Discovering the Lively World of Object-Centric Processes with Dynamic Relationships

Alessandro Gianola[1], Zeeshan Hameed[2], Marco Montali[2], Anjo Seidel[3], Mathias Weske[3], and Sarah Winkler[2]

[1] INESC-ID/Instituto Superior Técnico, Universidade de Lisboa, Portugal
alessandro.gianola@tecnico.ulisboa.pt
[2] Free University of Bozen-Bolzano, Italy {hameed,winkler,montali}@inf.unibz.it
[3] Hasso Plattner Institute, University of Potsdam, Germany
{anjo.seidel,mathias.weske}@hpi.de

Abstract. Object-centric process mining examines how processes interact with multiple co-evolving objects. Together with the introduction of the OCEL standard for object-centric event logs, several modeling approaches have emerged, including object-centric Petri nets (OCPNs) and object-centric Petri nets with identifiers (OPIDs). While OCPN discovery has been widely explored, OCPNs leave object relationships underspecified and cannot enforce synchronization between objects. OPIDs address synchronization by capturing object identities and relations, but existing discovery techniques can only handle stable relationships among objects that do not change during process execution. In this paper, we investigate discovery techniques for OPIDs that support fine-grained dynamic relations between objects. Our contributions are fourfold: (1) We identify and formally define for the first time the standard assumptions underlying object-centric process mining, specifically regarding how relationships are represented and manipulated within OCEL. (2) Focusing on dynamically changing one-to-many relationships, we determine the essential net patterns that govern the creation, deletion, and propagation of links according to these relationships, and we introduce a precise way to label activities with the applicable patterns. (3) Building on this labelling, we propose a novel discovery technique for OPIDs, capable of revealing dynamically changing relationships. (4) Combining this approach with the existing one for stable relationships, we provide a full procedure to discover OPIDs from OCELs that supports both stable and dynamically changing relationships, which is also implemented.

1 Introduction

Process mining aims to extract actionable insights from the execution data recorded by information systems [16]. By analyzing event logs, it allows to reconstruct the actual behavior of operational processes and reason about performance, deviations, and conformance. While traditional process mining assumes that events refer to a single case notion, modern information systems store data about multiple interacting process instances that share co-evolving objects [14]. This has motivated the development of object-centric process mining, which

studies processes that simultaneously evolve several interrelated objects of different types. Executions are recorded using the standard OCEL format (object-centric event logs [6]), where each event may reference multiple objects, enabling a richer, more comprehensive, and more realistic view of processes.

A variety of modeling formalisms has been proposed to represent processes exhibiting object-centric behavior. The most lightweight option are object-centric Petri nets (OCPN) [17], which extend classical Petri nets to handle multiple object types and their token flows. OCPNs are relatively simple and easy to visualize, which makes them well suited for exploratory analysis, and their structural conciseness makes them highly amenable to automated discovery. However, they do not support synchronization based object-to-object relationships: they track the types of objects involved in transitions but not their identities, inter-object relationships, or cardinality constraints. As a consequence, OCPNs tend to be underspecified [15]. However, more expressive formalisms capturing various forms of synchronizations exist in the Petri net literature, such as synchronous proclets [5]. Also for operational process mining, more expressive object-centric modeling languages have been introduced to overcome the limitations of OCPNs, including variants of Petri nets with identifiers (PNIDs [7, 18]). A recent unifying proposal are object-centric Petri nets with identifiers (OPIDs) [9, 10], which enhance OCPNs with explicit object identities and synchronization semantics.

The discovery of OCPNs has been addressed in prior work [17], and the developed techniques are now well-established and supported by tools. However, because OCPNs do not enforce identity-aware synchronization, discovered models often permit more behavior than what is reflected or allowed in the event data. As argued in [13], when it comes to relationships between objects, an implicit restriction that is so far pervasive in the object-centric process mining literature is that such relationships are *stable* or *rigid*. This means that when an object of one type is linked to an object of another type, this association remains fixed throughout the execution. Recent work has shown how OPIDs can restrict replays to executions that respect these stable relationships [13], and how one can discover OPIDs where the only intended synchronization semantics is precisely that induced by stable many-to-one relationships: first by invoking OCPN discovery off-the-shelf, and second by lifting the OCPN into a corresponding OPID that manipulates relationships in a stable way, guaranteeing log replayability.

All this literature leaves out the important case where relationships dynamically change over time — such as, for example, a process where a package is moved from a truck to another, or where the destination address is dynamically updated. While this aspect has been addressed in the context of conformance checking [9, 10], it is still largely missing when it comes to discovery.

In this paper, we address the problem of *discovering object-centric processes from event data, considering that relationships may dynamically change over time*. We do so through a fourfold contribution. (1) We identify and formally define for the first time some standard assumptions regarding how to log in an OCEL object-centric event data with dynamically changing relationships, using minimal additional domain knowledge while guaranteeing that the dynamics of such relationships can be unambiguously reconstructed. (2) Focusing on dynam-

ically changing one-to-many relationships, determine the essential net patterns that govern the creation, deletion, and propagation of co-evolving objects in such relationships, and we introduce a precise way to label activities based on the relationship patterns that they implement. We do so using OPIDs as target formalism, due to their natural ability of combining the main modelling constructs for object-centric processes [9]. (3) Building on these patterns, we propose a novel discovery technique capable of discovering dynamically changing relationships in OPIDs. (4) Combining this approach with the one already existing for stable relationships, we provide a full procedure for discovering OPIDs from OCEL that natively support stable and dynamically changing relationships. This procedure reconstructs to the case of dynamic relationships the approach used in [13], that is, we first use off-the-shelf existing OCPN discovery techniques (extensively validated in terms of performance [17]), and we then transform the obtained OCPN into an OPID that suitably manipulates dynamic relationships. This procedure is implemented and the code available.

Dynamic relationships were so far not investigated for OPIDs, though they were partially studied in two, distinct seminal works. In [12], temporal object type models (TOTeM) are mined from OCELs. However, the result is not an object-centric process model, as in our case. Moreover, dynamics are defined considering the temporal existence windows on objects, while relationships are tied to such temporal windows, assuming that they are not updated in between. In [19], *temporal implications between two types* are considered as one form of dynamic relations. The principal aim of this notion is to enforce that for a restricted timespan, one set of objects of some type σ (e.g., a particular employee) is linked to only one set of objects of another type τ (e.g., an order). Such a relationship can also be discovered by our approach. However, there are at least three major differences to our work. First, the authors do not clarify their assumptions on the OCEL semantics, i.e., it is not clear how long a link between two objects is supposed to exist. Second, the temporal implication always compares exact sets of objects in events, so that a growing/shrinking relationship, e.g., an order that is extended/reduced, cannot be modelled. Hence also the roles of the different activities wrt. creation and deletion of relationships cannot be distinguished. Finally, their implementation is restricted to object-centric process trees, and though in theory the approach can be extended to OCPNs, these OCPNs will always be block-structured, a restriction that does not apply here.

2 Background

To define OCELs, we recall notions by van der Aalst and Berti [17].

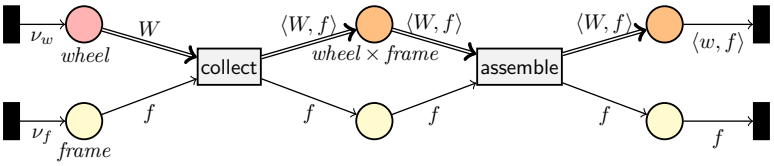
Definition 1. *Given a set of object types Σ, a set of activities $\mathcal{A}$, and a set of timestamps $\mathbb{T}$ under a total order $<$, an* object-centric event log *(OCEL) is a tuple $L = (E, \mathcal{O}, \pi_{act}, \pi_{obj}, \pi_{time})$ where: (i) E is a set of events; (ii) $\mathcal{O}$ is a set of object identifiers, each typed by the function $ot\colon \mathcal{O} \to \Sigma$; (iii) the functions $\pi_{act}\colon E \to \mathcal{A}$, $\pi_{obj}\colon E \to \mathcal{P}(\mathcal{O})$, and $\pi_{time}\colon E \to \mathbb{T}$, mapping each event to an activity, a set of objects, and a timestamp.*

The *object graph* $\mathcal{G}_L$ of an event log L is the undirected graph with node set $\mathcal{O}$, and an edge from o to o' if there is some event $e \in E$ such that $o \in \pi_{obj}(e)$ and $o' \in \pi_{obj}(e)$. Thus, the object graph indicates which objects share events.

Definition 2. *Let X be a connected component in $\mathcal{G}_L$. Then, the* trace graph *induced by X is the directed graph $\langle E_X, D_X \rangle$ where: (i) the set of nodes E_X is the set of all events $e \in E$ that involve objects in X, i.e., such that $X \cap \pi_{obj}(e) \neq \emptyset$, and (ii) the set of edges D_X consists of all $\langle e, e' \rangle$ such that for some $o \in \pi_{obj}(e) \cap \pi_{obj}(e')$, it is $\pi_{trace}(o) = \langle e_1, \ldots, e_n \rangle$ and $e = e_i$, $e' = e_{i+1}$ for some $0 \leq i < n$.*

2.1 Object-centric Petri Nets with Identifiers

Example 1 (Simple OPID for bike manufacturing). The OPID combines the object types *wheel* (red), *frame* (yellow), and relations for both (orange places).



We summarize the definitions from [9]. Every object type $\sigma \in \Sigma$ is assumed to have a domain $dom(\sigma) \subseteq \mathcal{O}$, given by all objects in $\mathcal{O}$ of type σ. List types with a base type in σ are denoted as $[\sigma]$. Each Petri net place has a *color* that is a cartesian product of data types from Σ; precisely, the set of colors $\mathcal{C}$ is the set of all $\sigma_1 \times \cdots \times \sigma_m$ such that $m \geq 1$ and $\sigma_i \in \Sigma$ for all $1 \leq i \leq m$. We fix a set of Σ-typed variables $\mathcal{X} = X \uplus X_{list} \uplus \Upsilon$ as the disjoint union of a set X of "normal" variables that refer to single objects, denoted by lower-case letters like x, with a type $\text{type}(x) \in \Sigma$, a set X_{list} of list variables that refer to a list of objects of the same type, denoted by upper case letters like U, with a type $\text{type}(U) = [\sigma]$ for some $\sigma \in \Sigma$, and a set Υ of variables referring to fresh objects, denoted as ν, with $\text{type}(\nu) \in \Sigma$.

In OPIDs [9], tokens are object tuples associated with a color. To define relationships between objects in consumed and produced tokens when firing a transition, arc *inscriptions* that can be *template* or *simple*, are defined. Template inscriptions contain one variable in X_{list} and capture an arbitrary number of tokens of the same color; they are used to label the correspondent of variable arcs in OCPNs. In Example 1, the inscriptions $\langle W, f \rangle$ and W are template, whereas e.g. f and ν_w are simple.

The color of an inscription $\iota = \langle x_1, \ldots, x_m \rangle$ is given by $\text{color}(\iota) = \langle \sigma_1, \ldots, \sigma_m \rangle$ where $\sigma_i = \text{type}(x_i)$ if $x_i \in X \cup \Upsilon$, and $\sigma_i = \sigma'$ if x_i is a list variable of type $[\sigma']$. We set $\text{vars}(\iota) = \{x_1, \ldots, x_m\}$. The set of all inscriptions is denoted Ω. Based on this, an OPID is formally defined as follows [9].

Definition 3. *An* object-centric Petri net with identifiers *(OPID) is defined as a tuple $N = (\Sigma, P, T, F_{in}, F_{out}, \text{color}, \ell)$, where:*

1. *P and T are finite sets of places and transitions such that $P \cap T = \emptyset$,*

2. $\text{color}\colon P \to \mathcal{C}$ *maps every place to a color,*
3. $\ell\colon T \to \mathcal{A} \cup \{\tau\}$ *is the transition labelling where* τ *marks an invisible activity,*
4. $F_{in}\colon P \times T \to \Omega$ *is a partial function called* input flow*, such that* $\text{color}(F_{in}(p,t)) = \text{color}(p)$ *for every* $(p,t) \in dom(F_{in})$*,*
5. $F_{out}\colon T \times P \to \Omega$ *is a partial function called* output flow*, such that* $\text{color}(F_{out}(t,p)) = \text{color}(p)$ *for every* $(t,p) \in dom(F_{out})$*.*

We set $\text{vars}_{in}(t) = \bigcup_{p \in P} \text{vars}(F_{in}(p,t))$*, and* $\text{vars}_{out}(t) = \bigcup_{p \in P} \text{vars}(F_{out}(t,p))$*, and require that* $\text{vars}_{in}(t) \cap \Upsilon = \emptyset$ *and* $\text{vars}_{out}(t) \subseteq \text{vars}_{in}(t) \cup \Upsilon$*, for all* $t \in T$*.*

If $F_{in}(p,t)$ is defined, it is called a *variable* flow if $F_{in}(p,t)$ is a template inscription, and *non-variable* flow otherwise; and similar for output flows. Variable flows play the role of variable arcs in OCPNs [17]; they can carry multiple tokens at once. The common notations $\bullet t = \{p \mid (p,t) \in dom(F_{in})\}$ and $t\bullet = \{p \mid (t,p) \in dom(F_{out})\}$ are used as well.

Semantics. Given the set of objects $\mathcal{O}$, the set of *tokens* $\boldsymbol{\mathcal{O}}$ is the set of object tuples $\boldsymbol{\mathcal{O}} = \{\mathcal{O}^m \mid m \geq 1\}$. The *color* of a token $\omega \in \boldsymbol{\mathcal{O}}$ of the form $\omega = \langle o_1, \ldots, o_m \rangle$ is denoted $\text{color}(\omega) = \langle \text{type}(o_1), \ldots, \text{type}(o_m) \rangle$. A *marking* of an OPID $N = \langle \Sigma, P, T, F_{in}, F_{out}, \text{color}, \ell \rangle$ is a function $M\colon P \to 2^{\boldsymbol{\mathcal{O}}}$, such that for all $p \in P$ and $\langle o_1, \ldots o_m \rangle \in M(p)$, it holds that $\text{color}(\langle o_1, \ldots o_m \rangle) = \text{color}(p)$. Let $\mathit{Lists}(\mathcal{O})$ denote the set of objects lists of the form $[o_1, \ldots, o_k]$ with $o_1, \ldots, o_k \in \mathcal{O}$ such that all o_i have the same type; the type of such a list is then $[\text{type}(o_1)]$. Next, *bindings* are defined to fix which objects are involved in a transition firing.

Definition 4. *A* binding *for a transition* t *and a marking* M *is a type-preserving function* $\beta\colon \text{vars}_{in}(t) \cup \text{vars}_{out}(t) \to \mathcal{O} \cup \mathit{Lists}(\mathcal{O})$*. To ensure the freshness of created values,* β *is required to be injective on* $\Upsilon \cap \text{vars}_{out}(t)$*, and* $\beta(\nu)$ *must not occur in* M *for all* $\nu \in \Upsilon \cap \text{vars}_{out}(t)$*.*

We denote by $\mathit{codom}(\beta) \subseteq \mathcal{O}$ the set of all objects occurring in tuples returned by β. Bindings are extended to inscriptions, denoted $\boldsymbol{\beta}$, to fix which tokens participate in a transition firing. For an inscription $\iota = \langle x_1, \ldots, x_m \rangle$, let $o_i = \beta(x_i)$ for all $1 \leq i \leq m$. Then $\boldsymbol{\beta}(\iota)$ is the set of object tuples defined as follows: if ι is a simple inscription, then $\boldsymbol{\beta}(\iota) = \{\langle o_1, \ldots, o_m \rangle\}$. Otherwise, there must be one x_i, $1 \leq i \leq n$, such that $x_i \in X_{list}$, and consequently o_i must be a list, say $o_i = [u_1, \ldots, u_k]$ for some $u_1, \ldots, u_k$. Then $\boldsymbol{\beta}(\iota) = \{\langle o_1, \ldots, o_{i-1}, u_1, o_{i+1}, \ldots, o_m \rangle, \ldots, \langle o_1, \ldots, o_{i-1}, u_k, o_{i+1}, \ldots, o_m \rangle\}$. $\mathcal{B}$ denotes the set of all bindings. Finally, a transition with a binding β is enabled if all object tuples pointed by $\boldsymbol{\beta}$ occur in the current marking:

Definition 5. *A transition* $t \in T$ *and a binding* β *for marking* M *are* enabled *in* M *if* $\boldsymbol{\beta}(F_{in}(p,t)) \subseteq M(p)$ *for all* $p \in \bullet t$*.*

Definition 6. *Let transition* t *be enabled in marking* M *with binding* β*. The* firing *of* t *yields the new marking* M' *given by* $M'(p) = M(p) \setminus \boldsymbol{\beta}(F_{in}(p,t))$ *for all* $p \in \bullet t \setminus t\bullet$*, and* $M'(p) = M(p) \cup \boldsymbol{\beta}(F_{out}(p,t))$ *for all* $p \in t\bullet \setminus \bullet t$*, and* $M'(p) = M(p) \setminus \boldsymbol{\beta}(F_{in}(p,t)) \cup \boldsymbol{\beta}(F_{out}(p,t))$ *for all* $p \in \bullet t \cap t\bullet$*.*

We write $M \xrightarrow{t,\beta} M'$ to denote that t is enabled for binding β in M, and its firing yields M'. An *accepting* OPID N is an OPID together with a set of initial

markings M_{init} and a set of final markings M_{final}. A run is accepted if it starts in the initial and ends in the final marking $M_{init} \xrightarrow{*} M_{final}$.

3 OCELs and Dynamic Relationships

Contemporary object-centric event data representations [6] lack an explicitly defined semantics regarding the way events impact the evolution of object-to-object relationships[4] — that is, which are the temporal intervals in which actual links of a given relationship hold. For terminological clarity, we use *relationship* at the type level, and *link* at the instance level. For example, the type-level information that "an order contains items" denotes a relationship, while the fact that "order o_1 contains item i_2" denotes a link. So far, virtually all approaches dealing with OCELs and corresponding mining/analysis support explicit relationships between objects [2], or implicit relationships established when they co-participate to the same event [4]. In either case, such relationships are assumed to be *rigid* (or *stable*), that is, once established, a link never ceases to exist [13]. Even in the most recent advancements dealing with temporal dynamics, such dynamics pertain the intervals of existence of objects, and when two objects are linked, the link is expected to hold in all the time instants where both objects are present [12]. In this paper we call a *dynamic many-to-one* relationship between a one-type σ and a many-type τ a relationship in which at every point in time, every object of type τ is linked to at most one object of type σ. In contrast, in *many-to-many* relationships, an object of type τ may at some point be linked to more than one object of type σ. If not mentioned otherwise, a *many-to-one* relationship will supposed to be dynamic.

The goal of this section is to provide a minimally intrusive mechanism to unambiguously view how an OCEL tracks the evolution of links of relationships. By "minimimally intrusive" we mean that we want to limit, as much as possible, the number of assumptions made, and the additional domain knowledge needed, still ensuring that the evolution of relationships can be univocally reconstructed from the log. This latter condition will be essential for our discovery technique.

First and foremost, there are two possible ways of interpreting events in an OCEL. We discuss them with an example.

Example 2. Consider the situation where an order o_1 is created with items i_1 and i_2, then item i_2 gets removed from the order, and later i_3 is added. Depending on the interpretation of object participation in events, this can be expressed in two different ways.

In what we call the *parameter semantics*, objects denote the actual parameters of the activity/transaction/operation referred by the event. An OCEL could thus look as follows:

[4] From now on, when we generically mention *relationships* we mean object-to-object relationships, making explicit reference when intending event-to-object relationships.

event	activity	order	items
$\#_1$	create order	o_1	i_1, i_2
$\#_2$	reduce order	o_1	i_2
$\#_3$	add to order	o_1	i_3

Note that each operation is associated with the objects to which the operation applies. The current state after each operation, in particular regarding which links are established and which removed, remains however implicit.

On the other hand, one can adopt a so-called *snapshot semantics*, where the objects listed with each event do not represent parameters, but provide instead a snapshot of the new state of affairs (e.g., the new state of an order) *after* the respective operation, thus indicating directly which links exist. This would yield:

event	activity	order	items
$\#_1$	create order	o_1	i_1, i_2
$\#_2$	reduce order	o_1	i_1
$\#_3$	add to order	o_1	i_1, i_3

Note how in event $\#_2$, the associated item is i_1, i.e., the *remaining* item in o_1 – witnessing that the set of active links for the order-item relationship consists only of (o_1, i_1). This is in contrast to the table above, where event $\#_2$ is associated with i_2, the item *to which the removal operation is applied*.

The parameter semantics makes it impossible to reconstruct which links hold and when from an OCEL alone: they need to be accompanied by explicit transactional domain knowledge on the updates induced by actions/operations [3]. At the same time, even though this is not explicitly mentioned, existing approaches typically adopt the snapshot semantics, as they use events and the objects appearing there to derive how objects are related [4] (not which links do not hold anymore). We therefore follow this assumption.

Assumption 1 *Events are recorded under the snapshot semantics, that is, listing which objects get related as an effect of the event.*

However, the next example shows that even under the snapshot semantics, there is inherent ambiguity in the way OCELs capture changing relationships.

Example 3. Consider employees handling orders. We have three events, recorded in the following OCEL:

event	activity	order	employee
$\#_4$	create order	o_1	p_1
$\#_5$	create order	o_2	p_1
$\#_6$	change order manager	o_1	p_2

From the co-participation of orders and employees to the same event, we can only infer that there exists a relationship between some of their instances, but we do not know with which multiplicity constraints. This extra-knowledge is essential. If the relationship is many-to-many, then we may infer from the OCEL that order o_1 is linked to both employees p_1 and p_2 after $\#_6$. If instead the relationship is one-to-many – more specifically, every employee can handle multiple orders, but every order is handled by one and only one employee – then we may instead infer that o_1 was initially linked to p_1, and upon $\#_6$ it changed its link to p_2. This witnesses that the relationship is a *dynamic* one-to-many relationship instead of a rigid many-to-many relationship: over time the same order can be linked to distinct employees but in every instant to only one.

Ex. 3 shows that some minimal domain knowledge is needed regarding the multiplicity of relationships at the type level. In addition, while OPIDs natively

support many-to-many relationships, they make it difficult to properly handle synchronization of related objects, as noted in [1, 5]. As pointed out in [5], synchronization can be handled naturally upon prior reification of every many-to-many relationship into a pair of one-to-many relationships with a newly introduced object type in between. This leads us to the following assumption.

Assumption 2 *We assume that domain knowledge is provided, indicating which object types are connected by a relationship, as well as which type is on the many side and which type is on the one side.*

In a one-to-many relationship, we refer to the "one" side as the parent, and to the "many" side as the child type — hence at every time instant, each parent object is related to zero or more child objects, and every child object is related to zero or one parent object. When referring to a one-to-many relationship from the "many" side, we call it a many-to-one relationship. We do not enforce mandatory participation on either side. In the remainder of this paper, we only refer to one-to-many relationships, but as far as our approach is concerned, one-to-one can be handled in the same way. Ex. 2 and 3 leave another point open regarding the snapshot semantics: how to unambiguously reconstruct which links hold after an event, depending on whether the event primarily operates on the parent or on the child endpoint of a one-to-many relationship. In Ex. 2, events primarily operate on the order type, that is, the parent side of the one-to-many relationship with the item type. Consequently, each event that operates on both the order and the item types needs to list the set of items to which the order is linked upon the occurrence of event. In particular, when o_1 occurs with i_1 in $\#_2$, this implies that the item set i_1, i_2 that occurred with o_1 in $\#_1$ is replaced: link (o_1, i_1) is kept, while the link (o_1, i_2) does not hold anymore.

Also in Ex. 3, events primarily operate on the order type. However, the order type now corresponds to the child side of the many-to-one relationship with the employee type. Differently from the previous case, now every event that operates on both an order and an employee mentions exactly one object of each type, but does not pertain *other* orders that the same employee handles. For example, the fact that employee p_1 handles a second order o_2 according to event $\#_4$ does *not* mean that p_1 does no longer handle order o_1 (as previously indicated by event $\#_1$) i.e., $\#_4$ does not delete the link between p_1 and o_1 established by $\#_1$.

This shows that the snapshot semantics must be further clarified: in the first case, the active links for an order to its items can be reconstructed from the current event alone, while in the second case the active links for an employee to its orders require to also consider previous events. These two ways of operating over links correspond to the classical alternative to maintain links from the many or the one side of the relationship. One may be tempted to impose that only one of the two conventions should be adopted in an OCEL. However, this cannot be done when events operate over three or more object types, as illustrated next.

Example 4. Consider an OCEL obtained by merging those in Ex. 2 and 3:

	activity	order	item	employee
$\#_1$	create order	o_1	i_1, i_2	p_1
$\#_2$	reduce order	o_1	i_1	
$\#_3$	add to order	o_1	i_1, i_3	
$\#_4$	create order	o_2	i_4, i_5	p_1
$\#_5$	wrap item		i_4	
$\#_6$	change order manager	o_1		p_2
$\#_7$	ship order	o_1	i_1, i_3	
$\#_8$	ship order	o_2	i_4, i_5	

Now orders are at once linked to the items they contain, and to the employees that handle them. To properly interpret which active links exist after an event, we need to know what is the primary object type (which we call *reference type*) for the corresponding activity. In the OCEL here, this is *order* for all events except wrap item (which only operates over an item, so must have *item* as reference type). Once this is settled, depending on which other types are involved in the activity, and which multiplicity the corresponding relationships have, we know how to reconstruct the active links. Notice that for events of type create order, we need to consider that the order-to-item relationship is treated from the parent side (thus enumerating the set of items to which the mentioned order is linked), while the order-to-employee one is treated from the child side (thus only indicating what is the current employee to which the mentioned order is linked).

Generalizing the observations in Ex. 4, we get to the following assumptions.

Assumption 3 *Every activity has one object type as a* reference type*, and every event in the log has exactly one object of the reference type of its activity. We call this object the* reference object*.*

Assumption 4 *Let $e = (a, O_e)$ be an event in the log with activity a and object set O_e, and t_{ref} the reference type of a. For every object $o \in O_e$ that has an object type t that is by the domain knowledge in a relationship with t_{ref}, either*

1. *(t_{ref}, t) is a many-to-one relationship, o is the only object in O_e with type t, and o is the only object currently in relationship with the reference object; or*
2. *(t_{ref}, t) is a one-to-many relationship and the subset of O_e with type t are all objects of type t currently in relationship with the reference object.*

These conventions clarify the confusion in Ex. 4: for all activities except wrap item, the reference type is *order*, and indeed each of these events mention only one object of type *order*, namely the one listed first in the table (so Condition 3 is satisfied). Since *order* — *item* is a one-to-many relationship, when the activities with reference type order contain a set of items, these are *all* items currently in the order (Condition 2). On the other hand, since *order* — *employee* is a many-to-one relationship, if the activities with reference type order mention an employee, they mention the *parent* of the relationship, and an event mentioning another order o_2 with p_1 does not delete the link between o_1 and p_1; however, when in event $\#_6$ the order o_1 occurs with another employee p_2, we assume that the link (o_1, p_1) gets deleted (Condition 1). We thus require for our approach the minimal background knowledge about the process as defined next:

Definition 7. *For a set of object types Σ and a set of activities $\mathcal{A}$, a* relationship summary *is a tuple $\mathcal{R} = (m2o, reftype)$ where $m2o \subseteq \Sigma^2$ is a set of many-to-one relations, and $reftype \colon \mathcal{A} \to \Sigma$ maps each activity to a reference type.*

A last point of attention pertains a "locality" principle for activities: when an event occurs, what is the scope of the changes its corresponding activity can induce? Intuitively, one may expect that the scope is that of the reference object, i.e., only links referring to it can be manipulated, either adding/removing child objects from a type to which it relates via a one-to-many relationship, or changing its parent in a many-to-one relationship.

However, we need to ensure that children in a one-to-many relationship have only one parent at a time, and the (implicit) parent updates needed to enforce that, may break this locality principle, as witnessed by the following example.

Example 5. Consider the following OCEL recording teams of multiple persons, with a many-to-one relationship from *person* to *team* (i.e., a person can be a member of only one team). Reference types are underlined in all activity names.

	activity	team	person
$\#_1$	create team	t_1	p_1, p_2
$\#_2$	create team	t_2	p_3, p_4
$\#_3$	add person to team	t_2	p_2
$\#_4$	create team	t_3	p_1, p_3

According to Assumptions 3 and 2, after $\#_3$ all of p_2, p_3, p_4 belong to team t_2 (note that the reference type of $\#_3$ is *person*). However, p_2 is at this point already a member of team t_1. Therefore, to maintain the multiplicity constraints of the many-to-one relationship, we need to conclude that the relationship (t_1, p_2) is implicitly deleted by $\#_3$. A similar problem occurs in $\#_4$, where persons p_1 and p_3 are added to a team even being already member of another one: also here an implicit deletion would be required to maintain the one-to-many relationship. However, while the implicit deletion in $\#_3$ pertains the former parent of the reference object, the implicit deletions needed in $\#_4$ pertain (former) parents of children of the reference object. So the latter case violates the locality principle.

Ex. 5 shows that the locality principle must be enforced as an additional assumption on the input OCEL.

Assumption 5 *Every event can only modify properties of its reference object.*

Specifically, this assumption avoids the need of implicit deletions as in $\#_4$ of Ex. 5, which is important for correctness of our algorithm presented in the next section: Whenever the reference object of an event e lies on the one-side of a one-to-many relationship, it must not be the case that any of the many-side objects $o_1, \ldots, o_k$ mentioned in e are already related to another object of the same reference type.

In the remainder of this paper, we will use the following notion to express that a log is compatible with a relationship summary:

Definition 8. *A log is* well-formed *wrt. relationship summary $\mathcal{R} = (m2o, reftype)$ if L satisfies Assumptions 1–5 according to the reference types reftype and the relationships $m2o$. To ensure that all $(\sigma, \tau) \in m2o$ are many-to-one relations in L, we require that all events with reference type τ (i.e., one-type) mention at most one object of type σ.*

Algorithm 1 Event classification

Input: log L that is well-formed wrt. relationship summary $\mathcal{R}$, types $\sigma, \tau \in \Sigma$
Output: labelling function $Lab : E^L \to \mathcal{P}(RL)$
$R = \emptyset$
for $e \in E^L_{\tau,\sigma}$ **do** ▷ ordered by timestamps
  let u be the object of type σ in e
  $Lab(e) = \emptyset$
  if e has reference type σ **then**
    let $O_e = \{o_1, \ldots, o_k\}$ be all objects of type τ in e
    for each object $o_i \in O_i$ **do**
      if $(o_i, u) \in R$ **then**
        add maintain to $Lab(e)$
      else
        add (o_i, u) to R , add create to $Lab(e)$
    $D = \{o \mid (o, u) \in R \text{ but } o \notin O_e\}$
    if $D \neq \emptyset$ **then**
      remove (o, u) from R, for all $o \in D$
      add delete to $Lab(e)$
  else ▷ reference type of e is τ
    let o be the single object of type τ in e
    if $(o, u) \in R$ **then**
      add maintain to $Lab(e)$
    else if $(o, u') \in R$ for some $u \neq u'$ **then**
      remove (o, u') from R and add (u, o)
      add update_parent to $Lab(e)$
    else
      add (o_i, u) to R , add create to $Lab(e)$

4 Detection of Relationship Patterns

We assume a log L with relationship summary $\mathcal{R} = (m2o, \mathit{reftype})$ as input. Below, we focus on a single many-to-one relationship $(\tau, \sigma) \in m2o$, where σ is the one-type and τ is the many-type; for multiple relationships, the procedure can be iterated. Let $Act_{\tau,\sigma}$ be the set of those activities in L for which each event involves an object of type σ and an object of type τ, and $E_{\tau,\sigma}$ be a list of all events in L with activity in $Act_{\tau,\sigma}$, ordered by timestamp. Our aim is to label first events and in a second step activities according to whether they create, maintain, or delete links between objects of type τ and objects of type σ. To this end, let $RL = \{\mathsf{create}, \mathsf{maintain}, \mathsf{update\_parent}, \mathsf{delete}\}$ be the set of relationship labels. Below, we refer to the set of events of a log L as E^L.

Our labelling procedure is presented in Alg. 1. Importantly, it maintain a set R that contains all current links, i.e., all pairs of objects currently in the $\tau - \sigma$ relationship. While processing the events in $E_{\tau,\sigma}$, Alg. 1 then updates the current relation R and assigns labels to events. Following Assumption 4, a case distinction is performed according to whether the reference type of e is σ or τ.

In the former case, if there is some object o_i of type τ mentioned in e such that $(o_i, u) \in R$, i.e., the object pair is already in the relation, we add label maintain

to e (Line 11). For each o_i s.t. (o_i, u) does not currently occur in R, we add it, and extend the label set with create (Line 13). In addition, if there are objects o of type τ that were up to now in relationship with u (i.e., $(o, u) \in R$) but do not occur in e, we remove them from R and add label delete for e (Line 17).

If the reference type of e is τ, only a single object o of type τ occurs in e by well-formedness of L. Similar to the first case, we add label maintain resp. create if (o, u) is already in R resp. o has no binding in R (Lines 21 and 26). If instead o already has a binding (o, u') in R, we perform an *implicit deletion* to maintain the relationship as many-to-one, i.e., remove (o, u') from R, add (o, u) instead, and remember this special case with label update_parent (Line 24).

Note that Assumption 5 could be operationally checked in Alg. 1, by verifying in Line 13 that object o_i has no other binding in R. We obtain the next key property, stating that the reconstructed relation R is indeed many-to-one. This is shown by induction on the number of loop iterations, using Assumption 5.

Lemma 1. *At any point in time, for every object o of type τ, there is at most one object u of type σ such that $(o, u) \in R$.*

Example 6. Consider again Ex. 4. with relationship *order — item*. In the next table, for all events mentioning both types, event labels are given, as well as the state of R when processing the log according to Alg. 1. Here $R_3 := \{(i_1, o_1), (i_3, o_1)\}$ and $R_4 := R_3 \cup \{(i_4, o_2), (i_5, o_2)\}$. Reference type is always *order*.

	activity	order	item	employee	R	event labels
$\#_1$	create order	o_1	i_1, i_2	p_1	$\{(i_1, o_1), (i_2, o_1)\}$	{create}
$\#_2$	reduce order	o_1	i_1		$\{(i_1, o_1)\}$	{delete, maintain}
$\#_3$	add to order	o_1	i_1, i_3		R_3	{create, maintain}
$\#_4$	create order	o_2	i_4, i_5	p_1	R_4	{create}
$\#_7$	ship order	o_1	i_1, i_3		R_4	{maintain}
$\#_8$	ship order	o_2	i_4, i_5		R_4	{maintain}

Example 7. Consider the following log dealing with teams of multiple persons, with a many-to-one relation *person — team* (i.e., each person can only be a member of one team). The reference types are underlined in all activity names.

	activity	team	person	R	
$\#_1$	create team	t_1	p_1, p_2	R_1	{create}
$\#_2$	add person to team	t_1	p_3	R_2	{create}
$\#_3$	create team	t_2	p_4, p_5	R_3	{create}
$\#_4$	add person to team	t_2	p_3	R_4	{create, update_parent}
$\#_5$	add person to team	t_1	p_6	R_5	{create}
$\#_6$	notify team	t_1	p_1, p_2, p_6	R_5	{maintain}

Here $R_1 := \{(p_1, t_1), (p_2, t_1)\}$, $R_2 := R_1 \cup \{(p_3, t_1)\}$, $R_3 := R_2 \cup \{(p_4, t_2), (p_5, t_2)\}$, $R_4 := (R_3 \setminus \{(p_3, t_1)\}) \cup \{(p_3, t_2)\}$, and $R_5 := R_4 \cup \{(p_6, t_1)\}$. Note how in $\#_4$ person p_3 is added to t_2, so since p_3 can only be member of one team, it is removed from t_1, which is reflected in R_4. Also later, when all team members are notified in $\#_6$, p_3 is not included.

Finally, we define the *relationship labels* of an activity $a \in Act_{\tau,\sigma}$. The labels of an activity represent which combination of creation, maintenance, and deletion patterns can occur in events with this activity.

Definition 9. *For $a \in Act_{\tau,\sigma}$, let the* relationship labels *of a be defined as $RL(a) = \{Lab(e) \mid e$ occurs in L with activity $a\}$.*

5 Discovery of OPIDs with Dynamic Relations

As a starting point of our approach, we assume that from a log L, an OCPN N_0 was already discovered, e.g. using the technique from [17], and N_1 is the equivalent OPID obtained after applying translation $\mathbb{T}_1$ from [13], i.e., $N_1 = \mathbb{T}_1(N_0)$. We require that the discovered OCPN N_0 has unique activity labels, which does by definition of $\mathbb{T}_1$ propagate to N_1. Below, we assume a pair (τ, σ) of object types as input, for which a dynamic many-to-one relationship should be tracked in the OPID. If multiple dynamic relationships are to be tracked, the following approach to augment the OPID is applied for each pair of types.

Let $N_1 = (\Sigma, P, T, F_{in}, F_{out}, color, \ell)$ be an OPID, while one-type σ and many-type τ are the types to which the transformation should be applied.

(1) **Link place.** We add a fixed *link place* $p_{\tau,\sigma}$, similar as in [13].

(2) **Copies of transitions**. For each transition t with activity $a \in Act_{\tau,\sigma}$, the transition before transformation looks like in Fig. 1 (a), possibly with additional input/output places with types different from σ and τ. We remove t itself and add instead *for every label set $B \in RL(a)$* a transition t_B, that implements the transition gadget in Fig. 1 (b). However, the arrows included depend on the labels in B, as follows:
   - if $\mathsf{create} \in B$, then the arrows labeled C resp. (u, C) are included;
   - if $\mathsf{delete} \in B$ then the arrow labeled (u, D) is included; and
   - if $\mathsf{update\_parent} \in B$, then the arrows labeled r, (w, r) and (u, r) are included; and
   - if $\mathsf{maintain} \in B$, then the arrows labeled M resp. (u, M) are included.

Any additional incoming/outgoing arcs to places with types different from σ and τ are maintained. Note that in step (2) for each label set $B \in RL(a)$, a separate copy t_B is created (so N_1 can have several copies of a transition in N_0). The relationship place $p_{\tau,\sigma}$ is shared, though. Let this transformation be called $\mathbb{T}_D$.

The next theorem shows that the OPID obtained by the transformation $\mathbb{T}_D$ replays the log L. To that end, we say that an OCPN N_0 *replays* a trace graph $G_X = \langle E_X, D_X \rangle$ with object set O if N_0 has an accepted run $M^O_{init} \xrightarrow{(t_1,b_1)} \cdot \xrightarrow{(t_2,b_2)} \cdots \xrightarrow{(t_n,b_n)} M^O_{final}$ such that there is a bijection $g\colon \{(t_1, b_1), \ldots, (t_n, b_n)\} \to E_X$ such that (i) if $g(t_i, b_i) = e$ for some event e with activity a and object set O_e then $\ell(t_i) = a$ and $O_e = range(b_i)$, (ii) for all $(e, e') \in D_X$ there are $1 \leq i < j \leq n$ such that $g(t_i, b_i) = e$ and $g(t_j, b_j) = e'$, and (iii) for all $1 \leq i < j \leq n$ such that $g(t'_i, b_i) = e_i$ and $g(t'_j, b_j) = e_j$, $\pi_{time}(e_i) \leq \pi_{time}(e_j)$.

Theorem 1. *Let G_X be a trace graph from L and $N = \mathbb{T}_D(\mathbb{T}_1(N_0))$ an OPID obtained from an OCPN N_0. If N_0 can replay G_X, then N has an accepted run*

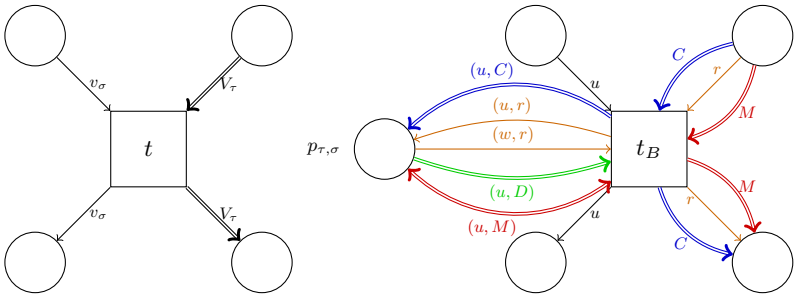


Fig. 1. Extending a transition t with capabilities to track a dynamic relationship.

$M_\emptyset \xrightarrow{*} M_0 \xrightarrow{(t_1,\beta_1)} M_1 \xrightarrow{(t_2,\beta_2)} \cdots \xrightarrow{(t_n,\beta_n)} M_n \xrightarrow{*} M_\emptyset$ *that admits a bijection* $f\colon \{(t_1,\beta_1),\ldots,(t_n,\beta_n)\} \to V_G$ *such that*

- *if* $f(t_i,\beta_i) = (a, O)$ *then* $\ell(t_i) = a$ *and* $O = range(\beta_i) \setminus \{\beta_i(d) \mid d \in D \cup \{w\}\}$, *and*
- *for all* $(e, e') \in E_G$ *there are* $1 \leq i < j \leq n$ *such that* $f(t_i,\beta_i) = e$ *and* $f(t_j,\beta_j) = e'$,

and $M_\emptyset \xrightarrow{*} M_0$ *and* $M_n \xrightarrow{*} M_\emptyset$ *fire only emitting and consuming transitions.*

6 Discussion and Conclusion

In this work, we have introduced the first, general approach to discover object-centric processes with dynamically evolving binary relationships. To this end, we identified standard assumptions underlying OCELs, to be able to unambiguously reconstruct the relationship dynamics from the recorded event data. We then presented an approach to identify essential net patterns that govern the creation, deletion, and propagation of links according to such relationships. Building on these patterns, we obtained the first discovery technique capable of revealing dynamically changing relationships in OPIDs.

The techniques has been fully implemented [8] in Python based on `pm4py`. As input, the tool takes an OCEL together with a small file in json format, which represents the relationship summary $\mathcal{R}$ (cf. Def. 7). The tool first discovers an OCPN from the given OCEL using the implementation in `pm4py` [17], and applies afterwards the transformation $\mathbb{T}_1$ from [13] to obtain an OPID N_1. By applying the pattern recognition approach in Alg. 1 to the given OCEL and the relationships mentioned in $\mathcal{R}$, activity labellings are obtained, which are in the final step used to augment the OPID N_1 with capabilities to dynamically track relations as described in Sec. 5. The approach thus combines the extensively validated approach from [17], with a computationally light post-processing step.

In future work, we plan to extend our approach to *data-aware* OPIDs (DOPIDs) [10], but also generalise it to handle n-ary relationships with richer cardinality constraints. A more radical continuation is to define techniques where the recorded event data obey to the parameter semantics, instead of the snapshot one (see Section 3). There is only one approach, developed in a completely different setting, that may provide a basis to attack this problem [11].

References

1. 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: Hildebrandt, T., Van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) Business Process Management, vol. 11675, pp. 139–156. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-26619-6_11
2. Berti, A., et al.: OCEL (Object-Centric Event Log) 2.0 Specification (2024)
3. Chaki, R., Calvanese, D., Montali, M.: Generation of timelines from event knowledge graphs using domain knowledge. In: Saeed, K., Dvorský, J., Fukumoto, M., Nishiuchi, N. (eds.) Computer Information Systems and Industrial Management - 24th International Conference, CISIM 2025, Fukuoka, Japan, September 11-13, 2025, Proceedings. Lecture Notes in Computer Science, vol. 15927, pp. 275–289. Springer (2025). https://doi.org/10.1007/978-3-032-02406-0_20, https://doi.org/10.1007/978-3-032-02406-0_20
4. Fahland, D.: Process Mining over Multiple Behavioral Dimensions with Event Knowledge Graphs, Lecture Notes in Business Information Processing, vol. 448. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-08848-3_9
5. Fahland, D.: Describing Behavior of Processes with Many-to-Many Interactions. In: Proc. 40th PETRINETS. LNCS, vol. 11522, pp. 3–24. Springer (2019)
6. Ghahfarokhi, A.F., et al.: OCEL: A Standard for Object-Centric Event Logs. In: New Trends in Database and Information Systems (Proc. ADBIS 2024). pp. 169–175. Springer (2021)
7. Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: Petri net-based object-centric processes with read-only data. Information Systems **107**, 102011 (2022)
8. Gianola, A., Hameed, Z., Montali, M., Seidel, A., Weske, M., Winkler, S.: Discovering opids with dynamic relationships: Implementation (2025), https://github.com/zeeshanhameed12/OPID_Discovery
9. Gianola, A., Montali, M., Winkler, S.: Object-Centric Conformance Alignments with Synchronization. In: Proc. 36th CAiSE. LNCS, vol. 14663, pp. 3–19. Springer (2024)
10. Gianola, A., Montali, M., Winkler, S.: Object-centric processes with structured data and exact synchronization - formal modelling and conformance checking. In: Proc. 37th CAiSE. LNCS, vol. 15702, pp. 185–202. Springer (2025). https://doi.org/10.1007/978-3-031-94571-7_11, https://doi.org/10.1007/978-3-031-94571-7_11
11. Gösgens, J., Jansen, N., Geffner, H.: Learning lifted STRIPS models from action traces alone: A simple, general, and scalable solution. CoRR **abs/2411.14995** (2024). https://doi.org/10.48550/ARXIV.2411.14995, https://doi.org/10.48550/arXiv.2411.14995
12. Liss, L., Adams, J.N., van der Aalst, W.M.P.: TOTeM: Temporal Object Type Model for Object-Centric Process Mining. In: Proc. Business Process Management Forum. pp. 107–123. Springer (2024)
13. Seidel, A., Winkler, S., Gianola, A., Montali, M., Weske, M.: To bind or not to bind? discovering stable relationships in object-centric processes. In: Proc. 44th ER 2025. LNCS, vol. 16189, pp. 223–241 (2025). https://doi.org/10.1007/978-3-032-08623-5_12
14. van der Aalst, W.M.P.: Object-Centric Process Mining: Dealing with Divergence and Convergence in Event Data. In: Software Engineering and Formal Methods. pp. 3–25. Springer (2019)

15. van der Aalst, W.M.P.: Toward more realistic simulation models using object-centric process mining. In: Proc. 37th ECMS. pp. 5–13 (2023)
16. van der Aalst, W.M.P., et al.: Process Mining Manifesto. In: Proc. Business Process Management Workshops. Springer (2012)
17. van der Aalst, W.M.P., Berti, A.: Discovering Object-centric Petri Nets. Fundamenta Informaticae **175**(1-4), 1–40 (2020)
18. van der Werf, J.M.E.M., Rivkin, A., Montali, M., Polyvyanyy, A.: Correctness Notions for Petri Nets with Identifiers. Fundamenta Informaticae **190**(2-4), 159–207 (2024)
19. van Detten, J.N., Schumacher, P., Leemans, S.J.J.: Modeling and discovering dynamic identity relations in object-centric process mining. In: Proc. 7th ICPM (2025)