical to mitigating risks or bottlenecks, thus monitoring the operational issues is essential for companies and organizations to maintain operational efficiency.

State-of-the-art constraint-checking/conformance-checking techniques are based on a *single case notion* in the event data, i.e., an event is associated with one object of a unique type (case notion). However, in real-life processes stored in ERP systems, one event can be associated with different objects of multiple types. For example, a simplified recruitment process is illustrated in Fig. 1. The organization opens a new vacancy based on specific needs. Candidates who are interested in the position



(a) Event data of a vacancy hiring process.



(b) flattened sequences based on a particular object type.

Fig. 1: Real-life event data for the recruitment process and flattened event data with one object type.

send applications before the submission deadline. After several rounds of reference checking or interviews, one candidate is hired for the vacancy.

In this process, events can be associated with multiple objects of different types, and each event may have multiple preceding and succeeding events. Therefore, the whole process is not a collection of homogeneously typed event sequence assumed in traditional process mining settings. To bridge this gap between object-centric settings and traditional conformance-checking techniques, the flattening approach [1] is currently used. It first chooses an object type as a case notion, then produces sequences of events related to the lifecycle of objects of this type. As illustrated in Fig. 1b, each case notion provides a sequential event sequence, describing the lifecycle of each object of the selected type. However, there are drawbacks to flattening event data in object-centric settings. To illustrate the problem, we showcase three example constraints as follows:

- constraint 1: *Open vacancy* should be later followed by *Close vacancy*.
- constraint 2: *Close new application* should be preceded by at least one *Submit application*.
- constraint 3: *Submit application* should happen within two months after *Open vacancy*.

When flattening with object type vacancy, constraint 1 is satisfied as *Open vacancy* is followed by *Close vacancy* afterward. However, constraint 2 is violated since *Close new application* is not preceded by any *Submit application*, and constraint 3 is also violated due to the absence of *Submit application* after *Open vacancy*. The underlying reason is that flattening removes activities not related to the object type, thus constraints that involve multiple object types can not be accurately checked.

To tackle the aforementioned problems, we propose a constraint-checking technique that operates in object-centric settings, which provides two key contributions:

1) We introduce an extraction technique that is able to identify process executions from an *Object-Centric Event Log (OCEL)* [5] using one leading object type and multiple secondary object types. **2)** We outline three different constraint models to specify constraints in the extracted process execution. The approach is implemented as a plugin in the ProM package "ObjectCentricConstraintChecking".

The remainder of the paper is organized as follows. In Section 2 we discuss the related work, then we present preliminaries and the notion of process executions in Section 3. In Section 4, we formalize the constraint models for process execution. In Section 5, we evaluate our approach with other existing techniques. Finally, Section 6 concludes the paper.

2 Related Work

The importance of constraint monitoring and follow-up process redesign has long been recognized by the industry. In [11], a technique was proposed to derive monitoring queries from a process model and monitor control-flow deviations. A run-time framework based on Linear Temporal Logic (LTL) was introduced [9]. On the whole, the majority of work for constraints in the process still revolves around traditional event data using a single case notion such that the event data present a sequential structure.

Due to the drawbacks of flattening with a single case identifier when dealing with object-centric event data, there have been a few studies highlighting handling event data with multiple dimensions [2]. A method to model business processes based on artifacts was discussed in [6]. A method to model and reason over object-centric behavioral constraints was proposed in [4]. Jalali et al.[8] tracked rules defined for process instances by considering multiple perspectives. In [7], the authors convert the multi-dimensional event log to graph-based data models and retrieve behavioral insights, such as direct-follow-relations.

3 Preliminaries

Given a set X, the power set $\mathcal{P}(X)$ denotes the set of all possible subsets. A sequence $\sigma = \langle x_1, ..., x_n \rangle$ assigns an order to elements of X, and $len(\sigma) = n$ is the length of σ. X^* denotes the set of all sequences over X.

A graph is a tuple $G = (V, E)$ where V is a set of nodes and $E \subseteq V \times V$ is a set of edges. In an undirected graph, $\forall_{v,v' \in V}: (v,v') \in E \leftrightarrow (v',v) \in E$. For two distinct nodes $v, v' \in V$, $path(v, v') = \{\langle (v, v_1),(v_1, v_2),...,(v_{k-1}, v_k),(v_k, v') \rangle \in E^*\}$ denotes the set of all possible paths between them. $conn(v, v') = true$ if and only if nodes v and v' are connected, i.e., $path(v, v') \neq \emptyset$. For two connected nodes v and v', the distance in between is the length of the shortest path $dist(v, v') = len(\sigma)$ such that $\sigma \in path(v, v') \wedge \forall_{\sigma' \in path(v,v')} \; len(\sigma') \geq len(\sigma)$.

Definition 1 (Object-Centric Event Log (OCEL)). *Let $\mathbb{U}_e$ be the universe of events, $\mathbb{U}_{etype}$ be the universe of event types (i.e., activities), $\mathbb{U}_{time}$ be the universe of timestamps, $\mathbb{U}_o$ be the universe of objects, and $\mathbb{U}_{ot}$ be the universe of object types. An object-centric event log is a tuple $L = (E, O, \pi_{etype}, \pi_{otype}, \pi_{omap}, \pi_{time}, <)$, where:*
- $E \subseteq \mathbb{U}_e$ is a set of events,
- $O \subseteq \mathbb{U}_o$ is a set of objects,

Table 1: Example object-centric event log L_1.

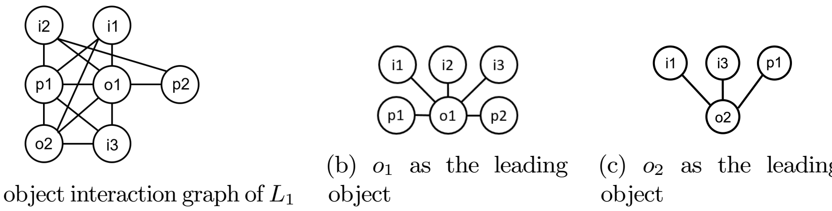
Event	Object type			Event type	Timestamp
	Order	Item	Package		
e1	o1	i1, i2		Place order	2019-01-12 09:36:06
e2	o1	i1		Item out of stock	2019-01-12 10:01:02
e3	o1	i2		Pick item	2019-01-12 10:06:58
e4	o1	i1		Reorder item	2019-01-12 11:59:42
e5	o2	i3		Place order	2019-01-12 13:13:19
e6	o1	i1, i2		Pay order	2019-01-12 15:31:07
e7	o2	i3		Pick item	2019-01-12 16:36:52
e8	o1	i2	p2	Create package	2019-01-14 09:12:35
e9	o1, o2	i1, i3	p1	Create package	2019-01-14 11:01:29
e10	o1		p2	Package delivered	2019-01-15 12:36:01
e11	o1, o2		p1	Package delivered	2019-01-15 15:44:53
e12	o2			Payment reminder	2019-01-22 09:15:41
e13	o2			Pay order	2019-01-23 17:21:56

- $\pi_{etype}: E \to \mathbb{U}_{etype}$ *is the function associating an event to an event type,*
- $\pi_{otype}: E \to \mathbb{U}_{ot}$ *is the function associating an object to an object type,*
- $\pi_{omap}: E \to \mathcal{P}(O)$ *is the function associating an event to a set of related objects,*
- $\pi_{time}: E \to \mathbb{U}_{time}$ *is the function associating an event to a timestamp,*
- $<$ *is a total order on the events.*

An example OCEL L_1 is present in Table 1, where we only keep the event id, object type, event type, and timestamp. There are multiple objects, including $i1$, $i2$, $i3$, $o1$, $o2$, $p1$, $p2 \in \mathbb{U}_o$.

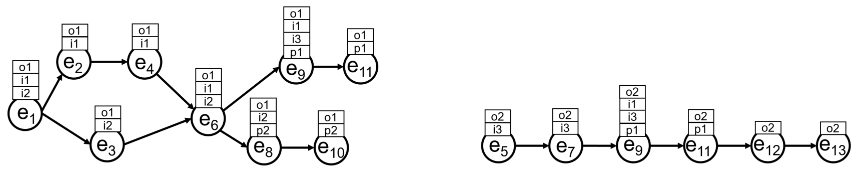
Definition 2 (Object Interaction Graph). *Let* $L = (E, O, \pi_{etype}, \pi_{otype}, \pi_{omap}, \pi_{time}, \leq)$ *be an OCEL, its object interaction graph* $OG_L = \{O, I\}$ *where* O *is a set of objects and* $I = \cup_{e \in E}\{(o_1,o_2) \mid o_1,o_2 \in \pi_{omap}(e) \wedge o_1 \neq o_2\}$.

The object interaction graph of an OCEL consists of nodes of all the objects, and every pair of objects co-occurring in the set of related objects of an event is connected with an undirected edge. The object interaction graph of L_1 has been illustrated in Fig. 2a. The degree of dependency between objects can be measured by the distance in the object interaction graph. For example, items $i1$ and $i3$ are associated with order $o2$, while item $i2$ is not. This association is reflected in the graph as the distance between $i1/i3$ and $o2$ is 1, whereas the distance between $i2$ and $o2$ is 2.



(a) The object interaction graph of L_1 (b) o_1 as the leading object (c) o_2 as the leading object

Fig. 2: Object interaction graph and two subgraphs extracted with leading object type.



(a) Process execution pe_1 (b) Process execution pe_2

Fig. 3: Process executions extracted from corresponding object interaction graphs.

Then we introduce the case concept, i.e., process executions in object-centric event data. Then we explain an extraction technique to extract process executions from OCEL.

Definition 3 (Process Execution). *Let $L = (E, O, \pi_{etype}, \pi_{otype}, \pi_{omap}, \pi_{time}, <)$ be an object-centric event log, and $OG_L = (O, I)$ be the object interaction graph of L. A process execution $pe = (E', O', R')$ is a tuple where:*
- $E' \subseteq E$ is a set of events,
- $O' \subseteq O$ forms a connected subgraph in OG_L.
- $R' \subseteq \{(e, e') \in E' \times E' \mid \exists_{o' \in O'} \exists_{\langle e_1, \ldots, e_n \rangle = trace(o')} \exists_{1 \leq i < n} e = e_i \land e' = e_{i+1}\}$ is the set of process flow relation (i.e. direct-follow-relations),
$\mathbb{U}_{pe}$ denotes the universe of process executions.

We adopt the extraction of process executions following the techniques in [3]. Consider the object interaction graph in Fig. 2a, by selecting Orders as the leading object type, we first extract two object interaction graphs depicted in Fig. 2b and Fig. 2c. Based on the object interaction graph, we extract two process executions present in Fig. 3a and Fig. 3b. For instance, process execution $pe_1 = (E, O, R)$ where $E = (\{e_1, e_2, e_3, e_4\ e_6, e_8\ e_9, e_{10}, e_{11}\}$, $O = \{o1, i1, i2, i3, p1, p2\}$, $R = \{(e_1, e_2), (e_1, e_3), (e_2, e_4), (e_3, e_6), (e_4, e_6), (e_6, e_8), (e_6, e_9), (e_8, e_{10}), (e_9, e_{11})\}$. In order to formalize the semantics of constraint models in the next section, we introduce several notations for process executions.

Definition 4 (Notations for Process Executions). *Let $pe = (E, R, O)$ be a process execution, $et \in \mathbb{U}_{etype}$ be an event type, $e \in E$ be an event in pe. We introduce the following notations:*
- $pre_e(E) = \{e' \in E \mid conn(e', e) = true\}$ is the set of events in pe preceding e, from which e is reachable,
- $suc_e(E) = \{e' \in E \mid conn(e, e') = true\}$ is the set of reachable events succeeding e,
- $all_{et}(E) = \{e' \in E \mid \pi_{etype}(e') = et\}$ is the set of events corresponding to event type et,
- $first_{et}(E) = e$ such that $\pi_{etype}(e) = et \land \nexists_{e' \in E}\ \pi_{etype}(e') = et \land e' < e$ is the first event of type et in p. $first_{et}(E) = \perp$ denotes the absence of events of type et,
- $last_{et}(E) = e$ such that $\pi_{etype}(e) = et \land \nexists_{e' \in E}\ \pi_{etype}(e') = et \land e' > e$ is the last event of type et in p. $last_{et}(E) = \perp$ denotes the absence of events of type et.

As for pe_1 depicted in Fig. 3a, $pre_{e_3}(E) = \{e_1\}$ is the set of events preceding e_3, $suc_{e_3}(E) = \{e_6, e_8\ , e_9, e_{10}, e_{11}\}$ is the set of events succeeding e_3. $all_{Pick\ item}(E) = \{e_3\}$ is the set of events that execute *Pick item*. $first_{Create\ package}(E) = e_8$ is the

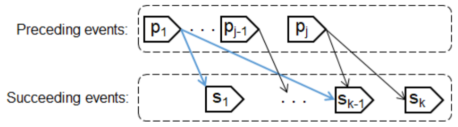


Fig. 4: Sorted events of preceding/succeeding event types in process execution pe.

first event in pe_1 that executes *Create package*. $last_{Package\ delivered}(E) = e_{11}$ is the last event in pe_1 that executes *Package delivered*.

4 Object-Centric Constraint Model

In this section, we explain how to model the constraints for process executions using graphical notation. We first introduce the process flow cardinality and temporal constraint models to describe the constraints from a behavioral perspective. Each constraint has a preceding event type, a succeeding event type, and constraint information that enforces the restrictions between them. Afterward, we focus on the performance constraints w.r.t. an event type in the process executions. Each constraint has one event type and constraint information that specifies the performance requirements.

4.1 Process Flow Constraint Model

In the process execution extracted from object-centric event data, the same event type may occur repeatedly. For example, one *Place order* event is later followed by several *Create package* events, as the items from one order may be packed and sent in different packages. Likewise, one *Create package* may be preceded by multiple *Place order* events, as a package may contain items from different orders.

The above examples are abstracted and present in Fig. 4. Given a process execution $pe = (E, O, R)$, we abstract two event types, i.e., the preceding event type Pre and succeeding event type Suc, such that $Pre = \{p \in E \mid \pi_{type}(p) = pre\}$ is the set of events of type pre, and $Suc = \{s \in E \mid \pi_{type}(s) = suc\}$ is the set of events of type suc. We sort the events in Pre and Suc based on their timestamps. A preceding event $p \in Pre$ is connected to a succeeding event $s \in Suc$ if $conn(p, s)$ is true. As depicted, a preceding event p_1 is connected to succeeding events s_1 and s_{k-1} (edges in blue), which indicates that events s_1 and s_{k-1} are the connected succeeding events for p_1. We introduce cardinality types to indicate the allowed number of connected succeeding events after each preceding event or the required number of connected preceding events before each succeeding event.

Definition 5 (Process Flow Cardinality Constraint Model (CCM)). *Let $A \subseteq \mathbb{U}_{etype}$ be the set of event types, and $C \subseteq \mathbb{U}_{ct}$ be the set of cardinality types. A process flow cardinality constraint model $CCM = (V, C_{cd}, L_{pre\_cd}, L_{suc\_cd})$ is a graph where:*
- *$V \subseteq A$ is a set of nodes,*
- *$C_{cd} \subseteq V \times V$ is the set of cardinality edges that connect a preceding event type to a succeeding event type,*
- *$L_{pre\_cd} \in C_{cd} \rightarrow \mathcal{P}(\mathbb{N})$ maps edges to the cardinality of the preceding event types,*
- *$L_{suc\_cd} \in C_{cd} \rightarrow \mathcal{P}(\mathbb{N})$ maps edges to the cardinality of the succeeding event type.*

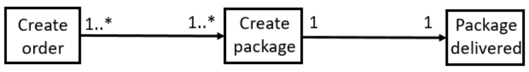


Fig. 5: A CCM for event types *Place order*, *Create package* and *Package delivered.*

Fig. 5 depicts a process flow cardinality constraint model $CCM = (A, C_{cd}, L_{pre\_cd}, L_{suc\_cd})$ with $A = \{$*Place order*, *Create package*, *Package delivered*$\}$, $C_{card} = \{c_1 = ($*Place order*, *Create package*$), c_2 = ($*Create package*, *Package delivered*$)\}$. $L_{pre\_cd}(c_1) = \mathbb{N}^+$ (denoted as $1..*$) indicates that each event executing *Create package* is preceded by at least one event executing *Place order.* $L_{suc\_cd}(c_1) = \mathbb{N}^+$ indicates that each event executing *Place order* is succeeded by at least one event executing *Create package.* $L_{pre\_cd}(c_2) = \{1\}$ (denoted as 1..1) indicates that each event executing *Package delivered* is preceded by exactly one event executing *Create package.* $L_{suc\_cd}(c_2) = \{1\}$ indicates that each event executing *Create package* is succeeded by exactly one event executing *Package delivered.*

Definition 6 (Conformance of CCM). *Let* $CCM = (V, C_{cd}, L_{pre\_cd}, L_{suc\_cd})$ *be a process flow cardinality constraint model, and* $pe = (E, O, R)$ *be a process execution.* pe *satisfies* $c_{cd} = (et_1, et_2) \in C_{cd}$*, if and only if:*
- $\forall_{e \in all_{et_1}(E)}\ |suc_e(all_{et_2(E)})| \in L_{suc\_cd}\ (c_{cd})$,
- $\forall_{e' \in all_{et_2}(E)}\ |pre_{e'}(all_{et_1(E)})| \in L_{pre\_cd}\ (c_{cd})$.

For each constraint in CCM, the preceding event type et_1 defines the events set $all_{et_1}(E)$, while the succeeding event type et_1 defines the events set $all_{et_2}(E)$. For each preceding event $e \in all_{et_1}(E)$, it is checked whether the number of succeeding events of type et_2 is within the allowed number. Likewise, for each succeeding event $e' \in all_{et_2}(E)$, it is checked whether the cardinality constraint is satisfied.

For example, consider the process execution pe_1 in Fig. 3a, event e_1 that executes *Place order* is succeeded by events e_8 and e_9 that executes *Create package.* Both e_8 and e_9 that execute *Create package* are preceded by e_1 that executes *Place order.* Likewise, the cardinalities between event type *Create package* and *Package delivered* are also satisfied. Therefore, process execution pe_1 satisfies the CCM in Fig. 5.

Subsequently, we introduce a model that reflects the temporal constraints between a pair of events in the process execution. For instance, items from one order may be sent with multiple packages, and the timely delivery of all goods guarantees customer satisfaction. Therefore, we define the process flow temporal constraint model to enforce that the time between two specific events must conform to predefined time frames, e.g., it is required that the last *Package delivered* should take place within a week after the *Place order.*

Definition 7 (Process Flow Temporal Constraint Model (TCM)). *Let* $A \subseteq \mathbb{U}_{etype}$ *be the set of event types,* $U \in \{$*seconds, minutes, hours, days, weeks, months*$\}$ *be the time unit, and* $P \in \{first, last\}$ *be the set of temporal patterns. A process flow temporal constraint model* $TCM = (V, C_{tp}, L_{pt}, L_{tp})$ *is a graph where:*
- $V \subseteq A$ *is a set of nodes,*
- $C_{tp} \subseteq A \times A$ *is the set of temporal edges,*
- $L_{pt} \in C_{tp} \rightarrow P \times P$ *maps temporal edges to temporal pattern,*
- $L_{tp} \in C_{tp} \rightarrow \mathbb{R} \times \mathbb{R} \times U$ *maps temporal edges to time frames.*

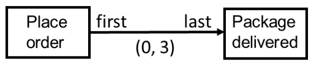


Fig. 6: A TCM for event types *Place order* and *Package delivered*.

Fig. 6 depicts a process flow temporal constraint model $TCM = (V, C_{tp}, L_{pt}, L_{tp})$ with $A = \{$*Place order*, *Package delivered*$\}$, $C_{tp} = \{c_1 = ($*Place order*, *Package delivered*$)\}$. $L_{pt}(c_1) = (first, last)$ indicates that we select the first event in the process execution that executes *Place order* and the last event in the process execution that executes *Package delivered*. $L_{tp}(c_1) = (0, 3, \text{days})$ indicates that the time frames between the first *Place order* and the last *Package delivered* should be within three days.

Definition 8 (Conformance of TCM). *Let* $TCM = (V, C_{tp}, L_{pt}, L_{tp})$ *be a process flow temporal constraint model, and* $pe = (E, O, R)$ *be a process execution. For* $c_{tp} = (v_1, v_2) \in C_{tp}$*, pe satisfies if and only if:*
- $L_{pt}(c_{tp}) = (first, first)$: $\pi_{time}(first_{v_1}(E))$ - $\pi_{time}(first_{v_2}(E)) \in L_{tp}(c_{tp})$,
- $L_{pt}(c_{tp}) = (first, last)$: $\pi_{time}(first_{v_1}(E))$ - $\pi_{time}(last_{v_2}(E)) \in L_{tp}(c_{tp})$,
- $L_{pt}(c_{tp}) = (last, first)$: $\pi_{time}(last_{v_1}(E))$ - $\pi_{time}(first_{v_2}(E)) \in L_{tp}(c_{tp})$,
- $L_{pt}(c_{tp}) = (last, last)$: $\pi_{time}(last_{v_1}(E))$ - $\pi_{time}(last_{v_2}(E)) \in L_{tp}(c_{tp})$.

For each constraint in the model, the preceding event type v_1 with the preceding pattern determines which preceding event to take and the succeeding event type v_2 with the succeeding pattern determines the corresponding succeeding event. It is checked whether the time gap between the two events satisfies the temporal constraint.

For instance, consider the process execution pe_1 in Fig. 3a. Event e_1 that executes *Place order* is succeeded by events e_{10} and e_{11} that execute *Package delivered*. In this scenario, e_1 is chosen as the preceding event, and e_{11} is chosen as the last succeeding *Package delivered*. The time frame is calculated based on their timestamps. Since the time in between is less than three days, pe_1 satisfies TCM.

Definition 9 (Performance Constraint Model (PCM)). *Let* $A \subseteq \mathbb{U}_{etype}$ *be the set of event types,* $\mathbb{U}_{tp}$ *be the universe of time performance types,* $OT \subseteq \mathbb{U}_{ot}$ *be the set of object types,* $T \subseteq \mathbb{U}_{tp}$ *be the time performance types,* $U \in \{$*seconds, minutes, hours, days, weeks, months*$\}$ *be the time unit,* F *be the frequency performance type, and* $C \subseteq \mathbb{U}_{ct}$ *be the set of cardinalities. A performance constraint model* $PCM = (V, E_{ot}, E_t, E_{fq}, L_{ot}, L_t, L_{fq})$ *is a graph where:*
- $V \subseteq A \times OT \times T \times F$ *is a set of nodes,*
- $E_{ot} \subseteq V \times OT$ *is the set of cardinality edges,*
- $E_t \subseteq V \times T$ *is the set of time edges,*
- $E_{fq} \subseteq V \times F$ *is the set of frequency edges,*
- $L_{ot} \in E_{ot} \rightarrow \mathcal{P}(\mathbb{N})$ *maps cardinality edges to object cardinality, i.e., the allowed number of associated objects for this object type,*
- $L_t \in E_t \rightarrow \mathbb{R} \times \mathbb{R} \times U$ *maps time edges to time frames,*
- $L_{fq} \in E_{fq} \rightarrow \mathcal{P}(\mathbb{N})$ *maps frequency edges to frequency value.*

Fig. 7 defines a $PCM = (V, E_{ot}, E_t, E_{fq}, L_{ot}, L_t, L_{fq})$ where $V = \{$*Create package*, *Waiting time*, *Item*, *Order*, *Frequency*$\}$, $E_{ot} = \{e_1 = ($*Create package*, *Order*$)$,

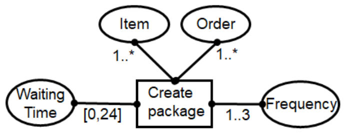


Fig. 7: A PCM for event type *Create package.*

$e_2 = (\textit{Create package},\ \textit{Item})\}$, $E_t = \{e_3 = (\textit{Create package},\ \textit{Waiting time})\}$, $E_{fq} = \{e_4 = (\textit{Create package},\ \textit{Frequency})\}$, $L_{ot}(e_1) = \mathbb{N}^+$ and $L_{ot}(e_2) = \mathbb{N}^+$ indicates that the number of items and orders should be greater than 1, $L_t(e_3) = (0,\ 24,\ \text{hours})$ indicates the waiting time, $L_{fq}(e_5) = 1..3$ implies that the occurrence of *Create Package* in the process execution should lie between one to three times. Consider the process execution pe_1 in Fig. 3a. Events e_8 and e_9 correspond to event type *Create package*, thus the frequency of this event type satisfies the constraint. Event e_9 is associated with two item objects and one order object, while event e_8 is associated with one item and one order object, thus the constraint on object frequency is satisfied. Moreover, since the waiting time for e_8 and e_9 falls within 24 hours, pe_1 satisfies the PCM.

Next, we define the Object-Centric Constraint Model (OCCM) models, which relate multiple behaviors in process executions through a combination of process flow modeling and performance modeling.

Definition 10 (Object-Centric Constraint Model (OCCM)). *An object-centric constraint model OCCM = (TCM, CCM, PCM) is a hybrid graph where:*
- $CCM = (V,\ C_{cd},\ L_{pre\_cd},\ L_{suc\_cd})$ *is a process flow cardinality constraint model,*
- $TCM = (V',\ C_{tp},\ L_{pt},\ L_{tp})$ *is a process flow temporal constraint model,*
- $PCM = (V'',\ E_{ot},\ E_t,\ E_{fq},\ L_{ot},\ L_t,\ L_{fq})$ *is a performance constraint model.*

An object-centric constraint model OCCM combines constraints on the process flow, but also constraints of certain event types that should be satisfied in the process execution. A process execution conforms to the object-centric constraint model $OCCM = (CCM,\ TCM,\ PCM)$ if and only if it satisfies each constraint in CCM, TCM and PCM.

We implement the checking of constraints as a ProM plugin, which enables the user to 1) extract process executions based on a leading object type 2) configure the constraint model based on Definition 5, Definition 7 and Definition 9 3) evaluate the violations of the constraint model in each process execution according to Definition 6 and Definition 8.

5 Implemantations and Evaluations

In this section, we first introduce the implementations of the approach and then evaluate its feasibility by comparing it with other existing techniques.

5.1 Implementations

The approach presented before has been implemented as a ProM plug-in, which supports the following functions: **1)** The extraction of process executions from object-centric event logs. **2)** User-defined graphical OCCM. **3)** Constraint checking and violation diagnosis for the extracted process executions.

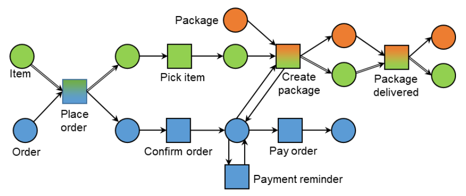


Fig. 8: An order management process: It begins with the event type "Place order", which involves the placement of an order with multiple items. Once the order has been confirmed, the items in the order are picked from inventory and packed into a package. Finally, the "Package delivered" and "Pay order" events mark the completion of the process.

5.2 Evaluations

Based on the implementation, we evaluate the ability of our approach by comparing it with other state-of-the-art models that model the same constraints. Fig. 8 is an *Object-Centric Petri Nets (OCPNs)* [2] that illustrates the simplified process model of an order management process. We illustrate the following five constraints for the extracted process executions in Fig. 9:

– Constraint 1: *Pick item* should be preceded by precisely one *Place order*. Place order could be succeeded by arbitrary number of *Pick item*.
– Constraint 2: the final *Pay order* after the first *Payment reminder* should occur within two weeks.
– Constraint 3: the allowed number of item objects associated with *Create package* should be less than five.
– Constraint 4: the allowed frequency of *Payment reminder* is less than or equal to two times.
– Constraint 5: the waiting time for *Reorder item* should be less than one day.

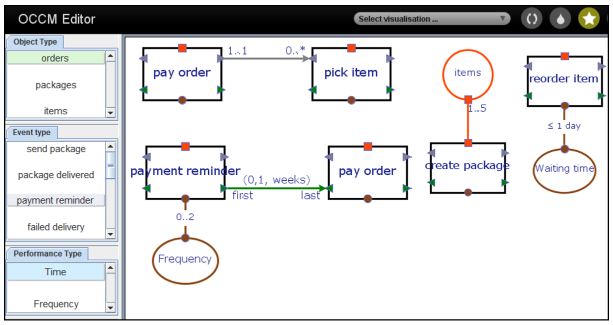
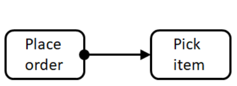


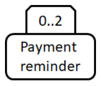
Fig. 9: OCCM in the ProM plug-in.

Next, we evaluate the ability of the OCCM by comparing it with other models that describe the aforementioned constraints in the process.

Fig. 10 is the graphical notations of Declare templates. The response constraint in Fig. 10a approaches constraint 1 by specifying that if *Pick item* occurs, *Place order* occurs beforehand. However, it could not imply the allowed number of *Pick item* after *Place order*, or the allowed number of *Place order* before *Pick item*. As for the atMostTwo constraint in Fig. 10b, it approaches constraint 4 by requiring that *Payment*

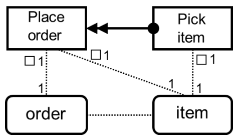


(a) Response constraint

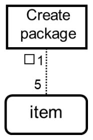


(b) AtMostTwo constraint

Fig. 10: Declare templates. Constraints 2, 3, and 5 cannot be modeled.

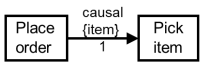


(a) Behavioral constraint

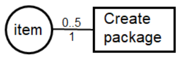


(b) AOC cardinalities

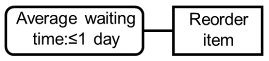
Fig. 11: OCBC modeling technique [4]. Constraints 2, 3, and 5 cannot be modeled.



(a) Causal constraint



(b) Object involvement constraint



(c) performance constraint

Fig. 12: OCCGs modeling technique [10]. Constraints 1, 2, and 4 cannot be modeled.

reminder occurs no more than two times in each case. Declare template is incapable of describing constraints 2, 3, and 5. The isolated case notion does not consider the interaction of different object types, and the performance-related constraints are not specified.

There are other declarative-based techniques that address the drawbacks of using single-case notions. Fig. 11 employs OCBC models to describe the constraints in the event log. This technique relates events in the log through a data perspective. One major drawback is that the temporal constraint, such as constraint 2, is missing due to the absence of process executions. Also, time and frequency performance constraints are missing in OCBC.

Another recent technique uses object-centric constraint graphs (OCCGs) to evaluate constraints based on the metrics of the entire object-centric event log, i.e., the technique does not pinpoint violations for single process execution. As depicted in Fig. 12a, the model describes that a simple causal relation between two event types regarding an item object should hold for all events of type *Place order*. Due to the absence of a process execution notion, it assigns an object type label for the ordering relation. However, when multiple object types interact, the technique is not suitable for modeling specific cardinality or temporal constraints between two events. Thus constraints 1 and 2 are not represented, and frequency constraint 4 is not possible.

In summary, the declarative approach ignores object-centric settings and fails to reflect performance, cardinality, and temporal constraints. In contrast, OCBC and OCCG modeling techniques partly address the issue while not considering constraints for each single process execution, whereas our modeling technique can explicitly represent one-to-many and many-to-many relations on a more granular level. The comparison is summarized in Table 2.

Table 2: Comparison of our approach with existing techniques.

Techniques	Object-centric Settings	Case Notion	Performance	Cardinality	Temporal
Declare [4]	-	✓	-	-	-
OCBC [4]	✓	-	-	✓	-
OCCG [10]	✓	-	✓	-	-
Our work	✓	✓	✓	✓	✓

6 Conclusion

In this paper, we proposed a novel graphical constraint-checking technique for process executions extracted from object-centric event data. The declarative nature of our approach enables a more flexible constraint modeling than procedural languages such as Petri nets. We implemented the language as an editor in ProM, which supports designing models to describe constraints for process executions. One future research direction is to scale the atomic events to non-atomic as the event data in real life can have a non-zero execution duration. Additionally, we plan to extend the approach for more complex constraints, such that deviations on the attribute levels can be detected.

References

1. van der Aalst, W.M.P.: Object-centric process mining: Dealing with divergence and convergence in event data. In: SEFM. Lecture Notes in Computer Science, vol. 11724, pp. 3–25. Springer (2019)
2. van der Aalst, W.M.P., Berti, A.: Discovering object-centric petri nets. Fundam. Informaticae **175**(1-4), 1–40 (2020)
3. Adams, J.N., Schuster, D., Schmitz, S., Schuh, G., van der Aalst, W.M.P.: Defining cases and variants for object-centric event data. In: ICPM. pp. 128–135. IEEE (2022)
4. Artale, A., Kovtunova, A., Montali, M., van der Aalst, W.M.P.: Modeling and reasoning over declarative data-aware processes with object-centric behavioral constraints. In: BPM. Lecture Notes in Computer Science, vol. 11675, pp. 139–156. Springer (2019)
5. Berti, A., van der Aalst, W.M.P.: OC-PM: analyzing object-centric event logs and process models. Int. J. Softw. Tools Technol. Transf. **25**(1), 1–17 (2023)
6. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: BPM. Lecture Notes in Computer Science, vol. 4714, pp. 288–304. Springer (2007)
7. Esser, S., Fahland, D.: Multi-dimensional event data in graph databases. J. Data Semant. **10**(1-2), 109–141 (2021)
8. Jalali, A., Johannesson, P.: Multi-perspective business process monitoring. In: BMMDS/EMMSAD. Lecture Notes in Business Information Processing, vol. 147, pp. 199–213. Springer (2013)
9. 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: BPM. Lecture Notes in Computer Science, vol. 6896, pp. 132–147. Springer (2011)
10. Park, G., van der Aalst, W.M.P.: Monitoring constraints in business processes using object-centric constraint graphs. In: ICPM Workshops. Lecture Notes in Business Information Processing, vol. 468, pp. 479–492. Springer (2022)
11. Weidlich, M., Ziekow, H., Mendling, J., Günther, O., Weske, M., Desai, N.: Event-based monitoring of process execution violations. In: BPM. Lecture Notes in Computer Science, vol. 6896, pp. 182–198. Springer (2011)