Monitoring Constraints in Business Processes Using Object-Centric Constraint Graphs

Gyunam Park [ID] and Wil. M. P. van der Aalst [ID]

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

Abstract. Constraint monitoring aims to monitor the violation of constraints in business processes, e.g., an invoice should be cleared within 48 hours after the corresponding goods receipt, by analyzing event data. Existing techniques for constraint monitoring assume that a single case notion exists in a business process, e.g., a patient in a healthcare process, and each event is associated with the case notion. However, in reality, business processes are *object-centric*, i.e., multiple case notions (objects) exist, and an event may be associated with multiple objects. For instance, an Order-To-Cash (O2C) process involves *order*, *item*, *delivery*, etc., and they interact when executing an event, e.g., packing multiple items together for a delivery. The existing techniques produce misleading insights when applied to such object-centric business processes. In this work, we propose an approach to monitoring constraints in object-centric business processes. To this end, we introduce *Object-Centric Constraint Graphs* (OCCGs) to represent constraints that consider the interaction of objects. Next, we evaluate the constraints represented by OCCGs by analyzing Object-Centric Event Logs (OCELs) that store the interaction of different objects in events. We have implemented a web application to support the proposed approach and conducted two case studies using a real-life SAP ERP system.
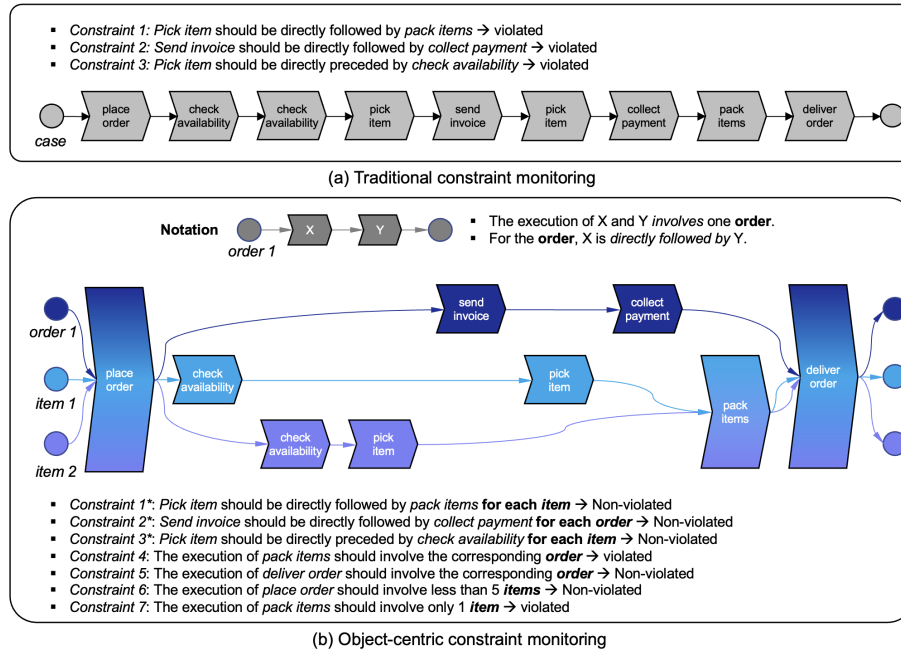
Keywords: Constraint Monitoring · Object-Centric Process Mining · Compliance Checking · Process Monitoring

1 Introduction

It is indispensable for organizations to continuously monitor their operational problems and take proactive actions to mitigate risks and improve performance [2]. Constraint monitoring aims at detecting violations of *constraints* (i.e., operational problems) in business processes of an organization by analyzing event data recorded by information systems [8]. Once violations are detected, the organization can activate management actions to cover the respective violation [10].

A plethora of techniques has been suggested to implement constraint monitoring. For instance, a technique is proposed to detect events violating constraints, e.g., detecting an X-ray event with a long waiting time, using behavioral profiles and Complex Event Processing (CEP) [13]. Maggi et al. [9] propose a technique to detect process instances violating constraints, e.g., detecting a patient with multiple executions of X-rays, using Linear Temporal Logic (LTL).

The existing techniques assume that an event in event data is associated with a single object of a unique type (so-called case), e.g., a patient in a healthcare process. Thus, constraints are defined over the single case notion, e.g., each patient (i.e., case) should be registered before triage. However, in real-life business processes, an event may be associated with multiple objects of different types, i.e., real-life business processes are *object-centric* [1]. For instance, the omnipresent Purchase-To-Pay (P2P) process involves different object types, e.g., *purchase order*, *material*, *invoice*, *goods receipt*, etc., and an event may be associated with multiple objects of different types, e.g., clearing invoice is associated with a purchase order, an invoice, and a goods receipt to enable so-called *three-way matching*.

(a) Traditional constraint monitoring

(b) Object-centric constraint monitoring

Fig. 1: Comparing (a) traditional and (b) object-centric constraint monitoring

Applying the existing techniques to such object-centric settings results in misleading insights. Fig. 1(a) shows events of a "case" in an Order-To-Cash (O2C) process using *order* as the case notion. First, an order is placed, and the availability of two items of the order is checked, respectively. Next, one of the items is picked, and the invoice of the order is sent to the customer. Afterward, the other item is picked, and the payment of the invoice is collected. Finally, the items are packed and delivered to the customer. The three constraints shown in Fig. 1(a) are violated by the case. For instance, *Constraint 1* is violated since *pick item* is followed by *send invoice* in the case and *Constraint 3* is violated since *pick item* is preceded by *send invoice*.

However, in reality, the order and each item have different lifecycles as shown in Fig. 1(b). First, we place an order with two items. While the invoice is sent and the payment is collected for the order, we check the availability of each item and pick each of them. We finally deliver the order with two items after packing two items together. In

this object-centric setting, constraints should be defined in relation to objects to provide accurate insights. For instance, *Constraint 1** extends *Constraint 1* with the corresponding object type (i.e., item). Contrary to *Constraint 1*, *Constraint 1** is not violated since *pick item* is directly followed by *pack item* for any items. Moreover, we can analyze more object-centric constraints by considering the interaction of different objects. First, we can analyze if an execution of an activity involves (un)necessary objects (cf. *Constraint 4* and *Constraint 5*). Also, we can analyze the cardinality of objects for executing an activity (cf. *Constraint 6* and *Constraint 7*).

In this work, we propose a technique for constraint monitoring in object-centric settings. To this end, we first introduce *object-centric behavioral metrics* that can be computed from Object-Centric Event Logs (OCELs), e.g., a metric to measure the degree to which *pick item* precedes *pack items* in the lifecycle of *items*. Next, we develop Object-Centric Constraint Graphs (OCCGs) to formally represent constraints using such metrics. Finally, *monitoring engine* evaluates the violation of the constraints represented by OCCGs by analyzing OCELs.

We have implemented a web application to support the approach. A demo video and a manual are available at `https://github.com/gyunamister/ProPPa.git`. Moreover, we have conducted case studies with a production process and a Purchase-To-Pay (P2P) process supported by an SAP ERP system.

The remainder is organized as follows. We discuss the related work in Sec. 2 and present the preliminaries, including OCELs in Sec. 3. In Sec. 4, we introduce object-centric behavioral metrics. Afterward, we present OCCGs to formally represent constraints and the monitoring engine to evaluate the violation of the constraints in Sec. 5. Next, Sec. 6 introduces the implementation of the proposed approach and case studies using real-life event data. Finally, Sec. 7 concludes the paper.

2   Related Work

Many approaches have been proposed to monitor the violation of constraints by analyzing event data. Weidlich et al. [13] propose a technique to abstract process models to behavioral profiles and produce event queries from the profile. Violated executions of events are monitored using Complex Event Processing (CEP) engines with the event queries. Awad et al. [5] define a set of generic patterns regarding the occurrence of tasks, their ordering, and resource assignments and generate anti-patterns from the generic patterns to monitor event executions. Maggi et al. [9] represent control-flow properties of a running process instance using Linear Temporal Logic (LTL) and evaluate their violations at runtime. Also, Petri-net-based constraints are aligned with event logs to evaluate whether the execution of business processes conforms to the constraints [12]. Indiono et al. [7] propose an approach to monitoring Instance-Spanning Constraints (ISCs) that span multiple instances of one or several processes based on Rete algorithm. However, the existing techniques may produce misleading insights in object-centric settings since it does not consider the interaction among objects of different types. Moreover, object-centric constraints, e.g., the cardinality of an object type for the execution of an activity, are not supported in the existing techniques.

Table 1: A fragment of an event log.

event id	activity	timestamp	order	item
e_{93}	*place order (po)*	25-10-2022:09.35	$\{o_1\}$	$\{i_1,i_2,i_3\}$
e_{94}	*evaluate credit (ec)*	25-10-2022:13.35	$\{o_1\}$	$\emptyset$
e_{95}	*confirm order (co)*	25-10-2022:15.35	$\{o_1\}$	$\{i_1,i_2,i_3\}$

This paper is in line with the recent developments in object-centric process mining [1]. Object-centric process mining breaks the assumption of traditional process mining techniques that each event is associated with a single case notion (i.e., object), allowing one event to be associated with multiple objects. Moreover, a process discovery technique is proposed to discover Object-Centric Petri Nets (OCPNs) from OCELs [3]. Furthermore, Adams et al. [4] propose a conformance checking technique to determine the precision and fitness of the net, and Park et al. propose an approach to object-centric performance analysis [11]. Esser and Fahland [6] propose a graph database as a storage format for object-centric event data, enabling a user to use queries to calculate different statistics. This work extends the current development in the field of object-centric process mining by proposing a constraint monitoring technique in object-centric settings.

3   Preliminaries

Given a set X, the powerset $\mathscr{P}(X)$ denotes the set of all possible subsets. We denote a sequence with $\sigma = \langle x_1,\dots,x_n \rangle$ and the set of all sequences over X with X^*. Given a sequence $\sigma \in X^*$, $x \in \sigma$ if and only if $\exists_{1 \leq i \leq |\sigma|}\ \sigma(i)=x$.

Definition 1 (Universes). $\mathbb{U}_{ei}$ *is the universe of event identifiers,* $\mathbb{U}_{oi}$ *is the universe of object identifiers,* $\mathbb{U}_{act}$ *is the universe of activity names,* $\mathbb{U}_{time}$ *is the universe of timestamps,* $\mathbb{U}_{ot}$ *is the universe of object types,* $\mathbb{U}_{attr}$ *is the universe of attributes,* $\mathbb{U}_{val}$ *is the universe of values, and* $\mathbb{U}_{map}=\mathbb{U}_{attr} \nrightarrow \mathbb{U}_{val}$ *is the universe of attribute-value mappings. For any* $f \in \mathbb{U}_{map}$ *and* $x \notin dom(f)$, $f(x)=\bot$.

Using the universes, we define an object-centric event log as follows.

Definition 2 (Object-Centric Event Log). *An object-centric event log is a tuple* $L=(E,O, \mu,R)$, *where* $E \subseteq \mathbb{U}_{event}$ *is a set of events,* $O \subseteq \mathbb{U}_{oi}$ *is a set of objects,* $\mu \in (E \rightarrow \mathbb{U}_{map}) \cup (O \rightarrow (\mathbb{U}_{time} \rightarrow \mathbb{U}_{map}))$ *is a mapping, and* $R \subseteq E \times O$ *is a relation, such that for any* $e \in E$, $\mu(e)(act) \in \mathbb{U}_{act}$ *and* $\mu(e)(time) \in \mathbb{U}_{time}$, *and for any* $o \in O$ *and* $t,t' \in \mathbb{U}_{time}$, $\mu(o)(t)(type)=\mu(o)(t')(type) \in \mathbb{U}_{ot}$. $\mathbb{U}_L$ *is the set of all possible object-centric event logs.*

For the sake of brevity, we denote $\mu(e)(x)$ as $\mu_x(e)$ and $\mu(o)(t)(x)$ as $\mu_x^t(o)$. Since the type of an object does not change over time, we denote $\mu_{type}^t(o)$ as $\mu_{type}(o)$. Table 1 describes a fraction of a simple event log $L_1=(E_1,O_1,\mu_1,R_1)$ with $E_1=\{e_{93},e_{94},e_{95}\}$, $O_1=\{o_1,i_1,i_2,i_3\}$, $R_1=\{(e_{93},o_1),(e_{93},i_1),\dots\}$, $\mu_{act}(e_{93})=po$, $\mu_{time}(e_{93})=$*25-10-2022:09.35*, $\mu_{type}(o_1)=$*Order*, and $\mu_{type}(i_1)=$*Item*.

We define functions to query event logs as follows:

Definition 3 (Notations). *For an object-centric event log $L=(E,O,\mu,R)$, we introduce the following notations:*

- *$acts(L)=\{\mu_{act}(e)\,|\,e\in E\}$ is the set of activities,*
- *$events(L,a)=\{e\in E\,|\,\mu_{act}(e)=a\}$ is the set of the events associated to $a\in acts(L)$,*
- *$types(L)=\{\mu_{type}(o)\,|\,o\in O\}$ is the set of object types,*
- *$objects(L,ot)=\{o\in O\,|\,\mu_{type}(o)=ot\}$ is the set of the objects associated to $ot\in types(L)$,*
- *$events(L,o)=\{e\in E\,|\,(e,o)\in R\}$ is the set of the events containing $o\in O$,*
- *$objects(L,e)=\{o\in O\,|\,(e,o)\in R\}$ is the set of the objects involved in $e\in E$,*
- *$seq(o)=\langle e_1,e_2,...,e_n\rangle$ such that $events(L,o)=\{e_1,e_2,...,e_n\}$ and $\mu_{time}(e_i)\leq\mu_{time}(e_j)$ for any $1\leq i<j\leq n$ is the sequence of all events where object $o\in O$ is involved in, and*
- *$trace(o)=\langle a_1,a_2,...,a_n\rangle$ such that $seq(o)=\langle e_1,e_2,...,e_n\rangle$ and $a_i=\mu_{act}(e_i)$ for any $1\leq i\leq n$ is the trace of object $o\in O$.*

For instance, $acts(L_1)=\{po,ec,co\}$, $events(L_1,po)=\{e_{93}\}$, $types(L_1)=\{Order,Item\}$, $objects(L_1,Order)=\{o_1\}$, $events(L_1,o_1)=\{e_{93},e_{94},e_{95}\}$, $objects(L_1,e_{93})=\{o_1,i_1,i_2,i_3\}$, $seq(o_1)=\langle e_{93},e_{94},e_{95}\rangle$, and $trace(o_1)=\langle po,ec,co\rangle$.

Using the notations, we characterize an event log as follows:

Definition 4 (Log Characteristics). *Let $L=(E,O,\mu,R)$ be an object-centric event log. For $ot\in types(L)$ and $a,b\in acts(L)$, we define the following characteristics of L:*

- *$\#_L(ot,X)=|\{o\in objects(L,ot)\,|\,\forall_{x\in X}\,x\in trace(o)\}|$ counts the objects of type ot whose trace contains $X\subseteq acts(L)$,*
- *$\#_L(ot,a,b)=|\{o\in objects(L,ot)\,|\,\exists_{1\leq i<j\leq|trace(o)|}\,trace(o)(i)=a\wedge trace(o)(j)=b\}|$ counts the objects of type ot whose trace contains a followed by b,*
- *$\#_L^0(ot,a)=|\{e\in events(L,a)\,|\,|\{o\in objects(L,e)\,|\,\mu_{type}(o)=ot\}|=0\}|$ counts the events relating no objects of type ot for the execution of a,*
- *$\#_L^1(ot,a)=|\{e\in events(L,a)\,|\,|\{o\in objects(L,e)\,|\,\mu_{type}(o)=ot\}|=1\}|$ counts the events relating one object of type ot for the execution of a, and*
- *$\#_L^*(ot,a)=|\{e\in events(L,a)\,|\,|\{o\in objects(L,e)\,|\,\mu_{type}(o)=ot\}|>1\}|$ counts the events relating more than one object of type ot for the execution of a.*

For instance, $\#_{L_1}(Order,\{po\})=1$, $\#_{L_1}(Item,\{po\})=3$, $\#_{L_1}(Item,\{po,ec\})=0$, $\#_{L_1}(Order,po,ec)=1$, $\#_{L_1}^0(Order,ec)=0$, $\#_{L_1}^0(Item,ec)=1$, $\#_{L_1}^1(Order,po)=1$, $\#_{L_1}^1(Item,po)=0$, $\#_{L_1}^*(Order,po)=0$, and $\#_{L_1}^*(Item,po)=1$.

4   Object-Centric Behavioral Metrics

To introduce OCCGs, we first explain three types of object-centric behavioral metrics derived from an event log: *ordering relation*, *object involvement*, and *performance* metrics. Such metrics are used to define the semantics of OCCGs in Sec. 5.

An ordering relation metric refers to the strength of a causal/concurrent/choice relation between two activities in an OCEL w.r.t. an object type.

Definition 5 (Ordering Relation Metrics). *Let L be an object-centric event log. For $ot\in types(L)$ and $a,b\in acts(L)$, we define the following ordering relation metrics of L:*

- $causal_L(ot,a,b) = \begin{cases} \frac{\#_L(ot,a,b)}{\#_L(ot,\{a,b\})}, if\, \#_L(ot,\{a,b\}) > 0 \\ 0, otherwise \end{cases}$

- $concur_L(ot,a,b) = \begin{cases} 1 - \frac{max(\#_L(ot,a,b),\#_L(ot,b,a)) - min(\#_L(ot,a,b),\#_L(ot,b,a))}{\#_L(ot,a,b) + \#_L(ot,b,a)}, \\ if\, \#_L(ot,a,b) + \#_L(ot,b,a) > 0 \\ 0, otherwise \end{cases}$

- $choice_L(ot,a,b) = \begin{cases} 1 - \frac{\#_L(ot,\{a,b\}) + \#_L(ot,\{a,b\})}{\#_L(ot,\{a\}) + \#_L(ot,\{b\})}, if\, \#_L(ot,\{a\}) + \#_L(ot,\{b\}) > 0 \\ 0, otherwise \end{cases}$

$causal_L(ot,a,b)$, $concur_L(ot,a,b)$, and $choice_L(ot,a,b)$ all produce values between 0 (weak) and 1 (strong). For L_1 in Table 1, $causal_{L_1}(Order,po,co) = 1$, $concur_{L_1}(Order,po, co) = 0$, $choice_{L_1}(Order,po,co) = 0$, showing that po and co has a strong (not only directly, but also eventually) causal ordering relation.

Next, an object involvement metric quantitatively represents how the execution of an activity involves objects.

Definition 6 (Object Involvement Metrics). *Let L be an object-centric event log. For* $ot \in types(L)$ *and* $a \in acts(L)$, *we define three object involvement metrics of L in the following.*

- $absent_L(ot,a) = \frac{\#_L^0(ot,a)}{|events(L,a)|}$ *is the strength of ot's absence in a's execution.*
- $singular_L(ot,a) = \frac{\#_L^1(ot,a)}{|events(L,a)|}$ *is the strength of ot's singularity in a's execution.*
- $multiple_L(ot,a) = \frac{\#_L^*(ot,a)}{|events(L,a)|}$ *is the strength of ot's multiplicity in a's execution.*

All object involvement metrics produce values between 0 (weak) and 1 (strong). For L_1 in Table 1, $absent_{L_1}(Item,ec) = 1$, showing that items are not involved in the execution of ec. $singular_{L_1}(Order,po) = 1$ and $multiple_{L_1}(Item,po) = 1$, indicating that the execution of po involves only one order and multiple items.

Finally, a performance metric refers to a performance/frequency value related to the execution of an activity.

Definition 7 (Performance Metrics). *Let L be an object-centric event log. Let* $\mathbb{U}_{measure}$ *be the universe of performance/frequency measure names, e.g., the average waiting time. A performance metric of L, $perf_L \in (acts(L) \times \mathbb{U}_{measure}) \nrightarrow \mathbb{R}$, maps an activity and a performance/frequency measure to the value of the performance/frequency measure w.r.t. the activity.*

Note that we deliberately "underspecify" performance metrics, abstracting from the definition of individual performance metrics. Performance metrics may include the average number of objects per object type for the execution of an activity (e.g., the average number of *items* for placing an order), the average *sojourn time* for the execution of an activity (e.g., the average sojourn time for confirming an order), etc. For L_1 in Table 1, $perf_{L_1}(po,avg\text{-}num\text{-}items) = 3$, which denotes that the average number of *items* for *po* in L_1 is 3. Also, $perf_{L_1}(co,avg\text{-}sojourn\text{-}time) = 2\ hours$, which denotes that the average sojourn time for *co* in L_1 is 2 hours.

5   Object-Centric Constraint Monitoring

In this section, we explain our proposed approach to object-centric constraint monitoring. To this end, we first introduce Object-Centric Constraint Graphs (OCCGs) to represent constraints. Next, we introduce a monitoring engine to evaluate the violation of constraints represented by OCCGs by analyzing OCELs.

5.1   Object-Centric Constraint Graphs (OCCGs)

An OCCG is a directed graph that consists of nodes and edges, as depicted in Fig. 2. Nodes consist of activities, object types, and *formulas*. A formula is a logical expression defined over performance measures of an activity using relational operators ($\leq, \geq, =$) as well as logical operators such as conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$). Edges describe control-flow, object involvement, and performance edges.
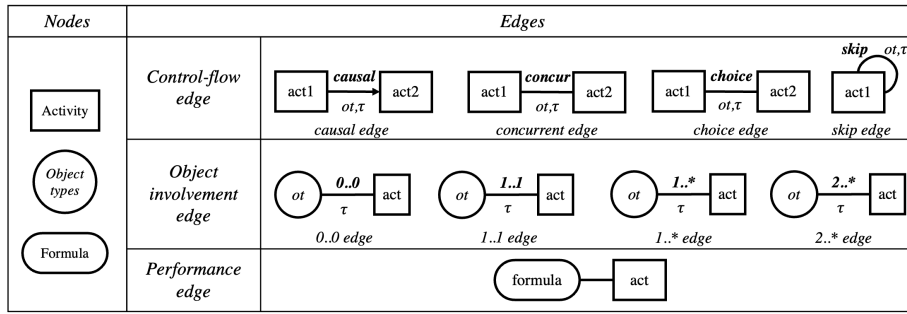

Fig. 2: Graphical notations of OCCGs. $act \in \mathbb{U}_{act}$, $ot \in \mathbb{U}_{ot}$, and $\tau \in [0,1]$.

Definition 8 (Object-Centric Constraint Graph). *Let $F(X)$ be the set of all possible logical expressions with set X. Let $A \subseteq \mathbb{U}_{act}$, $OT \subseteq \mathbb{U}_{ot}$, and $\mathscr{F} \subseteq F(\mathbb{U}_{measure})$. Let $C=\{causal,concur,choice,skip\}$ be the set of control-flow labels and $I=\{0..0,1..1,1..*,2..*\}$ the set of object involvement labels. An object-centric constraint graph is a graph $cg=(V,E_{flow},E_{obj},E_{perf},l_c,l_i,l_\tau)$ where*
  – *$V \subseteq A \cup OT \cup \mathscr{F}$ is a set of nodes,*
  – *$E_{flow} \subseteq A \times OT \times A$ is a set of control-flow edges,*
  – *$E_{obj} \subseteq OT \times A$ is a set of object involvement edges,*
  – *$E_{perf} \subseteq \mathscr{F} \times A$ is a set of performance edges,*
  – *$l_c \in E_{flow} \to C$ maps control-flow edges to control-flow labels such that, for any $(a,ot,b) \in E_{flow}$, if $l_c((a,ot,b))=skip$, $a=b$,*
  – *$l_i \in E_{obj} \to I$ maps object involvement edges to object involvement labels, and*
  – *$l_\tau \in E_{flow} \cup E_{obj} \to [0,1]$ maps control-flow and object involvement edges to thresholds.*
$\mathbb{U}_{cg}$ denotes the set of all possible object-centric constraint graphs.

Fig. 3(a)-(k) introduces some example of OCCGs defined in an O2C process. For instance, Fig. 3(a) is formally represented as follows: $cg'=(V',E'_{flow},\emptyset,\emptyset,l'_c,\emptyset,l'_\tau)$ where $V'=\{collect\ payment,send\ reminder\}$, $E'_{flow}=\{e_1=(collect\ payment,Order,send\ reminder)\}$, $l'_c(e_1)=causal$, and $l'_\tau(e_1)=0$.
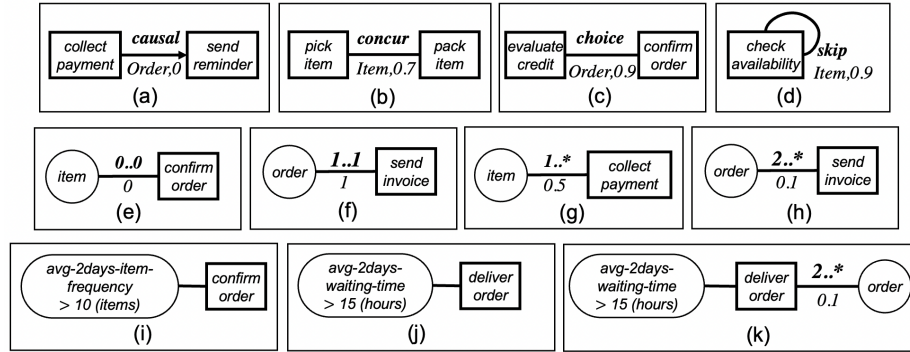
Fig. 3: Examples of object-centric constraint graphs.

We define the semantics of an OCCG with the notion of *violation*. An OCCG is *violated* in an OCEL if all constraints represented in its edges are satisfied.

Definition 9 (Semantics of object-centric constraint graphs). *Let L be an object-centric event log. An object-centric constraint graph $cg=(V,E_{flow},E_{obj},E_{perf},l_c,l_i,l_\tau)$ is violated in L if*

1. *for any $e=(a,ot,b)\in E_{flow}$ s.t. $ot\in types(L)\wedge a,b\in acts(L)$,*
   - *$causal_L(ot,a,b)>l_\tau(e)$ if $l_c(e)=causal$,*
   - *$concur_L(ot,a,b)>l_\tau(e)$ if $l_c(e)=concur$,*
   - *$choice_L(ot,a,b)>l_\tau(e)$ if $l_c(e)=choice$, and*
   - *$1-\frac{\#_L(ot,\{a\})}{|objects(L,ot)|}>l_\tau(e)$ if $l_c(e)=skip$,*
2. *for any $e=(ot,a)\in E_{obj}$ s.t. $ot\in types(L)\wedge a\in acts(L)$,*
   - *$absent_L(ot,a)>l_\tau(e)$ if $l_i(e)=0..0$,*
   - *$singular_L(ot,a)>l_\tau(e)$ if $l_i(e)=1..1$,*
   - *$1-absent_L(ot,a)>l_\tau(e)$ if $l_i(e)=1..*$, and*
   - *$multiple_L(ot,a)>l_\tau(e)$ if $l_i(e)=2..*$,*
3. *for any $(f,a)\in E_{perf}$ s.t. $a\in acts(L)$, f evaluates to true w.r.t. $perf_L$.*

For instance, Fig. 3(a) is violated if *collect payment* is preceded by *send reminder* at all w.r.t. *Order*, Fig. 3(b) is violated if *pick item* and *pack item* are concurrently executed with the strength higher than 0.7 w.r.t. *Item*, Fig. 3(e) is violated if *confirm order* is executed without involving *Item* at all, Fig. 3(k) is violated if the average waiting time of the last two days for *deliver order* is longer than 15 hours, and its execution involves multiple orders with the strength higher than 0.1, etc.

5.2 Monitoring Engine

A monitoring engine analyzes the violation of OCCGs by analyzing an OCEL.

Definition 10 (Monitoring Engine). *A monitoring engine monitor $\in \mathbb{U}_L \times \mathbb{U}_{cg} \rightarrow \{true,false\}$ is a function that maps an object-centric event log and an object-centric*

constraint graph to a Boolean value. For any $L \in \mathbb{U}_L$ and $cg \in \mathbb{U}_{cg}$, monitor(L,cg)=true if cg is violated in L, and false, otherwise.

We implement the monitoring engine by 1) computing the object-centric behavioral metrics of an event log and 2) evaluating the violation of OCCGs based on them. First, the derivation of ordering relation metrics and object involvement metrics is deterministic according to Def. 5 and Def. 6, respectively. However, the computation of performance metrics is non-trivial. In this work, we use the approach proposed in [11] to compute performance measures, such as *sojourn time*, *waiting time*, *service time*, *flow time*, *synchronization time*, *pooling time*, and *lagging time*, and frequency measures, such as *object count*, *object type count*, etc. Finally, using the object-centric behavioral metrics, we evaluate the violation of OCCGs according to Def. 9.

6   Implementation and Case Studies

This section presents the implementation of the approach presented in this paper and evaluates the feasibility of the approach by applying it to a production process and a P2P process of a real-life SAP ERP system.

6.1   Implementation

The approach presented in this work has been fully implemented as a web application[1] with a dedicated user interface. The following functions are supported:
  – Importing OCELs in different formats, including OCEL JSON, OCEL XML, and CSV.
  – Designing object-centric constraint graphs using graphical tools.
  – Computing object-centric behavioral metrics of OCELs and evaluating the violation of object-centric constraint graphs based on the metrics.
 – Visualizing monitoring results with detailed analysis results.

6.2   Case Study: Production Process

Using the implementation, we conduct a case study on a production process of a fictitious company supported by an SAP ERP system. The process involves four object types: *production order*, *reservation*, *purchase requisition*, and *purchase order*. Fig. 4 shows a process model of the production process using Object-Centric Petri Nets (OCPNs) as a formalism. We refer readers to [3] for the details of OCPNs.

We represent the following constraints using OCCGs:
 – *Skipping Purchase Requisition Approval (PRA)*; A purchase requisition should not skip the approval step at all. Fig. 5(a) represents the constraint.
 – *No reservation for Purchase Requisition Approval (PRA)*; The execution of *approve purchase requisition* is supposed to include the corresponding reservation most of the time. Fig. 5(b) represents the constraint.

---

[1] A demo video and sources: `https://github.com/gyunamister/ProPPa.git`
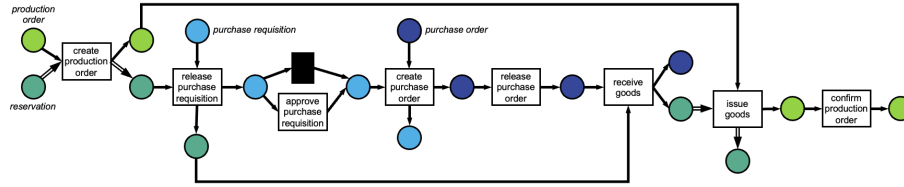
Fig. 4: Production process: First, a production order is created with a variable number of reservations (i.e., required materials). Next, a purchase requisition is released and approved. Afterward, a purchase order is created based on the purchase requisition. Once the order is released, the reservations are received and issued for production. Finally, the production order is confirmed.

– *Excessive reservations per Production Order (PO)*; The execution of *create production order* should not involve more than one reservation on average. Fig. 5(c) represents the constraint.
– *Delayed Purchase Order Release (POR)*; The average sojourn time of *release purchase order* should be less than 15 days. Fig. 5(d) represents the constraint.
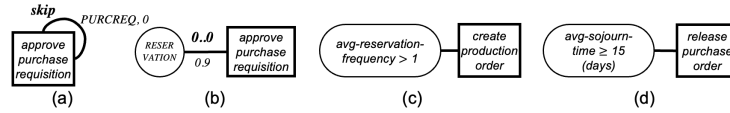
Fig. 5: OCCGs representing the constraints of the production process.

We monitor the process using three OCELs extracted from the SAP ERP system. Each event log contains events of different time windows; L_{prod}^{Jan22}, L_{prod}^{Feb22}, and L_{prod}^{Mar22} contain events of Jan., Feb., and Mar. 2022[2]. Table 2 shows the monitoring result. For instance, *no reservation for PRA* and *excessive reservations per PO* are violated for the three months. *Skipping PRA* only is violated in the last two months, while *delayed RPO* is violated only for Feb. 2022.

Table 2: Monitoring results of the production process. ✓ denotes the violation.

Constraints	Event log		
	L_{prod}^{Jan22}	L_{prod}^{Feb22}	L_{prod}^{Mar22}
Skipping PRA		✓	✓
No reservation for PRA	✓	✓	✓
Excessive reservations per PO	✓	✓	✓
Delayed POR		✓	

6.3   Case Study: Procure-To-Pay (P2P) Process

Next, we explain a case study on the P2P process. The process involves five object types: *purchase requisition*, *material*, *purchase order*, *goods receipt*, and *invoice*. Fig. 6 shows a process model of the process.

We represent the following constraints using OCCGs:

---

[2] Event logs are publicly available at `https://github.com/gyunamister/ProPPa.git`.
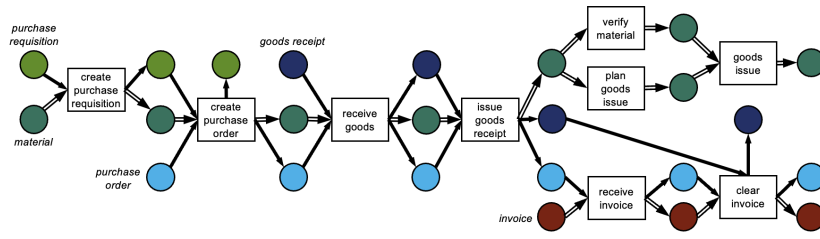
Fig. 6: P2P process: First, a purchase requisition is created with multiple materials. Next, a purchase order is created based on the purchase requisition and materials. Afterward, the materials are received and a goods receipt is issued. Then, the materials are verified and issued, and concurrently the invoice for the purchase order is received and cleared.

- *Concurrency between Verify Material (VM) and Plan Goods Issue (PGI)*; VM and PGI are usually not supposed to be concurrently executed. Fig. 7(a) represents the constraint.
- *Clearance of multiple invoices*; The execution of *clear invoice* should not involve multiple invoices at all. Fig. 7(b) represents the constraint.
- *Excessive materials per Purchase Order (PO)*; The execution of *create purchase order* should involve less than five materials on average. Fig. 7(c) represents the constraint.
- *Delayed Purchase Order Creation (POC)*; The average sojourn time of *create purchase order* should be less than three days. Fig. 7(d) represents the constraint.
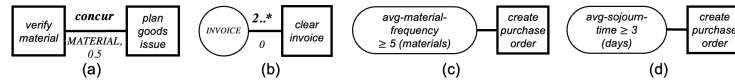
Fig. 7: OCCGs representing the constraints of the P2P process.

We monitor the process using three OCELs extracted from the SAP ERP system. Each event log contains events of different time windows; L_{p2p}^1 starting from 01-Aug-2021 and ending at 14-Oct-2021, L_{p2p}^2 starting from 15-Oct-2021 and ending at 18-Jan-2022, and L_{p2p}^3 starting from 01-Feb-2022 and ending at 16-May-2022. Table 3 shows the monitoring result. *Concurrency between VM and PGI* and *clearance of multiple invoices* are only violated in the first two time windows, whereas *Excessive materials per PO* and *delayed POC* are only violated in the last time window.

Table 3: Monitoring results of the P2P process. ✓denotes the violation.

Constraints	Event log		
	L_{p2p}^1	L_{p2p}^2	L_{p2p}^3
Concurrency between VM and PGI	✓	✓	
Clearance of multiple invoices	✓	✓	
Excessive materials per PO			✓
Delayed POC			✓

7   Conclusion

In this paper, we proposed an approach to process-level object-centric constraint monitoring. To this end, we first introduced object-centric behavioral metrics and defined OCCGs using the metrics. The proposed monitoring engine evaluates the constraints represented by OCCGs by analyzing OCELs. We have implemented the approach as a Web application and discussed two case studies.

This paper has several limitations. The suggested object-centric constraint graphs only represent the constraints selectively introduced in this work. More advanced constraints are not considered, e.g., ordering relations with the temporality (e.g., eventual or direct causality). Also, constraint graphs do not support timed constraints, e.g., no involvement of an object type during a specific period of time. In future work, we plan to extend the proposed approach to support more complete set of constraints, including more advanced constraints. Another interesting direction of future work is to apply the proposed approach to real-life business processes.

Acknowledgment

References

1. van der Aalst, W.M.P.: Object-centric process mining: Dealing with divergence and convergence in event data. In: Ölveczky, P.C., Salaün, G. (eds.) SEFM 2019. pp. 3–25
2. van der Aalst, W.M.P.: Process Mining - Data Science in Action. Springer (2016)
3. van der Aalst, W.M.P., Berti, A.: Discovering object-centric Petri nets. Fundam. Informaticae **175**(1-4), 1–40 (2020)
4. Adams, J.N., van der Aalst, W.M.P.: Precision and fitness in object-centric process mining. In: Ciccio, C.D., Francescomarino, C.D., Soffer, P. (eds.) ICPM 2021. pp. 128–135. IEEE (2021)
5. Awad, A., et al.: Runtime detection of business process compliance violations: an approach based on anti patterns. In: Wainwright, R.L., et al. (eds.) 30th ACM SAC. pp. 1203–1210 (2015)
6. Esser, S., Fahland, D.: Multi-dimensional event data in graph databases. J. Data Semant. **10**(1-2), 109–141 (2021)
7. Indiono, C., Mangler, J., Fdhila, W., Rinderle-Ma, S.: Rule-based runtime monitoring of instance-spanning constraints in process-aware information systems. In: Debruyne, C., et al. (eds.) OTM 2016. pp. 381–399 (2016)
8. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.P.: Compliance monitoring in business processes: Functionalities, application, and tool-support. Inf. Syst. **54**, 209–234 (2015)
9. 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.) RV 2011. pp. 131–146
10. Park, G., van der Aalst, W.M.P.: Action-oriented process mining: bridging the gap between insights and actions. Progress in Artificial Intelligence pp. 1–22 (2022)
11. Park, G., Adams, J.N., van der Aalst, W.M.P.: OPerA: Object-centric performance analysis. In: Ralyté, J., et al. (eds.) ER 2022. vol. 13607, pp. 281–292 (2022)
12. Ramezani, E., Fahland, D., van der Aalst, W.M.P.: Where did I misbehave? diagnostic information in compliance checking. In: Barros, A., Gal, A., Kindler, E. (eds.) BPM 2012. pp. 262–278
13. Weidlich, M., Ziekow, H., Mendling, J., Günther, O., Weske, M., Desai, N.: Event-based monitoring of process execution violations. In: Rinderle-Ma, S., et al. (eds.) BPM 2011. pp. 182–198