OC-DECLARE: Discovering Object-Centric Declarative Patterns with Synchronization

Aaron Küsters and Wil M.P. van der Aalst

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

Abstract. Real-world processes involve objects of different types that interact with each other, like customers, orders, and items. Accurately representing and analyzing these processes requires object-centric process mining techniques that do not force flattening processes to a single perspective. Most of the object-centric modeling, conformance checking, and discovery approaches presented so far capture little to no interaction between objects and their control flow. However, object synchronization is a fundamental necessity to depict such processes accurately. For example, in an order management process, an order should always be confirmed together with all the items it was placed with. In this paper, we propose an object-centric extension to the traditional DECLARE approach for imposing declarative process rules through a set of constraint template instantiations. Through the use of simple primitives, our approach, OC-DECLARE, can express new types of object-centric constraints. Moreover, the pattern-style representation allows also using and interpreting OC-DECLARE constraints individually, e.g., in prediction tasks. We provide a novel algorithm to automatically discover OC-DECLARE constraints from noisy object-centric event logs, including interactions between objects and synchronization behavior. We evaluate the mining approach on four publicly available object-centric datasets, including one real-life one, and identify patterns and constraints that cannot be found using previous object-centric discovery approaches.

Keywords: Object-Centric · Process Mining · Declarative Process Models · Process Discovery · Conformance Checking.

1 Introduction

Traditionally, process discovery approaches assume a fixed case notion in their input event data. The resulting process models are then interpreted for one case at a time. However, real-life processes often involve multiple objects of different types. As part of a paradigm shift towards object-centric process mining, many process modeling approaches combining multiple object perspectives have been proposed [1, 2, 6, 8, 11, 12]. For some of them, there are also corresponding discovery or conformance checking approaches [1, 2, 6, 8, 12, 15, 21]. However, most of them heavily underspecify object-centric processes. Many discovery approaches consider different object types in isolation during discovery, and simply

merge them into a single model, without addressing synchronization between objects [1, 8, 12].

In particular, the identities of objects are not tracked, and multiple objects of one type cannot be distinguished. Thus, these approaches are fundamentally unable to include *synchronization between objects*, for example, an activity always occurring for *combinations of objects*, based on explicit relationships between the objects or prior occurrences in events [12]. To the best of our knowledge, no previously proposed object-centric discovery approach can discover different types of synchronization behavior across objects and sets of objects. Our approach, OC-DECLARE, inspired by the traditional DECLARE method [17], addresses this gap using three simple concepts of object involvements (All, Each, and Any). These three constructs are needed to address object multiplicity, i.e., that events can be associated with multiple objects of one type. For example, All correlates events based on all the involved objects of one type, while Each builds one set of correlated events for each involved object of that type, individually. In this paper, we introduce OC-DECLARE as a process model and constraint language, and additionally address conformance checking and discovery of OC-DECLARE constraints. Figure 1 shows an OC-DECLARE model of an order management process. For example, the edge between the activities `Place Order` and `Confirm Order` specifies the following constraint: For all `Place Order` events e and every order object associated with e, there has to be a `Confirm Order` event e' afterwards, involving the order, all the items associated with e, and one employee assigned to a customer involved in e (through object relationships). Through the OC-DECLARE discovery approach introduced in this paper, object-centric constraints capturing object set interactions can automatically be mined from data. For instance, given a corresponding object-centric event log as input, our approach can automatically discover the aforementioned edge between `Place Order` and `Confirm Order`.

OC-DECLARE is inspired by the classical DECLARE approach [17]. Declarative process models, like DECLARE [17] or DCR graphs [13], describe a process

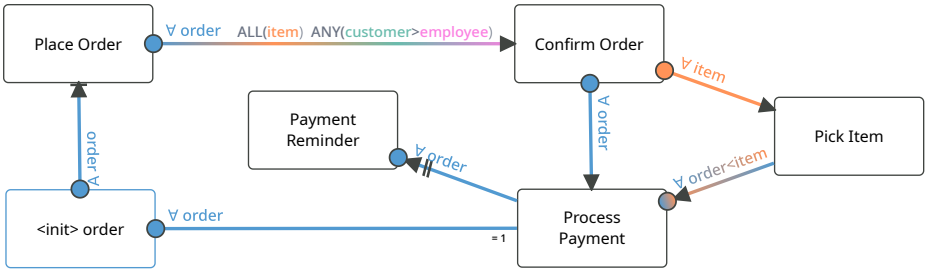

Fig. 1. An OC-DECLARE model for an order management process. The model contains seven individual constraints, specified by arcs between activities. Arcs are annotated with object types and their involvement mode, either Each (∀), All, or Any.

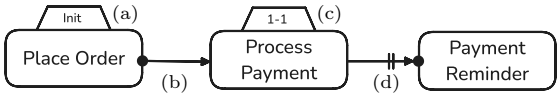

Fig. 2. A simple DECLARE model imposing the following constraints: (a) `Place Order` happens initially (b) when `Place Order` occurs, `Process Payment` always happens afterwards (c) `Process Payment` happens exactly once, and (d) when `Payment Reminder` occurs, `Process Payment` must not have occurred before.

in terms of constraints that forbid certain behavior, instead of describing all possible allowed behavior. As such, declarative process models can express loosely structured processes more easily. Figure 2 shows an example DECLARE model, including a subset of the constraints imposed by the OC-DECLARE model from Figure 1 for the object type order. In DECLARE, binary constraints (i.e., involving two activities) are modeled as arcs between the corresponding activity nodes, while unary constraints are annotated on the top of the activity node. DECLARE constraints can also be represented using textual templates. For example, in Figure 2, (a) can be written as Init(`Place Order`).

OC-DECLARE is an object-centric extension of the traditional DECLARE, that allows modeling constraints involving multiple object types. To that end, OC-DECLARE is not limited to simply combining DECLARE constructs across object types, but can also capture interactions between them and handle events associated with multiple objects. Meanwhile, the approach is still fundamentally simple, with models only consisting of annotated arcs between activities. This also enables using constraints individually in isolation using text-based templates. For instance, classification or pattern detection techniques like [19], building on top of the traditional DECLARE, could be extended to an object-centric setting using OC-DECLARE. Apart from introducing OC-DECLARE, we also present how existence-based OC-DECLARE constraints can be automatically discovered. Through defining a preference relation, our discovery approach discovers the set of maximal existence constraints. Moreover, certain uninteresting constraints, for example, based on resource-like object types, can be filtered out automatically.

This paper is organized as follows. We first discuss related work in Section 2. Next, in Section 3, we recall object-centric event logs. We then introduce the syntax of OC-DECLARE in Section 4, together with their semantics and the conformance checking of constraints. Next, we introduce our discovery approach for OC-DECLARE constraints in Section 5. We present the results of our experimental evaluation in Section 6, before we conclude this paper in Section 7.

2 Related Work

In this section, we provide an overview of object-centric process models, as well as their conformance checking and discovery approaches. Afterwards, we discuss the current state of object-centric discovery, highlighting shortcomings.

There are a few approaches for object-centric process models [1,2,6,8,11,12]. Automatic discovery has been addressed for [1, 2, 6, 8], based on input object-centric event data [6, 8, 9, 21]. For [1, 2, 12] there are conformance-checking approaches [2, 12, 15]. In the following, we briefly describe object-centric process models for which discovery or conformance checking has been addressed.

Object-Centric Petri Nets (OCPN), introduced in [1], extend traditional Petri nets by coloring places and tokens with an *object type*. To model activities involving multiple objects of a type, variable arcs are introduced, which allow producing or consuming an arbitrary number of tokens. No guards or limitations on the combination of tokens consumed by transitions are imposed. The authors also present a discovery approach, discovering Petri nets for each object type separately and then merging them together [1]. Moreover, there is an alignment-based conformance checking approach for object-centric Petri nets [15]. In [9], the authors describe how to detect *silent objects*, corresponding to common object combinations across activity executions, that can be integrated in OCPN-like models. However, the approach does not differentiate between different object involvement patterns. Furthermore, the approach constructs the smallest combinations of object types that uniquely identify events, instead of discovering the most precise synchronizations, following an inverse procedure to our approach. Based on prior work, adding identifiers to Petri nets [20], *Object-centric Petri nets with IDentifiers* (OPID) have been introduced in [12]. Notably, OPIDs allow specific synchronization of tokens corresponding to a combination of object instances and object instance sets. The synchronization is, however, only based on subset and not exact synchronization (e.g., that all items confirmed for an order were also involved in its placement, but not the other way around). However, exact synchronization, while not expressible in OPIDs directly, can be enforced when calculating alignments under the assumption that remaining object tokens could not proceed and complete without an exact synchronization [12].

Object-Centric Process Trees (OCPT) were introduced in [8], annotating each leaf node of a process tree with the involved object types and a set of relations. The authors also sketch a discovery technique based on discovering a traditional process tree from a pre-processed directly-follows graph. Synchronization is not addressed in the modeling or discovery approach, and each object is considered independently per executed tree node.

Object-Centric Behavioral Constraints (OCBC) models combine behavioral constraints, inspired by DECLARE [17], with object modeling techniques, similar to ER or UML [2,4]. As a result, OCBC models are made up of two submodels: A Behavioral Constraint Model with activities as nodes and a Class Model with object types as nodes. Connections between both models are used to specify cardinality constraints and event correlation. Two types of behavioral event correlations are possible: Directly through a common object type or indirectly, through a relation between two object types. This coupling between the behavioral model and the class model means that the behavioral model can only be interpreted in combination with the class model. Exact or subset synchronization is not directly expressible in OCBC without additional restrictions, as behavioral

constraints are interpreted with ANY-involvement semantics (e.g., that after an order was created, there is a `pick item` event with *any* of the order's items). A conformance relation between OCBC models and object-centric event logs was presented in [2]. Additionally, a discovery approach has been developed [21], also handling infrequent behavior.

Object-Centric DCR graphs (OC-DCR) extend DCR graphs with constructs related to object instance spawning, as well as flat 1:n and m:n synchronization between all objects of specified object types [6]. In particular, OC-DCR graphs are interpreted in the context of a set of objects, with each object instance having its own subgraph. The authors also present a discovery method for OC-DCR graphs based on a traditional DCR discovery algorithm [6].

In summary, most of the object-centric discovery approaches do not consider synchronization and advanced interactions or relationships between objects. While OCBC can express constraints spanning multiple object types using object-to-object relationships, synchronization is not fully addressed in the models and discovery [21]. In particular, fundamentally OCBC only considers behavioral constraints with ANY-semantics and does not explicitly handle object multiplicity, i.e., events involving multiple objects of one type [21]. Still, in combination with other restrictions, e.g., that an activity can only occur once for an object, subset and exact synchronization can be modeled by combining behavioral and class-level constructs. In OC-DCR, object multiplicity is also not addressed [6]. However, the approach allows for one-to-many and many-to-many synchronizations that are interpreted between all instances of specified object types, which requires interpreting models in the context of a clear object set or hierarchy [6]. As such, the approach assumes hierarchical object types, forming an implicit case notion. In general, there is not always such a hierarchical case notion in object-centric event data.

Overall, none of these object-centric discovery approaches fully handle object multiplicity and the resulting synchronization across multiple object sets.

3 Preliminaries

In this section, we introduce object-centric event logs and related notations used throughout this paper. We begin by introducing the types of entities we consider.

Definition 1. *We use the following pairwise disjoint universes:*

- $\mathcal{E}$ *Universe of events* *(e.g., e_1)*
- $\mathcal{O}$ *Universe of objects* *(e.g., o_1)*
- $\mathcal{ET}$ *Universe of event types (i.e., activities)* *(e.g., `Confirm Order`)*
- $\mathcal{OT}$ *Universe of object types* *(e.g., `order`)*
- $\mathbb{T}$ *Universe of timestamps* *(e.g., `25.11.19 13:00`)*

Next, we introduce Object-Centric Event Logs (OCEL). An OCEL consists of a set of objects and a set of events with specified types. There can be relationships between events and objects (E2O) as well as objects and objects (O2O). The presented formalization corresponds to a subset of the OCEL 2.0 specification [5].

Table 1. An example OCEL of an order management process, containing the following object types and objects: `customer` (o_1), `order` (o_2, o_{10}), `item` ($o_3, o_4, o_5, o_{11}, o_{12}$), and `employee` ($o_6, o_7, o_8, o_9$). On the left, the events, their event types (activities), timestamps, and E2O relations are shown. The object instances are colored according to their type. The table on the right shows the O2O relationships of objects.

Event	Activity	Timestamp	E2O Objects
e_1	Place Order	01.01.25 10:00	$\{o_1, o_2, o_3, o_4, o_5\}$
e_2	Confirm Order	01.01.25 13:00	$\{o_2, o_3, o_4, o_5, o_6, o_7\}$
e_3	Pick Item	01.01.25 16:00	$\{o_3, o_8\}$
e_4	Pick Item	02.01.25 09:00	$\{o_4, o_9\}$
e_5	Pick Item	02.01.25 12:00	$\{o_5, o_9\}$
e_6	Place Order	02.01.25 16:00	$\{o_1, o_{10}, o_{11}, o_{12}\}$
e_7	Confirm Order	02.01.25 18:00	$\{o_1, o_{10}, o_{11}, o_{12}, o_6\}$
e_8	Process Payment	03.01.25 12:00	$\{o_1, o_2, o_7\}$

Object	O2O Objects
o_1	$\{o_6, o_7\}$
o_2	$\{o_1\}$
o_3	$\{o_2\}$
o_4	$\{o_2\}$
o_5	$\{o_2\}$
o_{10}	$\{o_1\}$
o_{11}	$\{o_{10}\}$
o_{12}	$\{o_{10}\}$

Definition 2. *Object-Centric Event Logs (OCEL) can be described as a tuple* $L = (E, O, R, type, time)$ *of the following components:*

- ***Events*** $E \subseteq \mathcal{E}$*, a set of events.*
- ***Objects*** $O \subseteq \mathcal{O}$*, a set of objects.*
- ***Relations*** $R \subseteq (E \times O) \cup (O \times O)$*, E2O and O2O relationships*
- ***Type Assignments****,* $type \colon E \cup O \to \mathcal{ET} \cup \mathcal{OT}$*, assigning types to events and objects, such that* $\forall_{e \in E}\ type(e) \in \mathcal{ET}$ *and* $\forall_{o \in O}\ type(e) \in \mathcal{OT}$
- ***Event Times****,* $time \colon E \to \mathbb{T}$*, assigning unique timestamps to each event. In particular, we assume that time is injective, totally ordering events.*

Table 1 shows an example OCEL of an order management process involving the object types `customer`, `order`, `item`, and `employee`.

Definition 3. *We introduce notational shorthands for important properties in the context of an OCEL* $L = (E, O, R, type, time)$*:*

- $E_L = E$*, for the events of* L
- $O_L = O$*, for the objects of* L
- $R_L = R$*, for the event-to-object and object-to-object relations in* L
- $type_L = type$*, for the event and object types*
- $time_L = time$*, for the event timestamps*
- $E_L^{et} = \{e \in E \mid type(e) = et\}$*, for the events of type* et *in* L
- $O_L^{ot} = \{o \in O \mid type(o) = ot\}$*, for the objects of type* ot *in* L
- $obj_L(x) = \{o \in O \mid (x, o) \in R\}$*, for the objects related to an* $x \in E \cup O$

Apart from direct associations of events and objects, we also want to handle indirect associations through O2O relationships. For example, through an O2O relationship, the `Pick Item` event e_3 from Table 1 could be linked not only to the `item` o_3, but also to the corresponding `order` object o_2. To achieve this, we introduce a set of transitive object types $\mathcal{OT}^* = \mathcal{OT} \cup (\mathcal{OT} \times \{>, <\} \times$

$\mathcal{OT}$), where $<$ or $>$ indicates the direction of the O2O relation. The previous example can then be written as $(\texttt{item}, >, \texttt{order}) \in \mathcal{OT}^*$. For an OCEL $L = (E, O, R, type, time)$, we define $obj_L^{ot}(e)$ for any $ot \in \mathcal{OT}^*$ and $e \in E_L$ as follows:

$$obj_L^{ot}(e) = \begin{cases} \{o \in O_L^{ot} \mid (e,o) \in R\}, & \text{if } ot \in \mathcal{OT} \\ \{o' \in O_L^{ot_2} \mid \exists_{o \in O_L^{ot_1}} (e,o) \in R \wedge (o,o') \in R\}, & \text{if } ot = (ot_1, >, ot_2) \\ \{o' \in O_L^{ot_2} \mid \exists_{o \in O_L^{ot_1}} (e,o) \in R \wedge (o',o) \in R\}, & \text{if } ot = (ot_1, <, ot_2) \end{cases}$$

Consider the example OCEL from Table 1. For the object type $ot = \texttt{item}$, $obj_L^{ot}(e_1) = \{o_3, o_4, o_5\}$. Similarly, for the transitive types $ot' = (\texttt{item}, >, \texttt{order})$ and $ot'' = (\texttt{customer}, <, \texttt{order})$, $obj_L^{ot'}(e_3) = \{o_2\}$ and $obj_L^{ot''}(e_1) = \{o_2, o_{10}\}$.

4 OC-DECLARE

In this section, we introduce OC-DECLARE formally. As OC-DECLARE does not rely on a case notion, that partitions events into unique sets, event correlation needs to be approached differently. To this end, we define multiple types of *event filters*, functions that take a set of events and return a subset of the input set.

Event Filters We start by introducing event filters based on activities.

Definition 4. *Given an OCEL L, an input set of events $E \subseteq E_L$, and an event type $et \in \mathcal{ET}$, we define $filter_{et}^{ACT}(E) = \{e \in E \mid type_L(e) = et\}$.*

For example, considering L from Table 1, $filter^{\text{Act}}_{\texttt{Pick Item}}(E_L) = \{e_3, e_4, e_5\}$. Next we specify two types of event filters for a given set of objects O, that keep events if they involve all the objects in O (ALL), or at least one of them (ANY).

Definition 5. *Let L be an OCEL. Given a set of objects $O \subseteq O_L$, the filter functions $filter_O^{ALL}(E) = \{e \in E \mid O \subseteq obj_L(e)\}$ and $filter_O^{ANY}(E) = \{e \in E \mid O \cap obj_L(e) \neq \emptyset\}$ are defined for an input set of events $E \subseteq E_L$. Given a set of object sets $S \subseteq \mathcal{P}(O_L)$, we also write $filter_S^{ALL}(E) = \bigcap_{O \in S} filter_O^{ALL}(E)$ and $filter_S^{ANY}(E) = \bigcap_{O \in S} filter_O^{ANY}(E)$ for the intersection across all contained sets.*

For example, with L from Table 1 and $O = \{o_3, o_4, o_5\}$, $filter_O^{\text{ALL}}(E_L) = \{e_1, e_2\}$. Similarly, for $S = \{\{o_3, o_4, o_5\}, \{o_6, o_9\}\}$, $filter_S^{\text{ANY}}(E_L) = \{e_2, e_4, e_5\}$.

To model temporal relations, we introduce a set of arrow types, *Arrows*, inspired by the classical DECLARE [17]. *Arrows* contains the following elements:

- Associated with (AS), responded-existence in DECLARE
- Eventually followed by (EF), response in DECLARE
- Eventually preceded by (EP), precedence in DECLARE
- Directly followed by (DF), chain response in DECLARE
- Directly preceded by (DP), chain precedence in DECLARE

Using arrow types, events can be filtered regarding a reference timestamp. Other DECLARE arrow types (e.g., alternate-precedence) could also be integrated.

Definition 6. *Let L be an OCEL. We write $ts(e) = time_L(e)$ as a shorthand. For a reference timestamp $t \in \mathbb{T}$, and an arrow type $ar \in Arrows$ the following time-based filter function is defined for an input set of events $E \subseteq E_L$:*

$$filter^{TIME}_{t,ar}(E) = \begin{cases} E, & \text{if } ar = \bullet\!\!-\!\!- \\ \{e \in E \mid ts(e) > t\}, & \text{if } ar = \bullet\!\!-\!\!\rightarrow \\ \{e \in E \mid ts(e) < t\}, & \text{if } ar = \bullet\!\!\leftarrow\!\!- \\ \{e \in E \mid ts(e) > t \wedge \nexists_{e' \in E}\ ts(e) > ts(e') > t\}, & \text{if } ar = \bullet\!\!-\!\!\rightarrow\!\!| \\ \{e \in E \mid ts(e) < t \wedge \nexists_{e' \in E}\ ts(e) < ts(e') < t\}, & \text{if } ar = |\!\!\leftarrow\!\!\bullet \end{cases}$$

For example, given the OCEL L from Table 1, $t =$ 02.01.25 16:00, and $ar = \bullet\!\!-\!\!\rightarrow$, $filter_{t,ar}(E_L) = \{e_7, e_8\}$. With $ar' = \bullet\!\!-\!\!\rightarrow\!\!|$, $filter_{t,ar'}(E_L) = \{e_7\}$.

Notice that the composition of all filter function types, except $filter^{\text{TIME}}_{t,ar}$, are commutative. This is because $filter^{\text{TIME}}_{t,ar}$ can filter events based on directly-follows or directly-precedes relationships of the input event set.

Syntax We can next define the syntax of OC-DECLARE constraints formally. Constraints are arcs between activity nodes. Each arc has one *source activity* and one *target activity*. As an extension to standard DECLARE [17], each arc also specifies upper (n_{max}) and lower (n_{min}) bounds on how many matching events of the target activity should exist.

Definition 7. *A tuple $(ar, s, t, oi, n_{min}, n_{max})$ is an* OC-DECLARE constraint *consisting of:*

- *An arrow type $ar \in Arrows$*
- *A source activity $s \in \mathcal{A}$*
- *A target activity $t \in \mathcal{A}$*
- *A partial object involvement function $oi \colon \mathcal{OT}^* \not\to \{\textsc{Each}, \textsc{All}, \textsc{Any}\}$*
- *$n_{min} \in \mathbb{N}_0$*
- *$n_{max} \in \mathbb{N}_0 \cup \{\infty\}$, where ∞ represents that there is no maximum value*

Additionally, we write $Each_{oi} = \{ot \in \text{dom}(oi) \mid oi(ot) = \textsc{Each}\}$, $All_{oi} = \{ot \in \text{dom}(oi) \mid oi(ot) = \textsc{All}\}$, and $Any_{oi} = \{ot \in \text{dom}(oi) \mid oi(ot) = \textsc{Any}\}$.

Intuitively, the object involvements allow specifying the following conditions for an object type ot and a reference event e: (Each) For each ot object o involved in e, the specified number of target events involving o should occur. (All) For all ot objects O involved in e, the specified number of target events involving all of O should occur. (Any) For all ot objects O involved in e, the specified number of target events involving at least one of O should occur.

If in a given OCEL L, all events of the source activity s are only ever associated with at most one object of a specified object type, the object involvements Each, All, and Any are equivalent for this object type.

Before we define the semantics of constraints formally, we first present two examples. Consider an OC-DECLARE constraint $D = (ar, s, t, oi, n_{min}, n_{max})$

with $ar = \bullet\!\!\rightarrow$, $s =$ `Place Order`, $t =$ `Confirm Order`, $Each_{oi} = \{$`order`$\}$, $All_{oi} = \{$`item`$\}$, $Any_{oi} = \{($`customer`$, >,$ `employee`$)\}$, $n_{min} = 1$, and $n_{max} = \infty$. D is shown graphically in Figure 3, where a filled circle indicates the source activity. Intuitively, D expresses that for every `Place Order` event e there should be (at least) one `Confirm Order` event e' afterwards for each `order` in e, such that e' involves the `order`, all the items in e, and (at least) one of the `employee` objects assigned to the `customer` in e (through an O2O relationship).

Fig. 3. An OC-DECLARE constraint arc between `Place Order` and `Confirm Order`.

As a second example, consider $D' = (ar, s, t, oi, n_{min}, n_{max})$ with $ar = \leftarrow\!\!\bullet$, $s =$ `Process Payment`, $t =$ `Pick Item`, $Each_{oi} = \{($`order`$, <,$ `item`$)\}$, $All_{oi} = \{\}$, $Any_{oi} = \{$`employee`$\}$, $n_{min} = 1$, and $n_{max} = \infty$. D' is shown graphically in Figure 4. Intuitively, D' expresses that for every event of type `Process Payment` e there should be at least one event `Pick Item` e' before e for each of the items related to the `order` in e (through O2O relationships), such that e' involves the corresponding `item`, and at least one of the `employee` objects involved with e. Note, that by using Each object involvement instead of All, D' specifies that there has to be one event for each `item` individually, but not necessarily one event involving all of them together.

Fig. 4. An OC-DECLARE constraint arc between `Process Payment` and `Pick Item`.

Similar to traditional DECLARE, OC-DECLARE constraints can also be represented in text-based syntax. For example, `EF(Place Order, Confirm Order, Each(order), All(item), Any(customer>employee),1,`∞`)` represents D. The arc D' can be represented as `EP(Process Payment, Pick Item, Each(order<item), Any(employee),1,`∞`)`.

While there is no clear definition of subset and exact synchronization concepts for declarative models, OC-DECLARE can express common subset and exact synchronization patterns using All. For example, `EP(Confirm Order, Place Order, Each(order), All(item),1,`∞`)` specifies that for `Confirm Order` events e there must be a `Place Order` event for the `order` involving *at least* all the items in e and possibly more. For exact synchronization, a second constraint in the reverse direction can be considered in addition, effectively enforcing that there

is a "synchronized" event pair of specified types with the same set of objects for that type. By design, subset synchronization in OC-DECLARE is based on the source activity, as only events of the source activity actually impose constraints.

Semantics and Conformance Checking Next, we introduce the semantics of OC-DECLARE constraints. In particular, in the context of an OCEL L, we define when a given event $e \in E_L$ satisfies the constraint.

Definition 8. *Let $D = (ar, s, t, oi, n_{min}, n_{max})$ be an OC-DECLARE constraint with $Each_{oi} = \{ot_1, \ldots, ot_n\}$. In the context of an OCEL L, we define when an event $e \in E_L^s$ satisfies D (written as $e \models_L D$):*

$$e \models_L D \Leftrightarrow \forall_{o_1 \in obj_L^{ot_1}(e), \ldots, o_n \in obj_L^{ot_n}(e)} \; n_{min} \leq |f(E_L)| \leq n_{max}$$

$$\text{where } f = filter_t^{ACT} \circ filter_{time_L(e),ar}^{TIME} \circ filter_{\{o_1,\ldots,o_n\}}^{ALL} \circ filter_{\{obj_L^{ot}(e) | ot \in All_{oi}\}}^{ALL} \circ filter_{\{obj_L^{ot}(e) | ot \in Any_{oi}\}}^{ANY}$$

For events of different activities $e' \in E_L \setminus E_L^s$, we say that D trivially holds and also write $e' \models_L D$.

Considering the previously introduced example OC-DECLARE constraints D (Figure 3) and D' (Figure 4) and the example OCEL L from Table 1, we can conclude that $e_1 \models_L D$, $e_6 \models_L D$, but $e_8 \not\models_L D'$, as there is no corresponding `pick item` event for each `item` of the `order`, involving at least one `employee` also present in e_8.

The semantics defined in Definition 8 describe when an event satisfies an OC-DECLARE constraint. Through this per-event interpretation, confidence scores of OC-DECLARE arcs for a complete input OCEL can be calculated.

Definition 9. *For an OCEL L and an OC-DECLARE constraint $D = (ar, s, t, oi, n_{min}, n_{max})$, the* confidence *of D in L is $conf_L(D) = \frac{|\{e \in E_L^s \mid e \models_L D\}|}{|E_L^s|} \in [0, 1]$*

The conformance value is 1 exactly when all events of the source activity satisfy the constraint, and 0 exactly when no event satisfies the constraint. A global confidence score for a set of OC-DECLARE constraints DS can be given by the fraction of events that fulfill all constraints: $gconf_L(DS) = \frac{|\{e \in E_L \mid \forall_{D \in DS}\ e \models_L D\}|}{|E_L|}$. The language of an OC-DECLARE model (i.e., a set of OC-DECLARE constraints) DS is then simply the set of all OCELs L with $gconf_L(DS) = 1$.

More fine-grained diagnostics are also possible: For example, a constraint with `item` as EACH might be violated for a `confirm order` event e if it is not followed by a `pick item` event for only one of the items associated with e. As advanced diagnostics, the `item` object which was not picked can be identified.

Including Object Constraints So far, we focused only on constraints based on correlation of events. However, the classical DECLARE [17] as well as more recent approaches like OCBC [2] also allow modeling other constraints. We differentiate three subtypes: (1) *Event cardinality constraints*, specifying that at

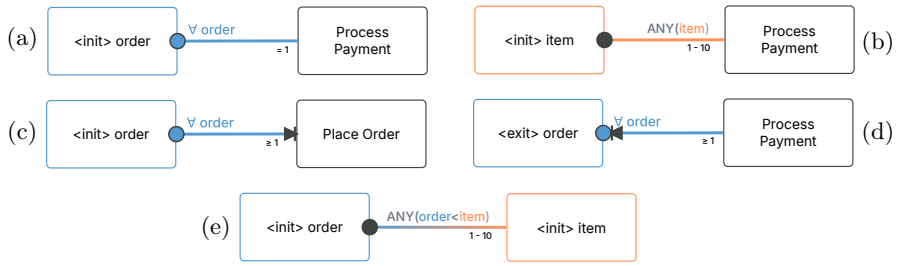

Fig. 5. Using artificially added init and exit events, the following constraints can be expressed: (a) Every `order` is paid exactly once (b) Every order placement is associated with 1–10 `item` (c) Every `order` is placed initially (d) Every `order` ends by being paid (e) Every `order` is associated with 1–10 items.

most or least n events of a specified activity occur per object. (2) *Object cardinality constraints*, specifying how many objects of one type can be associated with an event or object of another type. (3) *Initial/final activity constraints*, specifying the activity of the first or last event associated with an object. For example, specifying that exactly one `Process Payment` event should happen for each `order` is an event cardinality constraint. Specifying how many objects of type `item` should be involved in a `Place Order` event is an object cardinality constraint. Prescribing that every `order` object should at first be involved in a `Place Order` event is an initial/final activity constraint.

Integrating these types of constraints into OC-DECLARE is achieved using an implicit OCEL pre-processing step: Given an input OCEL L, the transformed OCEL L' is an extension of L, retaining all its events and objects with the same types, the same timestamps, and all relations. Additionally, for each object $o \in O_L$ with type $type_L(o) = \texttt{T}$, two new events e^o_{init} and e^o_{exit} are included in $E_{L'}$, with $(e^o_{init}, o), (e^o_{exit}, o) \in R_{L'}$, $type_{L'}(e^o_{init}) = \texttt{<init> T}$, $type_{L'}(e^o_{exit}) = \texttt{<exit> T}$, $time_{L'}(e^o_{init}) = \min\{time_L(f) \mid f \in E_L \wedge o \in obj_L(f)\} - \varepsilon$, and $time_{L'}(e^o_{exit}) = \max\{time_L(f) \mid f \in E_L \wedge o \in obj_L(f)\} + \varepsilon$, where $\varepsilon > 0$ can be arbitrarily small. These events correspond to the creation (init) and end (exit) of the object o. Figure 5 shows how these artificial events can be used with ●—, ●→, and ●← to express the three aforementioned constraint types.

While this lightweight extension allows modeling additional types of constraints, they are not required for the general OC-DECLARE approach as presented in this paper. For conformance checking and discovery, this pre-processing can simply be done beforehand for any input OCEL. For playing out or executing OC-DECLARE models, the artificial activities need to be considered according to their semantics (i.e., occurring before/after the first/last real event involving this object), but are afterwards removed from the output log.

Counts and Negated Constraints By allowing general n_{min} and n_{max} count limits for OC-DECLARE constraints, the negations of constraints can also trivially be represented. We consider *existence constraints* as all constraints with

$n_{min} = 1$ and $n_{max} = \infty$. Independent of the arrow type, the negation of an existence constraint has all the same components but instead the count limits $n_{min} = 0$ and $n_{max} = 0$. For example, the traditional DECLARE constraint NotResponse(s, t) can be represented with the normal response arrow type ($\bullet\!\!\rightarrow$) and $n_{min} = n_{max} = 0$.

5 Discovery

In this section, we describe how existence OC-DECLARE constraints (i.e., with counts $n_{min} = 1$ and $n_{max} = \infty$) can be automatically discovered from object-centric event logs with noise. A similar but inverse approach can also be developed for the negation of such constraints. Moreover, object constraints, can be discovered analogously to existing techniques (e.g., [21]).

For declarative models, discovery can often easily be implemented as a brute-force approach, starting from a fully constrained model, as done in [16] or [7]. A brute-force approach can also be used for OC-DECLARE, however as object types add significantly more possible instantiations of constraints, this might prove challenging in practice. Declarative discovery techniques often produce a huge number of constraints, making the resulting model overly complex [18].

To mitigate the problem of superfluous constraints and to enable fast discovery, we specify a subset of *preferred* constraints that should be automatically discovered. We can further distinguish between constraints being directly implied by others (i.e., *implied*) or simply defined as being preferred over others.

We start by noting observations on implied OC-DECLARE constraints in Lemma 1 (based on the arrow type) and Lemma 2 (based on object types): An OC-DECLARE existence constraint with an arrow type $ar \neq \bullet\!\!-$ always implies at least one other constraint. For example, if $ar = \bullet\!\!\rightarrow$, the same constraint with $\bullet\!\!-$ is implied because it imposes less strict filters on the target events. Intuitively, if there is at least one target event occurring afterwards ($\bullet\!\!\rightarrow$), there also is one target event occurring before *or* after ($\bullet\!\!-$).

Lemma 1. *Let L be an OCEL and let $D = (ar, s, t, oi, 1, \infty)$ and $D' = (ar', s, t, oi, 1, \infty)$ be two OC-DECLARE constraints. Then, the implication $e \models_L D \Rightarrow e \models_L D'$ holds for the following combinations of ar and ar': (1) $ar \neq \bullet\!\!-$ and $ar' = \bullet\!\!-$, (2) $ar = \bullet\!\!\rightarrow\!\!\blacktriangleright$ and $ar' = \bullet\!\!\rightarrow$, or (3) $ar = \blacktriangleleft\!\!\bullet\!\!-$ and $ar' = \blacktriangleleft\!\!\bullet\!\!-$.*

Furthermore, for a given arrow type $ar \in \{\bullet\!\!-, \bullet\!\!\rightarrow, \blacktriangleleft\!\!\bullet\!\!-\}$, stricter object involvements always lead to stricter target event filters and can only reduce the number of target events. Thus, for any given existence constraint, the same constraint with more lax object involvement (e.g., subsets of All, Each, Any or an object type in Each instead of All) is directly implied by it.

Lemma 2. *Consider an arrow type $ar \in \{\bullet\!\!-, \bullet\!\!\rightarrow, \blacktriangleleft\!\!\bullet\!\!-\}$. Let L be an OCEL. Then the following implications hold for all events $e \in E_L^s$:*

- *If a constraint $D = (ar, s, t, oi, 1, \infty)$ holds for e, then all constraints $D' = (ar, s, t, oi', 1, \infty)$ with $All_{oi'} \subseteq All_{oi}$, $Each_{oi'} \subseteq Each_{oi}$, and $Any_{oi'} \subseteq Any_{oi}$ also hold for e (i.e., $e \models_L D \Rightarrow e \models_L D'$).*

- *Let* $ot \in \mathcal{OT}^*$. *If* $D = (ar, s, t, oi, 1, \infty)$ *with* $ot \in Each_{oi}$ *holds for* e, *then* $D' = (ar, s, t, oi', 1, \infty)$ *with* $All_{oi'} = All_{oi}$, $Each_{oi'} = Each_{oi} \setminus \{ot\}$, *and* $Any_{oi'} = Any_{oi} \cup \{ot\}$ *also holds for* e *(i.e.,* $e \models_L D \Rightarrow e \models_L D'$*).*
- *Let* $ot \in \mathcal{OT}^*$. *If* $D = (ar, s, t, oi, 1, \infty)$ *with* $ot \in All_{oi}$ *holds for* e, *then* $D' = (ar, s, t, oi', 1, \infty)$ *with* $All_{oi'} = All_{oi} \setminus \{ot\}$, $Each_{oi'} = Each_{oi} \cup \{ot\}$, *and* $Any_{oi'} = Any_{oi}$ *also holds for* e *(i.e.,* $e \models_L D \Rightarrow e \models_L D'$*).*

Based on these observations, we define a preference relation between OC-DECLARE constraints, determining which constraints to prefer over others:

Definition 10. *Let* $D = (ar, s, t, oi, 1, \infty)$ *and* $D' = (ar', s', t', oi', 1, \infty)$ *be two OC-DECLARE constraints. We prefer* D *over* D' *(written as* $D' \preceq D$*) if* $s = s'$, $t = t'$, $All_{oi'} \subseteq All_{oi}$, $Each_{oi'} \subseteq All_{oi} \cup Each_{oi}$, *and* $Any_{oi'} \subseteq All_{oi} \cup Each_{oi} \cup Any_{oi}$. *When* $D \preceq D'$ *and* $D' \not\preceq D$, *we say that* D *is strictly preferred over* D' *(*$D' \prec D$*). Moreover, if* $D \preceq D'$ *and* $D' \preceq D$, *we still strictly prefer* D *over* D' *(i.e.,* $D' \prec D$*) if either: (1)* $ar \neq$ ●— *and* $ar' =$ ●—, *(2)* $ar =$ ●→| *and* $ar' =$ ●→, *or (3)* $ar =$ ●←● *and* $ar' =$ ●←.

Put into words, for the same reference and target activity, constraints are preferred if they have stricter object involvement requirements. Moreover, if they impose the same object involvement requirements, as secondary criteria, stricter event relationships (i.e., arrow types) are preferred.

Notice, that in contrast to Lemma 1 and Lemma 2, strict preference ($\prec$) does not correspond to implied constraints. For example, an eventually-follows constraint (●→) is preferred over a directly-follows one (●→|) if the former has a stricter object involvement, although the former does not imply the latter.

Leveraging the implications from Lemma 2, a discovery algorithm for OC-DECLARE constraints is outlined in Algorithm 1. It is inspired by the classical Apriori algorithm [3], exploiting monotonicity properties of the OC-DECLARE preference relation. Given a noise threshold $0 \leq \rho \leq 1$ and an arrow type $ar \in \{$●—, ●→, ●←●$\}$, it discovers the preferred set of constraints of the given arrow type for an OCEL L. The algorithm uses the following two subroutines:

- $combine(arc, arc')$, combines two arcs with the same source activity, target activity, and arrow type, by constructing the union of their object involvements, preferring more strict ones over others (e.g., *All* over *Each*).
- $reduce(Arcs)$, returns the subset of arcs not implied by others in the set

Given an OCEL L, let $Cons_L^{ar}$ be the set of all OC-DECLARE constraints with arrow type $ar \in Arrows$ that contain only object and event types from L, and involve at least one object type. For a noise threshold $0 \leq \rho \leq 1$, the subset of constraints with a confidence score of at least $1 - \rho$ is $SatCons_L^{ar,\rho} = \{D \in Cons_L^{ar} \mid conf_L(D) \geq 1 - \rho\}$. Given an OCEL L, a noise threshold ρ, and an arrow type $ar \in \{$●—, ●→, ●←●$\}$, the Algorithm 1 yields the maximal strictly preferred set of constraints $A = \{D \in SatCons_L^{ar,\rho} \mid \neg\exists_{D' \in SatCons_L^{ar,\rho}} (D' \neq D \wedge D \preceq D')\}$. It is easy to see, that Algorithm 1 only yields constraints of the correct arrow type that satisfy the confidence threshold. Additionally, per Lemma 2,

Algorithm 1 Behavioral OC-DECLARE Discovery

Input: OCEL L, noise threshold ρ, arrow type $ar \in \{\bullet\!\!-\!\!-, \bullet\!\!-\!\!\rightarrow, \bullet\!\!-\!\!\leftarrow\}$
Output: Set of OC-DECLARE Arcs A

$A \leftarrow \emptyset$
for $s, t \in \mathcal{ET}$ with $E^s_L \neq \emptyset$ and $E^t_L \neq \emptyset$ **do** ▷ Pair of activities present in L
  $X \leftarrow \emptyset$
  for $ot \in \mathcal{OT}^*$ with $\exists_{e \in E^s_L} obj^{ot}_L(e) \neq \emptyset$ **do**
    $arc_1 \leftarrow (ar, s, t, \{\}, \{\}, \{ot\}, 1, \infty)$ ▷ Try *any*
    if $conf_L(arc_1) \geq 1 - \rho$ **then**
      $X \leftarrow X \cup \{arc_1\}$
      $arc_2 \leftarrow (ar, s, t, \{ot\}, \{\}, \{\}, 1, \infty)$ ▷ Try *each* (only when *any* holds)
      if $conf_L(arc_2) \geq 1 - \rho$ **then**
        $X \leftarrow X \cup \{arc_2\}$
        $arc_3 \leftarrow (ar, s, t, \{\}, \{ot\}, \{\}, 1, \infty)$ ▷ Try *all* (only when *each* holds)
        if $conf_L(arc_3) \geq 1 - \rho$ **then**
          $X \leftarrow X \cup \{arc_3\}$
  do ▷ Construct and check arc combinations
    $Y \leftarrow \emptyset$
    for $arc, arc' \in X$ with $arc \neq arc'$ **do**
      $arc'' \leftarrow combine(arc, arc')$
      if $arc'' \notin X \wedge conf_L(arc'') \geq 1 - \rho$ **then** ▷ Check arc combination
        $Y \leftarrow Y \cup \{arc''\}$
    $X \leftarrow X \cup Y$
  while $Y \neq \emptyset$
  $A \leftarrow A \cup reduce(X)$ ▷ Add not-implied arcs to A
return A

the iterative combination of constraints constructs the maximal satisfied object involvements for the input parameters.

In addition to the preferred sets of constraints of a single arrow type, Algorithm 1 can also be used to discover the maximal *strictly preferred ($\prec$) set* of constraints across all arrow types (i.e., $\bullet\!\!-\!\!-$, $\bullet\!\!-\!\!\rightarrow$, $\bullet\!\!-\!\!\leftarrow$, $\bullet\!\!-\!\!\rightarrow\!\!|$, and $\bullet\!\!-\!\!\leftarrow\!\!\bullet$), as defined in Definition 10. For that, the algorithm is executed once with $ar = \bullet\!\!-\!\!-$, and for each discovered constraint, versions with stricter arrows (i.e., $\bullet\!\!-\!\!\rightarrow$, $\bullet\!\!-\!\!\leftarrow$, $\bullet\!\!-\!\!\rightarrow\!\!|$, and $\bullet\!\!-\!\!\leftarrow\!\!\bullet$) are checked and added, removing the less strict versions, as per Lemma 1. By Definition 10, this post-processing yields the set $B = \{D \in SatCons^{ar,\rho}_L \mid ar, ar' \in Arrows \wedge \neg\exists_{D' \in SatCons^{ar',\rho}_L} (D' \neq D \wedge D \prec D')\}$. Uninteresting constraints, for example, involving only resource-like object types (e.g., `employee`), can be removed by filtering the result of Algorithm 1 before testing other arrow versions.

While the discovery of negative constraints is outside the scope of this paper, intuitively, it corresponds to identifying minimal object involvements, as larger object involvements lead to negative constraints becoming less strict, while existence constraints become stricter.

6 Evaluation

In this section, we evaluate our approach. First, we investigate the runtime of the discovery algorithm and report the number of discovered constraints to evaluate its feasibility. Next, we perform a qualitative analysis on the discovered constraints, showcasing constraints not expressible or discoverable in prior work.

Table 2. Properties and discovery results for the object-centric datasets.

Dataset	Events	Event Types	Objects	Object Types	No O2O		Direct O2O	
					Duration	Count	Duration	Count
Logistics	321	7	132	9	0.06s	99	3.69s	140
P2P	1234	13	1341	6	0.03s	32	2.12s	53
Order	34303	22	5231	8	3.28s	116	32.88s	192
BPIC2017	2043141	24	34412	3	139.96s	102	255.84s	174

Our implementation based on Rust4PM [14], featuring discovery, conformance checking, and an interactive constraint editor, as well as our evaluation setup and results are available at `https://github.com/aarkue/oc-DECLARE`[1].

We applied our discovery approach with a noise threshold of $\rho = 0.2$ to four publicly available object-centric event logs: Three simulated datasets[2] and the real-life BPIC2017 dataset of a loan application process, based on [10]. For each dataset, the mean discovery runtime across 10 runs and the number of discovered constraints are shown in Table 2, together with general log properties. We ran our discovery approach in two variants, *No O2O*, i.e., without considering O2O relationships and *Direct O2O*, i.e., also considering direct O2O relationships. We did not consider both directions for O2O, as this leads to a significant increase in runtime. In our evaluation, we adapted Algorithm 1 for increased performance, including parallelization, to discover association constraints. Results which would not surpass the confidence threshold with a maximal event count of $n_{max} = 20$ are removed to exclude undesirable entries (e.g., only based on resource-like object types). Finally, for each remaining association constraints, stricter-arrow versions are tested and added as described in Section 5.

Overall, the runtimes across all considered logs range from below 1 second to slightly above 4 minutes, indicating the feasibility of our approach. Including object-to-object relationships comes with additional costs in increased runtime, as the number of possible arcs between activities increases significantly. Note that our discovery approach considers object-multiplicity and synchronization, and thus addresses a more complex problem than previously proposed object-centric discovery algorithms. Moreover, the number of discovered constraints is reasonable, not surpassing 200 for any tested log.

In Figure 6, we include some of the discovered constraints, inspired by the initial OC-DECLARE example from Figure 1. Notably, none of the constraints in Figure 6 can be expressed or discovered in prior work, as they combine multiple object involvements using Any, Each, and All. The discovered constraints demonstrate that our approach is indeed capable of discovering interesting and understandable constraints, including advanced synchronization behavior. For example, the discovered arc between `send package` and `package delivered` specifies that a package should be delivered after sending it, together with all the items and products, and by one of the employees who sent it.

[1] Version referenced: `https://doi.org/10.5281/zenodo.15554279`.

[2] Three simulated logs from `https://ocel-standard.org/event-logs/overview/`.

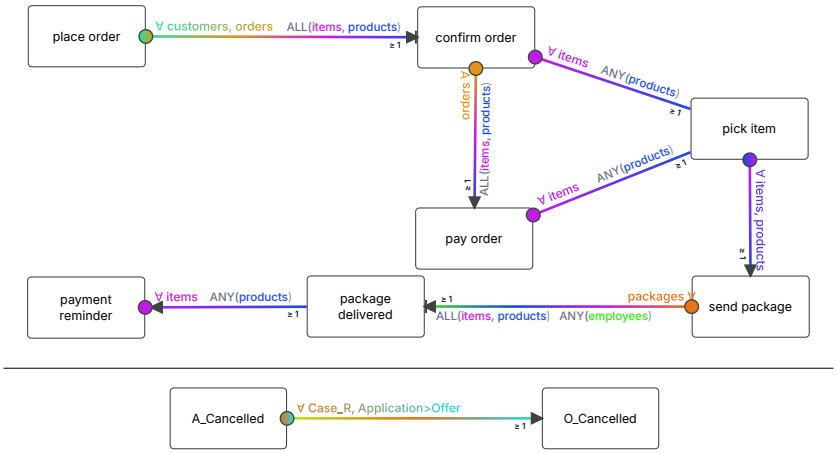

Fig. 6. On the top, a selection of discovered OC-DECLARE constraints for the Order OCEL without O2O relationships are shown. On the bottom, a single discovered constraint for the BPIC2017 OCEL with O2O relationships is included.

7 Conclusion

In this paper, we proposed three main contributions: First, we introduced an object-centric extension of the traditional DECLARE method. By modeling constraints as arcs between activities, a broad usage of OC-DECLARE constraint templates in other methods is possible. While keeping the basic concept rather simple, OC-DECLARE is still more expressive than previous (declarative) object-centric models. In particular, through differentiating three different types of object involvement (Each, All, and Any), OC-DECLARE can also handle object-multiplicity and supports object set synchronization. As a second contribution, we introduced a conformance-checking approach yielding a violation percentage. Moreover, we presented a discovery approach for an important subset of OC-DECLARE constraints, yielding a set of maximal existence constraints that are of particular interest. We evaluated our discovery approach on four publicly available object-centric datasets, including one real-life one. The results indicate that our discovery approach is feasible in terms of runtime and can indeed discover novel interesting and relevant constraints.

In future work, applications of OC-DECLARE constraints, e.g., to find indicating patterns for undesirable behavior or discover synchronization for other process models, should be investigated. Moreover, an underpinning in a formal logic could enable automatic reasoning over constraints. Lastly, a more sophisticated conformance checking approach, also quantifying partial mismatches of object sets, is of particular interest.

Acknowledgments. The authors gratefully acknowledge the German Federal Ministry of Education and Research (BMBF) and the state government of North Rhine-Westphalia for supporting this work as part of the NHR funding.

References

1. van der Aalst, W.M.P., Berti, A.: Discovering Object-centric Petri Nets. Fundam. Informaticae **175**(1-4), 1–40 (2020)
2. van der Aalst, W.M.P., Li, G., Montali, M.: Object-Centric Behavioral Constraints. CoRR (2017)
3. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In: VLDB. pp. 487–499. Morgan Kaufmann (1994)
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. LNCS, vol. 11675, pp. 139–156. Springer (2019)
5. Berti, A., Koren, I., Adams, J.N., Park, G., Knopp, B., Graves, N., Rafiei, M., Liß, L., genannt Unterberg, L.T., Zhang, Y., Schwanen, C.T., Pegoraro, M., van der Aalst, W.M.P.: OCEL (Object-Centric Event Log) 2.0 Specification. CoRR (2024)
6. Christfort, A.K.F., Rivkin, A., Fahland, D., Hildebrandt, T.T., Slaats, T.: Discovery of Object-Centric Declarative Models. In: ICPM. pp. 121–128. IEEE (2024)
7. Debois, S., Hildebrandt, T.T., Laursen, P.H., Ulrik, K.R.: Declarative process mining for DCR graphs. In: SAC. pp. 759–764. ACM (2017)
8. van Detten, J.N., Schumacher, P., Leemans, S.J.J.: Discovering Compact, Live and Identifier-Sound Object-Centric Process Models. In: ICPM. pp. 113–120. IEEE (2024)
9. van Detten, J.N., Schumacher, P., Leemans, S.J.J.: Object Synchronizations and Specializations with Silent Objects in Object-Centric Petri Nets. In: BPM. LNCS, vol. 14940, pp. 57–74. Springer (2024)
10. van Dongen, B.: BPI Challenge 2017 (2017), `https://data.4tu.nl/articles/dataset/BPI_Challenge_2017/12696884/1`
11. Fahland, D.: Describing Behavior of Processes with Many-to-Many Interactions. In: Petri Nets. LNCS, vol. 11522, pp. 3–24. Springer (2019)
12. Gianola, A., Montali, M., Winkler, S.: Object-Centric Conformance Alignments with Synchronization. In: CAiSE. LNCS, vol. 14663, pp. 3–19. Springer (2024)
13. Hildebrandt, T.T., Mukkamala, R.R.: Declarative Event-Based Workflow as Distributed Dynamic Condition Response Graphs. In: PLACES. EPTCS, vol. 69, pp. 59–73 (2010)
14. Küsters, A., van der Aalst, W.M.P.: Rust4PM: A Versatile Process Mining Library for When Performance Matters. In: BPM Demos. vol. 3758, pp. 91–95. CEUR-WS.org (2024)
15. Liss, L., Adams, J.N., van der Aalst, W.M.P.: Object-Centric Alignments. In: ER. LNCS, vol. 14320, pp. 201–219. Springer (2023)
16. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-guided discovery of declarative process models. In: CIDM. pp. 192–199. IEEE (2011)
17. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: Full Support for Loosely-Structured Processes. In: EDOC. pp. 287–300. IEEE (2007)
18. Slaats, T.: Declarative and Hybrid Process Discovery: Recent Advances and Open Challenges. J. Data Semant. **9**(1), 3–20 (2020)
19. Smedt, J.D., Deeva, G., Weerdt, J.D.: Mining Behavioral Sequence Constraints for Classification. IEEE Trans. Knowl. Data Eng. **32**(6), 1130–1142 (2020)
20. van der Werf, J.M.E.M., Rivkin, A., Polyvyanyy, A., Montali, M.: Data and Process Resonance - Identifier Soundness for Models of Information Systems. In: Petri Nets. LNCS, vol. 13288, pp. 369–392. Springer (2022)
21. Xiu, B., Li, G., Li, Y.: Discovery of Object-Centric Behavioral Constraint Models With Noise. IEEE Access **10**, 88769–88786 (2022)