Object-Centric Causal Nets

Lukas Liss, Caspar Mensing, and Wil M. P. van der Aalst

RWTH Aachen University, Aachen, Germany
{liss, wvdaalst}@pads.rwth-aachen.de, caspar.mensing@rwth-aachen.de

Abstract. Traditional process mining techniques are constrained by the use of a single case identifier. To address this, object-centric process mining was developed. Current object-centric process models like object-centric Petri nets are restricted by representational biases, for example concerning the concrete distribution of objects to their succeeding activities (abstracted by places) or optionally involved object types (abstracted using variable arcs). This paper presents four key contributions to object-centric process discovery, to overcome the mentioned representational biases. First, we define object-centric causal nets, allowing for the modeling of complex process behavior. Second, we introduce a baseline algorithm for mining these nets from object-centric event logs. Third, we provide a publicly available implementation. Fourth, we evaluated the implementation quantitative and qualitatively, demonstrating the algorithm's applicability across diverse event logs.

Keywords: Object-Centric Process Mining · Process Model · Process Discovery

1 Introduction

The real-world consist of complex processes comprising multiple subprocesses that operate on various objects from different types [8]. To handle the growing complexity, information systems are used to support these processes. They track the real process behavior of all the digitally traceable objects involved in the process. The goal of process discovery in information system management is to understand the real process and, based on that, improve the information system and the process itself [4]. To do so, it is essential to represent the real process as accurately as possible. It has been shown that an object-centric process perspective, which models the behavior of multiple interacting objects of a process at once, results in more accurate results because it is free of problems like divergence, convergence, and deficiency, compared to traditional process discovery [19]. Current object-centric process discovery approaches use design-oriented models like object-centric Petri net [21], which comes with a certain representational bias, which can lead to imprecise and undesirable process models, as demonstrated in the running example of an order handling process.

The running example process is depicted as an object-centric event log [2] in Table 1. This type of information can be extracted from information systems

Table 1: Excerpt of the object-centric event log of the running example.

ID	activity	time	order	container	box
1	a (check order)	1	o1		
2	b (register order)	2	o1		
3	b (register order)	3	o2		
4	c (get container)	4		c1	
5	s (send)	5	o1,o2	c1	
6	r (receive feedback)	6	o1,o2		
7	i (investigate)	7	o1		
8	n (no investigate)	8	o2		
9	b (register order)	9	o3		
10	d (get box)	10			b1
11	s (send)	11	o3		b1
12	r (receive feedback)	12	o3		
13	i (investigate)	13	o3		

to understand the process and then improve the information system. In the example process, orders can be checked first (see order *o1*) or directly registered (see order *o2*). Then they are *send* either using a container when multiple orders are shipped together (see e5) or in a box when only one order is sent (see e11). In the end, after receiving feedback for the orders sent together, the company investigates only one of the orders and does not investigate the others (see e7, e8, and e13). All three described process characteristics require artificial constructs or can not be modeled correctly using an object-centric Petri net, as one can see in the object-centric Petri net for the running example in Figure 1.

Object-centric Petri nets suffer from the following three representation biases: (1) They require artificial constructs like silent transitions (represented as black boxes) and places, which both have no correspondence in the event data [20]. (2) Object-centric Petri nets without duplicated labels (the only ones that can be discovered) overgeneralize optional object types by using variable arcs. This can be seen in the *send* activity, which is connected with variable arcs to the container and the box. This allows this activity to use multiple containers and boxes for the same *send* event, which is not represented in the process. Also, it is no longer modeled that individual orders are always sent via boxes due to the generalizing behavior of variable arcs. (3) The distribution of objects to parallel following events is not clear in object-centric Petri nets. In the example, precisely one of the orders that received feedback together is investigated, and the others are not. The Petri net also allows to investigate all, some or none.

To overcome these representational biases, we propose a new object-centric process model: object-centric causal nets. First, we define the syntax and semantics of object-centric causal nets. Second, we propose a discovery approach to mine them from event logs. Third, we provide a publicly accessible implementation of the miner. Finally, we perform a qualitative and quantitative evaluation using our implementation and publicly accessible event logs.

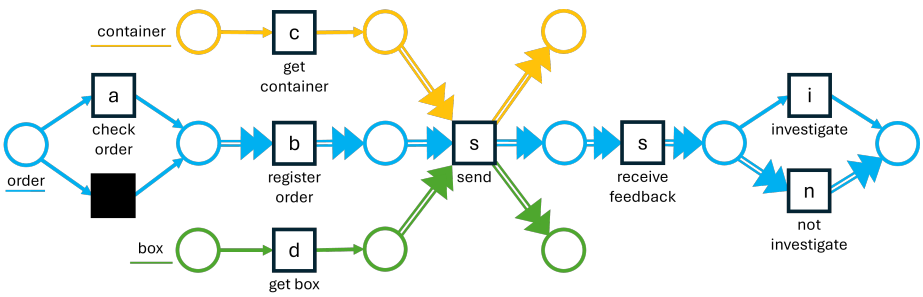


Fig. 1: Object-centric Petri net for the running example used to illustrate three limitations induced by the representational bias of object-centric Petri nets.

2 Related Work

Our approach belongs to the field of process model discovery [1] and is therefore part of the continuous improvement life cycle of information systems which is the overall goal of information system engineering research [24]. This involves a process-aware information system, where the differences between the real process and the designed process, as supported by the information system, can be discovered in a data-driven fashion [18,25]. Differences between the intended and real process behavior can then be used to improve the information system. Process discovery [23] is used in the information system engineering life-cycle to uncover the real process. The object-centric process discovery approach presented in this paper aims to improve the discovery of the actual system process, thereby improving the information system life cycle.

Many process models and also multiple discovery algorithms have been proposed over the years. Where traditional process discovery assumes a single case identifier, object-centric process discovery allows for multiple interacting objects in a process [8]. Thus making it possible to mine system behavior instead of isolated processes as conceptualized in [9]. It has been shown that object-centric approaches can improve the quality of the discovered models because they do not suffer from problems like divergence, convergence, and deficiency [19]. Thus, the research interest recently shifted towards object-centric process analysis, but most models and algorithms do not yet support object-centric processes.

In traditional process mining, there are model notations and miners, for example, for directly follows graphs, Petri nets [15], process trees [12], and causal nets [20]. Each model has unique representational biases, making it suited for specific tasks and less suited for others. Some of them have been generalized to the object-centric setting, like directly follows graphs [14], Petri nets [21], process trees [22], and DCR graphs [6], which orchestrate the lifecycle of object types and support instance spawning and one-to-many relations between objects. There exist also new discoverable object-centric exclusive process models like TOTeM models [13]. Other models like PHILharmonicFlows [11], the MERODE approach [16], or object-centric Petri nets with identifiers [10] also

try to offer different representational biases, but can not be discovered from event data. Current discoverable object-centric process models still have certain representational biases, so we propose object-centric causal nets to overcome them. More concretely, we want a model that does not use artificial constructs for skipping behavior, can clearly model optional object type involvement for activities, and explicitly models object distributions over following activities. The proposed object-centric causal net generalizes the general model idea of traditional causal nets [20] to object-centric processes and builds upon causal discovery algorithms like the flexible heuristic miner [26].

3 Preliminaries

Object-centric process mining deals with events that operate on a variety of objects of different types. Events are activities that happen at a timestamp for a number of objects of different types. $\mathbb{U}_{event}$ is the universe of event identifiers. The universe $\mathbb{U}_{act}$ contains all visible activities. $\mathbb{U}_{typ}$ is the universe of all object types. The universe of objects is $\mathbb{U}_{obj}$. Each object has exactly one type associated with it $\pi_{type} : \mathbb{U}_{obj} \to \mathbb{U}_{typ}$. $\mathbb{U}_{time}$ is the universe of all timestamps. This is the most used subset of the object-centric event log format OCEL 2.0 [2].

Definition 1 (Event Log). *Event log $L = (E, O, OT, \pi_{act}, \pi_{obj}, \pi_{time})$ with:*

- *$E \subseteq \mathbb{U}_{event}$ is a set of events, $O \subseteq \mathbb{U}_{obj}$ is a set of objects,*
- *$OT = \{\pi_{type}(o) | o \in O\}$ is a set of object types,*
- *$\pi_{act} : E \to \mathbb{U}_{act}$ maps each event to an activity,*
- *$\pi_{obj} : E \to \mathcal{P}(\mathbb{U}_{obj}) \setminus \{\emptyset\}$ maps each event to at least one object,*
- *$\pi_{time} : E \to \mathbb{U}_{time}$ maps each event to a timestamp*

4 Object-Centric Causal Nets

Object-centric causal nets describe the causal relations between the activities of a process and enriches this information with split and join constructs both for the workflow and for the object distribution perspective. The basis of an object-centric causal net is an object-centric dependency graph which captures the dependency relations between activities for multiple object types. The object-centric dependency graph is a directed graph with colored edges. The nodes represent activities, and the edges represent causal dependencies for the object type represented by the color.

Definition 2 (Object-Centric Dependency Graph). *An object-centric dependency graph is a tuple $OCDG = (A_A, A_{\triangleright}, A_{\square}, OT, D)$ where:*

- *$A = A_A \cup A_{\triangleright} \cup A_{\square}$ is the set of activities with visible activities $A_A \subseteq \mathbb{U}_{act}$, type-specific start activities $A_{\triangleright} = \{\triangleright_{ot} \mid ot \in OT\}$, and type-specific end activities $A_{\square} = \{\square_{ot} \mid ot \in OT\}$, such that $A_{\triangleright} \cap \mathbb{U}_{act} = \emptyset$ and $A_{\square} \cap \mathbb{U}_{act} = \emptyset$,*
- *$OT \subseteq \mathbb{U}_{ot}$ is a finite set of object types;*

– $D \subseteq (A_A \cup A_{\triangleright}) \times OT \times (A_A \cup A_{\square})$ *is the type-specific dependency relation, depicting the causal dependencies between two activities;*

such that all activities in the type-specific sub-graph $(A_{ot}, \{ot\}, D_{ot})$ *with edges* $D_{ot} = \{(a, ot', a') \in D \mid ot' = ot\} \subseteq D$ *and nodes* $A_{ot} = \{a \mid (a, ot, a') \in D_{ot} \vee (a', ot, a) \in D_{ot}\} \subseteq A$ *are on a path from one start activity* $\triangleright_{ot} \in A_{\triangleright}$ *to the corresponding end activity* $\square_{ot} \in A_{\square}$.

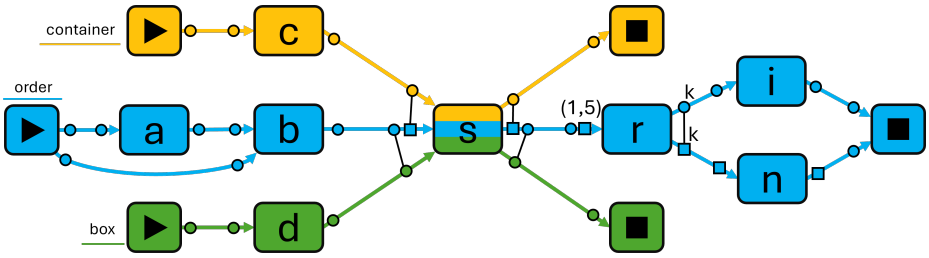


Fig. 2: Object-centric causal net for the running example that describes the example process correctly. Note that all markers are assumed to have unique keys except marked otherwise (here two markers have the key k).

Figure 2 shows the object-centric causal net for the running example, which has an object-centric dependency graph as a basis. There, for example, one can see that for the *order* activity r ($receive feedback$) is causally dependent from activity s ($send$). The object-centric dependency graph describes which activities are causally related to each other, but it does not describe three essential types of workflow information. (1) The workflow splits and joins: The logical connections between possible preceding or succeeding activities. For example, whether all causal successors must always follow or only one of them can follow. (2) The cardinality of involved objects: So whether single or multiple objects participate in the activity, and if multiple participate, how many exactly. (3) The distribution of objects of the same type over multiple succeeding activities: So whether objects of the same type that participated together in an event share the same workflow afterward or are split up. To model these workflow perspectives, we enhance the object-centric dependency graph with groups of markers.

An individual marker is connected to an activity and an object type and has a minimal and maximal cardinality and a key.

Definition 3 (Marker). *Given an object-centric dependency graph* $OCDG = (A_A, A_{\triangleright}, A_{\square}, OT, D)$*, a marker is a tuple* $m = (a, ot, c, k) = (a, ot, (c_{min}, c_{max}), k)$ *where:*

– $a \in A \cup A_{\triangleright} \cup A_{\square}$ *is an activity;*
– $ot \in OT$ *is an object type;*

- $c_{min}, c_{max} \in \mathbb{N}_{>0}$ *are cardinality constrains with* $c_{min} \leq c_{max}$*;*
- $k \in \mathbb{U}_{key}$ *is a key from the universe of keys* $\mathbb{U}_{key}$*.*

The set of potential markers for OCDG is given by $M_{OCDG} = \{(a, ot, c, k) \in A \times OT \times (\mathbb{N}_{>0} \times \mathbb{N}_{>0}) \times \mathbb{U}_{keys}\}$*.*

Markers are represented as dots (capacity of 1) and squares (multiple objects) and can be annotated with a specific capacity range like the marker in front of r (*receive*) in Figure 2. It models that at most five orders are received together. Only non unique keys are shown, like the key k for the two markers after r.

An object-centric causal net consists of an object-centric dependency graph annotated with input and output groups of markers for each activity.

Definition 4 (Object-Centric Causal Net). *An object-centric causal net is a tuple* $OCCN = (OCDG, I, O) = ((A_A, A_{\triangleright}, A_{\square}, OT, D), I, O)$ *where:*

- $OCDG = (A_A, A_{\triangleright}, A_{\square}, OT, D)$ *is an object-centric dependency graph;*
- $I \in A \rightarrow \mathcal{P}(\mathcal{P}(M_{OCDG}))$ *defines the set of possible groups of input markers per activity; and*
- $O \in A \rightarrow \mathcal{P}(\mathcal{P}(M_{OCDG}))$ *defines the set of possible groups of output markers per activity; such that*
- $\forall a \in A : \forall ms \in I(a) : \forall (a', ot, c, k) \in ms : (a', ot, a) \in D$ *(an input marker of an activity must stem from causally preceding activities);*
- $\forall a \in A : \forall ms \in O(a) : \forall (a', ot, c, k) \in ms : (a, ot, a') \in D$ *(an output marker of an activity must only connect to causally succeeding activities);*
- $\forall a \in A_{\triangleright} : I(a) = \{\emptyset\}$ *(no input marker groups for start activities);*
- $\forall a \in A_{\square} : O(a) = \{\emptyset\}$ *(no output marker groups for end activities); and*
- $\forall a, a' \in A, ot \in OT : |\{(a', ot, c, k) \in I(a)\}| \leq 1 \wedge |\{(a', ot, c, k) \in O(a)\}| \leq 1$ *(at most one marker per arc in input and output marker groups);*

Figure 2 shows the object-centric causal net for the running example. Activity s has two input marker groups. One models that one container can be sent with multiple orders: $\{(c, container, (1,1), k_i), (b, order, (1,*), k_j)\} \in I(s)$. It is visualized as the connected yellow dot and blue square in front of s. The other input marker group of s models that a box is sent with one order: $\{(d, box, (1,1), k_l), (b, order, (1,1), k_m)\} \in I(s)$.

The semantics of an object-centric causal net are defined by a state notion, and bindings (representing events) that can change the state. The state is defined as a multiset of obligations. An obligation like $(a, o1, b)$ describes that order $o1$ had performed activity a and thus has to perform activity b in the future.

Definition 5 (State of an OCCN). *Given an object-centric causal net* $OCCN = ((A_A, A_{\triangleright}, A_{\square}, OT, D), I, O)$*, the state space of* $OCCN$ *is* $S_{OCCN} = \mathbb{B}(A \times \mathbb{U}_{obj} \times A)$ *such that* $s \in S_{OCCN}$ *is a state, i.e., a multiset of pending obligations between causally related activities:* $\forall (a, ot, a') \in S_{OCCN} : (a, ot, a') \in D$*.*

For example, the state $s1 = [(a, o1, b), (d, b1, s)]$ describes that b must happen for order $o1$ (because a happened before) and s must happen for box

$b1$ (because d happened before). Bindings can change states. A binding consists of an activity a, the consumed obligations C and the produced obligations P. We use the shorthand notation $a \frac{C}{P}$ for bindings. For example, $s1 = [(b, o1, s), (d, b1, s)]$ $s \xrightarrow[(o1,r),(b1,\square_{box})]{(b,o1),(d,b1)} [(s, o1, r), (s, b1, \square_{box})] = s2$ notes that a binding for activity s transformed the state $s1$ by consuming one obligation for order $o1$ (from b) and one for box $b1$ (from d) and creating new obligations for both objects, resulting in state $s2$. This describes the state change by an event with activity s and objects $o1$ and $b1$. Bindings must match the object-centric causal net. All consumed (and produced) obligations must be mapped to one input (output) marker group. The object assigned to a marker must match its type, and the number of objects assigned per marker must be within its cardinality constraints. Markers with the same key can never share assigned objects. This models an object workflow split. For non-starting and non-ending activities, the objects used in the input and output binding must be the same.

Definition 6 (Binding). *Given an object-centric causal net* $OCCN = ((A_A, A_\triangleright, A_\square, T, D), I, O)$*, the tuple* $b = (a, C, P) \in B = A \times \mathcal{P}(A \times \mathbb{U}_{obj}) \times \mathcal{P}(\mathbb{U}_{obj})$ *is a binding iff there exist marker mappings* $\phi_C \in A \times \mathbb{U}_{obj} \to M_{OCDG}$*,* $\phi_P \in A \times \mathbb{U}_{obj} \to M_{OCDG}$ *such that:*

- $\exists mg \in I(a) : range(\phi_C) = mg$ *(one input marker group used)*
- $\forall (a', o) \in C : \exists c', c'' \in \mathbb{N}_{>0}, k \in \mathbb{U}_{keys} : \phi_C(a', o) = (a', \pi_{type}(o), (c', c''), k)$ *(mapping matches in activity and object type)*
- $\forall (a', ot, c, k) \in M_{OCDG} : |\{o \in \mathbb{U}_{obj} | \phi_C(a', o) = (a', to, c, k)\}| \in c$ *(cardinality within contains)*
- $\forall (a', ot', c', k), (a'', ot'', c'', k) \in M_{OCDG} : \{o \in \mathbb{U}_{obj} | \phi_C(a', o) = (a', ot', c', k) \wedge \phi_C(a'', o) = (a'', ot'', c'', k)\} = \emptyset$ *(exclusive choice of objects for same keys)*

The conditions above must also hold for ϕ_P *in an analog way.*

- $a \in A_A \Rightarrow \{o | \exists a' \in A : (a', o) \in C\} = \{o | \exists a' \in A : (o, a') \in P\}$ *(for non starting or ending activities the objects in* C *and* P *must be the same)*

A binding sequence $\sigma \in B^*$ *is a sequence of activity bindings.* $\langle\rangle$ *denotes the empty binding sequence. Function* $\psi(\sigma)$ *with* $\psi \in B^* \to S_{OCCN}$ *defines the state of an* $OCCN$ *after executing a binding sequence* σ*.* ψ *is defined inductively:* $\psi(\langle\rangle) = []$ *and*

$$\psi(\sigma \oplus (a, C, P)) = \left(\psi(\sigma) \setminus \biguplus_{(a',o) \in C} (a', o, a)\right) \uplus \left(\biguplus_{(o,a') \in P} (a, o, a')\right)$$

The language of an object-centric causal net is given by its valid binding sequences. A binding sequence is valid if, when applied to the empty state, it results in the empty state and only consumes existing obligations. Since each binding relates to an event with a certain activity and related objects, each valid bind sequence relates to a valid execution of the process according to the model.

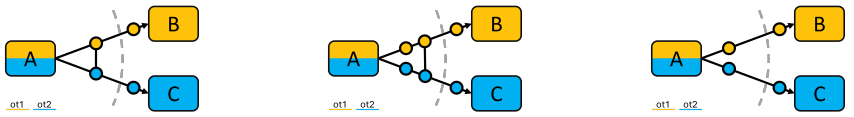
Definition 7 (Valid Binding Sequence). *Given an object-centric causal net* $OCCN = ((A_A, A_\triangleright, A_\square, T, D), I, O)$*, the binding sequence* $\sigma = \langle (a_1, C_1, P_1), (a_2, C_2, P_2), \ldots, (a_n, C_n, P_n) \rangle \in B^*$ *is valid iff*

– $\psi(\sigma) = []$; *and*
– *for any non-empty prefix* $\sigma' = \langle (a_1, C_1, P_1), \ldots, (a_l, C_l, P_l) \rangle$ $(1 \leq l \leq n)$*:* $\biguplus_{(a',o) \in C_l} (a', o, a) \leq \psi(\sigma'')$ *with* $\sigma'' = \langle (a_1, C_1, P_1), \ldots, (a_{l-1}, C_{l-1}, P_{l-1}) \rangle$.

$V(OCCN)$ *is the set of all valid sequences of* $OCCN$.

We semantically restrict the process space of $OCCN$ to $V(OCCN)$. Any invalid sequence is therefore not part of the process described by the model.

The syntax and semantics of object-centric causal nets allow precise modeling of the workflow, the object cardinality, and the object distribution. The workflow modeling is based on the same *AND, OR, and XOR-splits* that traditional causal nets use to describe processes. However, they can involve multiple objects and object types. Figure 3 shows the workflow splits for two object types.



(a) activity-**AND**-split (b) activity-**OR**-split (c) activity-**XOR**-split

Fig. 3: The workflow pattern splits object-centric causal nets can model. The dashed line only serves to separate the input- and output markers visually. More details on workflow splits and joins are described for traditional causal nets [20].

The concrete object cardinality for the activities is modeled by the (c_{min}, c_{max}) values of the markers. Figure 4 shows the cardinality relation between activities that can be modeled. For simplicity, we often visualize only whether the capacity is exactly one (with a circle) or multiple (0 or more).

(a) one-to-one (b) one-to-many (c) many-to-one (d) many-to-many

Fig. 4: Relation between the occurrences of A to the occurrences of B depending on whether the markers between A and B have a capacity of 1 object (represented as a circle) or multiple objects (represented as a square).

Object-centric causal net can precisely describe the object distribution to succeeding activities (see the splits in Figure 5) and, similarly, the merging of objects from preceding activities. The keys of markers describe this. If makers share the same key, any valid binding is required to distribute the objects involved in the event over the markers with the same key without duplicates. This models that objects of this type have to make an exclusive choice if the makers

have the same keys. For markers with unique keys, the objects are free to be assigned to one or multiple markers. Lets assume for example an event $A_{(o1,o2)}$ with objects $o1$ and $o2$ happened. The model in Figure 5b would require each object $o1$ and $o2$ to do either B or C because the markers have the same key k. The model in Figure 5a also allows any of the objects to do both B and C.

(a) free object distribution (b) object choice

Fig. 5: If markers share keys, objects need to exclusively choose one of the markers. Markers without indicated keys are assumed to have unique keys. Objects do not need to choose between them but can be assigned to multiple markers.

Combining all these modeling capabilities, one can correctly model the desired behavior of the running example process and overcome the representational biases of object-centric Petri nets pointed out in section 1. The object-centric causal net in Figure 2 models the running example. First, the skipping behavior for activity a ($checkorder$) is represented without the need for silent transitions in the object-centric causal net. Second, the input and output marker groups for s ($send$) clearly show that *containers* and *boxes* are never used together and that whenever *boxes* are involved, only one *order* is sent, whereas multiple *orders* can be sent with *containers*. Third, the markers with the same key in the output marker group for r ($receivefeedback$) ensure that only one order is investigated and the rest are not investigated. This shows that object-centric causal nets can overcome the mentioned representational biases of object-centric Petri nets.

5 Mining Object-Centric Causal Nets

In this section, we describe the object-centric causal net miner. The algorithm's structure is based on the frequent heuristic miner [26] used to discover traditional causal nets [20]. As defined in Definition 4, an object-centric causal net consists of an object-centric dependency graph enriched with groups of markers. Therefore, the discovery algorithm for object-centric causal nets first discovers an object-centric dependency graph and then discovers marker groups for that graph. Figure 6 shows an overview of the algorithm. The described algorithm is implemented in Python and publicly accessible on GitHub[1].

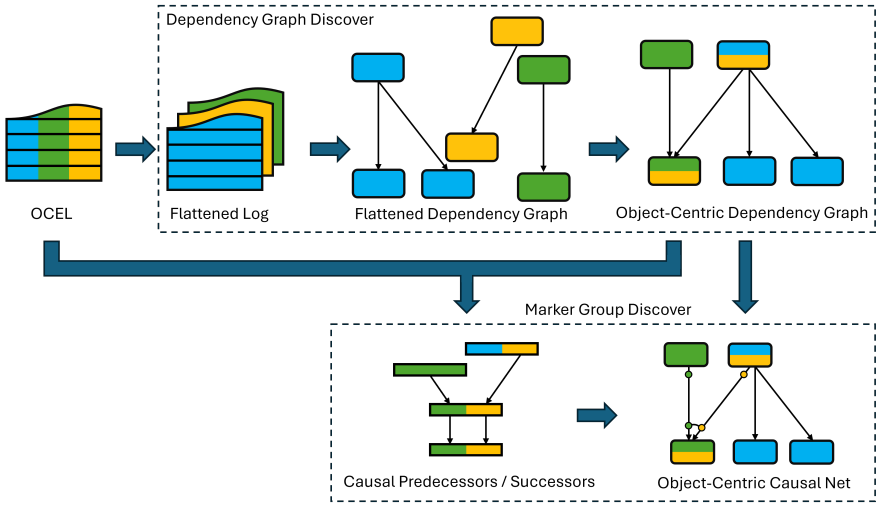


Fig. 6: Overview of the algorithm to mine object-centric causal nets.

5.1 Object-Centric Dependency Graph Discovery

The first step is discovering the object-centric dependency graph. As an input, this step uses an object-centric event log. Since the dependencies are object type-specific, they can be discovered independently for each object type. So, the object-centric event log is then flattened, as described by van der Aalst and Berti [21]. This step creates one traditional event log per object type containing the traces of the objects of that type. For the example log in Table 1, the flattened container log consists of one trace for container $c1$: $\langle c, s \rangle$.

Given the traditional event log, the traditional dependency graph discovery algorithm [26] is used to discover the dependency graphs for each object type. This comes with three parameters DT, L_1T, and L_2T that are described in [26]. Additionally, each object type-specific dependency graph is extended with a start (and end) node connected to all start (and end) activities. The object-centric graph is constructed by merging all nodes with the same activity label. This creates the object-centric dependency graph which is the foundation of Figure 2.

5.2 Marker Group Discovery

The second part of the object-centric causal net discovery is the discovery of the input and output marker groups. The input is the object-centric event log and the discovered object-centric dependency graph. This part consists of two steps. First, each event's causal predecessors and successor from the object-centric event log are determined. Second, input and output marker groups are discovered by detecting frequent patterns in the set of causal predecessors and

[1] `https://github.com/LukasLiss/OCCN-Miner`

successors of events of a certain activity. The marker groups are added to the graph. In the following, we will look at the *s (send)* events from the example log in Table 1 and how input marker groups are created for them. In the tool, users can filter for frequent marker groups to focus on the most dominant behavior.

Causal Predecessor and Successors Heuristics The first step uses the object-centric event log and the object-centric dependency graph to mine for each event in the log, its causal predecessors, and causal successors. The causal predecessors are those events that causally resulted in the execution of the given event. The causal successors are those events that causally result from the execution of the given event. Since the event log does not clearly state which events causally infer each other, a heuristic must be used to select suitable candidates. The heuristic uses both the dependency relations from the dependency graph and the temporal ordering present in the event log. This problem also exists for traditional causal net mining, and two approaches, which can be generalized to the object-centric setting, have been proposed to solve the issue.

One heuristic is window-based: Based on a user-defined window size, one limits the search for causal predecessors and successors to events within the window size of the given event. All dependent events within the window that share objects are considered to be causal predecessors or successors. For the *s (send)* event *e5* in Table 1, all *c*, *b*, and *d* events with a shared object within the window are predecessors because *s* causally depends on them based on the dependency graph. So with a large window size, the events *e2*, *e3*, and *e4* are causal predecessors, but with a window size of 1, only the closest event *e4* remains.

Another heuristic is based on event ordering: This heuristic assumes that a given event is a causal result of those events before the given event that have a dependency to the given event, and there is no other event between them that they also have a dependency to. So, a given event has all those events as its causal predecessors that have the given event as the closest following dependent event. For the causal successors this means that a given event has all those events as its causal successors that have the given event as the closest event before them that has a dependency relation to them. This does not require user input and allows finding predecessors and successors outside of the window size. For example, for the *send* event *e5*, the events *e2*, *e3*, and *e4* are causal predecessors because they share objects, and no events that also share the same object are in between them. So *e1* is not because *e2* happens in between and also contains object *o1*.

The published implementation uses the second heuristic of the ordering of events. But regardless of the used heuristic, the result of this step is a mapping of events $e \in \mathbb{U}_{events}$ to their causal successors $CS(e) \subset \mathbb{U}_{event}$ and predecessors $CP(e) \subset \mathbb{U}_{event}$. For the following, we assume $CS(e)$ and $CP(e)$ given.

Marker Group Algorithm The second step creates input and output marker groups for each event and its causal predecessors and successors. Listing 1.1 shows the algorithm to create an input marker group for a given event and its predecessors. It creates a marker for each activity in the causal predecessors

and selects the cardinality based on the number of objects of a certain type involved in the causal predecessor and the given event. Finally, markers that should have similar keys are identified by checking whether two markers from different activities but with the same object type distribute objects among each other. The objects are distributed between multiple markers of the same type if they do not share any object, then they receive the same key to show that the objects are distributed here.

For example, the *s (send)* event *e5* has predecessors with the following activities: $[b, b, c]$. This creates a marker group that connects b and c. The marker for s has a maximum capacity of 2. If more *s (send)* events have predecessors that match this marker group, no new groups are added. But for the *s (send)* event *e11* with successors with activities $[b, d]$, a new marker group is needed because d is not part of the existing one. So a new marker group that connects b and d is added. Both markers have a capacity of precisely one. These two marker groups are the input marker groups for the *send* activity in the example in Figure 2.

Algorithm 1.1: Marker computation for a given event.

```
1  input: Event e, Causal Predecessors CP(e)
2  output: Input marker group mg (contributed by event e)
3  begin
4    mg ← []
5    foreach a ∈ {π_act(e')|e' ∈ CP(e)}
6      foreach t ∈ {π_type(e')|e' ∈ CP(e) ∧ π_act(e') = a}
7        c_min ← min{|π_obj(e')_↓t ∩ π_obj(e)_↓t| |e' ∈ CP(e) ∧ π_act(e) = a}
8        c_max ← max{|π_obj(e')_↓t ∩ π_obj(e)_↓t| |e' ∈ CP(e) ∧ π_act(e) = a}
9        mg ← mg ∪ (a, t, (c_min, c_max), k) with unused k ∈ 𝕌_keys
10       end
11   end
12
13   # find partition of keys for object distribution
14   while ∃(a', t, (c'_min, c'_max), k') ∈ mg ∧ (a'', t, (c''_min, c''_max), k'') ∈ mg
15    such that a' ≠ a'' ∧ k' ≠ k'' ∧ π_obj(e')_↓t ∩ π_obj(e'')_↓t ≠ ∅
16    for e' ∈ CP(e) ∧ π_act(e') ∧ π_act(e') = a'
17    and e'' ∈ CP(e) ∧ π_act(e') ∧ π_act(e'') = a''
18       # unify keys
19       (a', t, (c'_min, c'_max), k') ← (a', t, (c'_min, c'_max), k'')
20   end
21   return mg
22 end
```

One computes all the input marker groups using the algorithm in Listing 1.1. Similarly, one computes the outgoing marker groups using causal successors. Multiple input marker groups can then be merged when they have the same structure (including the same distribution of keys but abstracting from concrete keys since they are unique). The capacity minimum and maximum are also adjusted based on which marker groups are merged. But markers with a capacity of 1 are not merged with those with a higher capacity to distinguish whether multiple or single objects can be used. The final object-centric causal net, is created by adding all marker groups to the object-centric dependency graph.

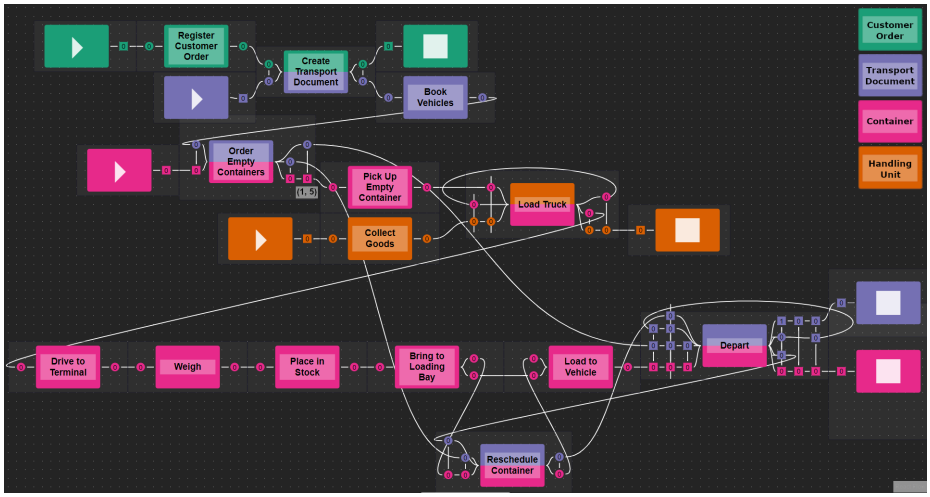


Fig. 7: The object-centric causal net mined, filtered, and visualized with our implementation on the *Logistics* log. The legend shows the object types.

6 Evaluation

In this section, we present the qualitative and quantitative evaluation results. We used 10 publicly available OCEL 2.0 [2] object-centric event logs.

Qualitative Evaluation To evaluate object-centric causal nets and their discovery algorithm qualitatively, we mined object-centric causal nets for the 10 logs and compared them with the discovered object-centric Petri net discovered from the same log. In the following, we focus on the filtered *Logistics* log results, but the other results are publicly accessible in the GitHub repository. We filtered the *Logistics* log for four object types *Customer Orders*, *Transport Document*, *Containers*, and *Handling Unit*. The resulting object-centric causal net can be seen in Figure 7 and the object-centric Petri net in Figure 8.

The overall workflow behaviour is very similar between the causal net and the Petri net. Both show 14 activities. The Petri net contains additionally 25 places and 26 silent transitions. The causal net contains 8 additional start and

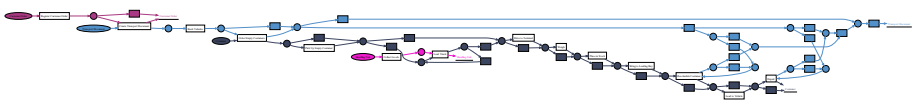


Fig. 8: Object-Centric Petri net [21] for the *Logistics* log mined with pm4py[3]. For better readability see original at `https://www.ocel-standard.org/event-logs/simulations/logistics/`.

end activites. Both describe similar workflow patterns like the sequence of *Create Transport Document* and *Book Vehicle*, or the loop of the *Load Truck* event for the *Container object type*. This shows that the discovered behavior is consistent with already existing approaches but has a different representational bias. For example, after *Order Empty Containers*, the *Transport Document* can either go to *Depart* or directly or to *Reschedule Container* first. For this optional skipping, one needs an artificial silent transition in a Petri net.

The object-centric causal net describes behavior aspects not modeled in the Petri net. For example, it shows concrete cardinalities for the output marker of *Order Empty Containers* which models that at maximum 5 containers are ordered together matching the log. This information can not be represented in an object-centric Petri net. The object-centric causal net in Figure 7 also shows an object XOR split, which would not be explicitly representable in an object-centric Petri net. There can be multiple *Transport Documents* per *Depart* activity. Each can either end its life cycle or be part of another *Depart* event. The qualitative evaluation shows that the behavior discovered using the presented algorithm is consistent with existing approaches in the workflow dimension and can discover additional aspects due to different representational biases.

Quantitative Evaluation This quantitative evaluation investigates the runtime of our implementation by using 10 publicly accessible event logs. The experiments were performed on a 12th Gen Intel i7-12650H processor with 2.3GHz and 16 GB of RAM. For the dependency graph discovery parameters [26] we used the following settings: $DT = 0.5$, $L_1T = 0.5$, and $L_2T = 0.5$. The computation times were measured three times and averaged to reduce the impact of noise. The evaluation shows that the algorithm, with an average overall run time of 125 seconds on the used event logs, is usable on consumer hardware. Figure 9 shows the relation between the overall runtime and the number of events and the number of objects. As the correlation coefficient is 0.92 for the number of events and 0.34 for the number of objects, the runtime is mainly influenced by the number of events in the event log. This is reasonable since the algorithm must discover the potential marker groups for each event in the event log. The linear regression line indicates a linear increase in the runtime with the size of the object-centric event log. Since, there are no algorithms to discover object-centric causal nets yet, we can not compare the performance with other implementations. Instead, we publish the given implementation as a baseline for future improvements.

7 Conclusion

In this paper, we introduce object-centric causal nets and make four contributions to the field of object-centric process discovery. First, we defined the syntax and semantics of object-centric causal nets, a model that can overcome some of the representational biases of current models like object-centric Petri nets. Second, we presented an algorithm to discover them from object-centric event data. This two-phase approach creates an object-centric dependency graph first

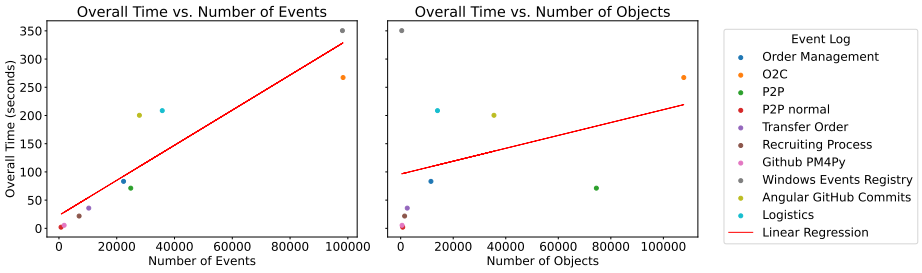


Fig. 9: The overall runtime of the object-centric causal net discovery using our implementation for variate of object-centric event logs.

by merging traditional dependency graphs. In the second phase, this graph is populated with input and output marker groups, resulting in an object-centric causal net. This phase uses a heuristic to identify which events with dependent activities causally proceed and succeed an event. Then the causal predecessors (successors) result in input (output) marker groups. Third, we published our algorithm implementation together with a visualization tool for object-centric causal nets that allows the user to filter for the most frequent input and output marker groups. Finally, we performed a qualitative and quantitative evaluation, showing that object-centric causal nets can represent behavior that can not be discovered and modeled using object-centric Petri nets.

This work motivates future research on object-centric causal nets for conformance checking [5], enhancement [7] and prediction [17].

Acknowledgments. The authors gratefully acknowledge the financial support by the Federal Ministry of Education and Research (BMBF) for the joint project Bridging AI (grant no. 16DHBKI023).

SPONSORED BY THE
Federal Ministry of Education and Research

References

1. Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, Andrea Marrella, Massimo Mecella, and Allar Soo. Automated discovery of process models from event logs: Review and benchmark. *IEEE Trans. Knowl. Data Eng.*, 31(4):686–705, 2019.
2. Alessandro Berti, István Koren, Jan Niklas Adams, Gyunam Park, Benedikt Knopp, Nina Graves, Majid Rafiei, Lukas Liß, Leah Tacke genannt Unterberg, Yisong Zhang, Christopher T. Schwanen, Marco Pegoraro, and Wil M. P. van der Aalst. OCEL (object-centric event log) 2.0 specification. *CoRR*, abs/2403.01975, 2024.

3. Alessandro Berti, Sebastiaan J. van Zelst, and Daniel Schuster. Pm4py: A process mining library for python. *Softw. Impacts*, 17:100556, 2023.
4. Tilo Böhmann, Jan Marco Leimeister, and Kathrin M. Möslein. Service systems engineering - A field for future information systems research. *Bus. Inf. Syst. Eng.*, 6(2):73–79, 2014.
5. Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich. *Conformance Checking - Relating Processes and Models*. Springer, 2018.
6. Axel KF Christfort, Andrey Rivkin, Dirk Fahland, Thomas T Hildebrandt, and Tijs Slaats. Discovery of object-centric declarative models. In *2024 6th International Conference on Process Mining (ICPM)*, pages 121–128. IEEE, 2024.
7. Massimiliano de Leoni. Foundations of process enhancement. In Wil M. P. van der Aalst and Josep Carmona, editors, *Process Mining Handbook*, volume 448 of *Lecture Notes in Business Information Processing*, pages 243–273. Springer, 2022.
8. Dirk Fahland. Process mining over multiple behavioral dimensions with event knowledge graphs. In Wil M. P. van der Aalst and Josep Carmona, editors, *Process Mining Handbook*, volume 448 of *Lecture Notes in Business Information Processing*, pages 274–319. Springer, 2022.
9. Peter Fettke and Wolfgang Reisig. Systems mining with heraklit: The next step. In Claudio Di Ciccio, Remco Dijkman, Adela del Río Ortega, and Stefanie Rinderle-Ma, editors, *Business Process Management Forum*, pages 89–104, Cham, 2022. Springer International Publishing.
10. Alessandro Gianola, Marco Montali, and Sarah Winkler. Object-centric conformance alignments with synchronization. In Giancarlo Guizzardi, Flávia Maria Santoro, Haralambos Mouratidis, and Pnina Soffer, editors, *Advanced Information Systems Engineering - 36th International Conference, CAiSE 2024, Limassol, Cyprus, June 3-7, 2024, Proceedings*, volume 14663 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2024.
11. Vera Künzle and Manfred Reichert. Philharmonicflows: towards a framework for object-aware process management. *J. Softw. Maintenance Res. Pract.*, 23(4):205–244, 2011.
12. Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering block-structured process models from event logs - A constructive approach. In José Manuel Colom and Jörg Desel, editors, *Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS 2013, Milan, Italy, June 24-28, 2013. Proceedings*, volume 7927 of *Lecture Notes in Computer Science*, pages 311–329. Springer, 2013.
13. Lukas Liss, Jan Niklas Adams, and Wil M. P. van der Aalst. Totem: Temporal object type model for object-centric process mining. In Andrea Marrella, Manuel Resinas, Mieke Jans, and Michael Rosemann, editors, *Business Process Management Forum - BPM 2024 Forum, Krakow, Poland, September 1-6, 2024, Proceedings*, volume 526 of *Lecture Notes in Business Information Processing*, pages 107–123. Springer, 2024.
14. Gyunam Park, Jan Niklas Adams, and Wil M. P. van der Aalst. Conformance checking and performance analysis using object-centric directly-follows graphs. In Andrea Marrella, Manuel Resinas, Mieke Jans, and Michael Rosemann, editors, *Business Process Management Forum - BPM 2024 Forum, Krakow, Poland, September 1-6, 2024, Proceedings*, volume 526 of *Lecture Notes in Business Information Processing*, pages 179–196. Springer, 2024.
15. Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.

16. Monique Snoeck. *Enterprise Information Systems Engineering - The MERODE Approach.* The Enterprise Engineering Series. Springer, 2014.
17. Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. Predictive business process monitoring with LSTM neural networks. In Eric Dubois and Klaus Pohl, editors, *Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings*, volume 10253 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2017.
18. Wil M. P. van der Aalst. Process-aware information systems: Lessons to be learned from process mining. *Trans. Petri Nets Other Model. Concurr.*, 2:1–26, 2009.
19. Wil M. P. van der Aalst. Object-centric process mining: Dealing with divergence and convergence in event data. In Peter Csaba Ölveczky and Gwen Salaün, editors, *Software Engineering and Formal Methods - 17th International Conference, SEFM 2019, Oslo, Norway, September 18-20, 2019, Proceedings*, volume 11724 of *Lecture Notes in Computer Science*, pages 3–25. Springer, 2019.
20. Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Causal nets: A modeling language tailored towards process discovery. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*, volume 6901 of *Lecture Notes in Computer Science*, pages 28–42. Springer, 2011.
21. Wil M. P. van der Aalst and Alessandro Berti. Discovering object-centric petri nets. *Fundam. Informaticae*, 175(1-4):1–40, 2020.
22. Jan Niklas van Detten, Pol Schumacher, and Sander J. J. Leemans. Discovering compact, live and identifier-sound object-centric process models. In *6th International Conference on Process Mining, ICPM 2024, Kgs. Lyngby, Denmark, October 14-18, 2024*, pages 113–120. IEEE, 2024.
23. Boudewijn F. van Dongen, Ana Karla Alves de Medeiros, and Lijie Wen. Process mining: Overview and outlook of petri net discovery algorithms. *Trans. Petri Nets Other Model. Concurr.*, 2:225–242, 2009.
24. Daniel Veit, Eric K. Clemons, Alexander Benlian, Peter Buxmann, Thomas Hess, Dennis Kundisch, Jan Marco Leimeister, Peter Loos, and Martin Spann. Business models - an information systems research agenda. *Bus. Inf. Syst. Eng.*, 6(1):45–53, 2014.
25. Barbara Weber, Manfred Reichert, Stefanie Rinderle-Ma, and Werner Wild. Providing integrated life cycle support in process-aware information systems. *Int. J. Cooperative Inf. Syst.*, 18(1):115–165, 2009.
26. A. J. M. M. Weijters and Joel Tiago S. Ribeiro. Flexible heuristics miner (FHM). In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France*, pages 310–317. IEEE, 2011.