Object-Centric Local Process Models

Viki Peeva(✉), Marvin Porsil(✉), and Wil M. P. van der Aalst(✉)

Chair of Process and Data Science, RWTH Aachen University, Aachen, Germany
{peeva,wvdaalst}@pads.rwth-aachen.de, marvin.porsil@rwth-aachen.de

Abstract. Process mining is a technology that helps understand, analyze, and improve processes. It has been present for around two decades, and although initially tailored for business processes, the spectrum of analyzed processes nowadays is evermore growing. To support more complex and diverse processes, subdisciplines such as object-centric process mining and behavioral pattern mining have emerged. Behavioral patterns allow for analyzing parts of the process in isolation, while object-centric process mining enables combining different perspectives of the process. In this work, we introduce *Object-Centric Local Process Models* (OCLPMs). OCLPMs are behavioral patterns tailored to analyzing complex processes where no single case notion exists and we leverage object-centric Petri nets to model them. Additionally, we present a discovery algorithm that starts from object-centric event logs, and implement the proposed approach in the open-source framework ProM. Finally, we demonstrate the applicability of OCLPMs in two case studies and evaluate the approach on various event logs.

Keywords: Local process models · Behavioral patterns · Pattern mining · Object-centric process mining · Object-centric event logs

1 Introduction

Process mining takes event data generated as a byproduct of organizations' operations and provides insights and improvements of the analyzed process. This is achieved by automatically discovering process models, computing conformance checking metrics, or enhancing the model with concrete KPIs. Traditional process mining considers the process from start to end and uses a single case notion. However, in reality, the process interacts with various entities, in the community known as object types or artifacts. There exist different strategies how to connect or model such entities together with the control-flow of the process. In our work, we focus on *object-centric process mining* as described in [2] and *Object-Centric Event Logs* (OCELs) [14]. Moreover, process issues like delays, high costs, etc., almost never occur on a global level but in specific subcontexts, requiring pattern mining. Pattern mining is a known discipline in data science and the concept has also been established in the area of process mining with

We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

A. Delgado and T. Slaats (Eds.): ICPM 2024 Workshops, LNBIP 533, pp. 376–388, 2025.
https://doi.org/10.1007/978-3-031-82225-4_28

works for discovering frequent subsequences [6], episodes [18], and local process models [23]. While end-to-end process models describe the entire process, process or behavioral patterns only explain (match) particular sub-behaviors of the process. In particular, the proposed approach focuses on *Local Process Models* (LPMs), which are a type of behavioral pattern, allowing for constructs such as sequence, choice, concurrency, and loop.

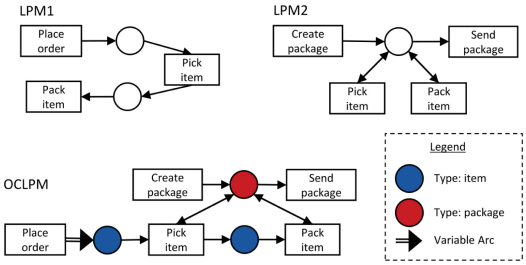
To combine the two areas, in this paper, we define *Object-Centric Local Process Models* (OCLPMs) as a behavioral pattern alternative for object-centric process mining. Additionally, we build a framework around an already existing LPM discovery approach to discover such OCLPMs. The discovery approach is built upon the formalisms of Petri nets, and as a result, the discovered OCLPMs are represented as object-centric Petri nets (OCPNs). Specifically, we list the following contributions:

(1) Adapting existing LPM discovery approach for OCLPM discovery.
(2) Implementing the algorithm in the publicly accessible framework ProM.
(3) Demonstrating feasibility and applicability in real-world scenarios.

The rest of the paper is structured as follows. First, we illustrate the necessity for OCLPMs in Sect. 2. Then, we present related work in Sect. 3, and give the necessary background to follow the rest of the paper in Sect. 4. In Sect. 5, we describe the proposed framework and all the surrounding details, after which, Sect. 6 covers the experiments demonstrating its applicability. In Sect. 7, we discuss the strengths and weaknesses of the proposed approach. Finally, in Sect. 8 we conclude the paper and offer an outlook on possible extensions.

2 Motivating Example

The necessity of pattern mining for processes has been demonstrated and discussed in previous works [22,23]. Challenges like spaghetti and flower process models make behavioral patterns even more attractive. However, current pattern representations lack the ability to model the process from multiple viewpoints. We use the example depicted in Fig. 1 to show the benefits of having patterns that are object-centric aware. The excerpt event log is for an order management process and includes ten events of five different activities and three object types. To discover traditional LPMs on such object-centric event logs, we would choose one object type and focus on the viewpoint of the chosen object type. From the perspective of each item, the process starts with *Place order*. However, in the process, one *Place order* is executed for more items, meaning one *Place order* event is followed by multiple *Pick item* and *Pack item* events. This can not be caught by the model because of replicating events, also called *convergence*. Moreover, for each item, first *Pick Item* and then *Pack Item* occurs. However, from the perspective of the package, it would appear there are random interleaving of the two activities. Therefore, resulting in unconnected loops in the model, see *lpm*2 in Fig. 1, called *divergence*.



activity	order	item	package	case
Place order	o1	i1, i2		c1
Create package			p1	c1,c2
Pick item		i1	p1	c1
Pack item		i1	p1	c1
Place order	o2	i3		c2
Pick item		i2	p1	c1
Pick item		i3	p1	c2
Pack item		i2	p1	c1
Pack item		i3	p1	c2
Send package			p1	c1,c2

Fig. 1. Event log excerpt with example LPMs from the perspectives of the items ($lpm1$) and packages ($lpm2$) and one OCLPM depicting both perspectives.

By using OCLPMs and modeling the multiple perspectives together resolves the aforementioned problems. Consider the OCLPM in Fig. 1. The execution of *Place order* resulting in multiple items is clearly depicted with the variable arc between *Place order* and *Pick item*, solving the convergence problem. Additionally, the order of *Pick item* and *Pack item* is explicitly represented in the OCLPM, avoiding divergence. Considering this, we conclude that OCLPMs are valuable for processes with multiple object types.

3 Related Work

Object-Centric Modeling. As indicated in the introduction, there exist multiple strategies for modeling control-flow together with the associated data participators. Artifact-centric approaches [7,9,20] model the business process in terms of business artifacts. Such artifacts contain data and change states. A state change can result because of a step in the process, or it can trigger an advance in the process. Flexible case modeling [16,17] breaks the process into a domain model defining the participating entities, object lifecycles, process fragments to model the control-flow, and a goal state. Proclets [1,11] model the control flow from the perspective of the different object types and can have synchronization points between them. The synchronization points allow for one-to-one and one-to-many relationships. Although powerful, not many techniques in process mining are able to work with such models. Event-knowledge graphs and object-centric process models [2,13] are newer paradigms that put the control-flow in the main focus, but now from the perspective of multiple key entities, known as object types. For more information regarding the different modeling strategies, we refer to [2,11]. However, the more involved modeling strategies can be quite complex, and the ones focusing on control-flow can result in spaghetti models.

Behavioral Patterns. All modeling strategies above, represent the entire process in one model. However, processes can be complex, resulting in complicated over-fitting, spaghetti, models or simple, but under-fitting, flower, models. Some approaches, alleviate the complexity by introducing scenarios or fragmenting the

process [12,15]. However, these still have the goal of representing the entire process. Behavioral patterns, like sequences, episodes, and local process models [4,6,8,21–23], deal with complexity by modeling subparts of the process and ignoring the rest. This makes them suitable for a spectrum of applications and not only modeling the process (see [22]). An alternative are declarative process modeling languages [3] that instead of using the activities to model control-flow, they define a set of constraints between the activities, like precedence or response. There also exist object-centric extensions for declarative constraints [19], where constraints are related to object types. However, with this work, we extend local process models as in [22] to be object-centric [2].

4 Preliminaries

We define sets ($X = \{a, b\}$), multisets ($M = [a^2, b^3]$), sequences ($\sigma = \langle a, b, c \rangle$ where $\sigma_1 = a$), and tuples ($t = (a, b, c)$). Given a set X, $\mathcal{P}(X)$ is the power set of X, and X^* represents the set of all sequences over X. We use $f(X) = \{f(x) \mid x \in X\}$ ($f(\sigma) = \langle f(\sigma(1)), f(\sigma(2)), \ldots, f(\sigma(n)) \rangle$) to apply the function f to every element in the set X (the sequence σ). Finally, we write $f_{\restriction X}$ ($\sigma_{\restriction X}$) to denote the projection of the domain of function f (the sequence σ) onto X.

Event Logs. Collected data used for process analysis is transformed into *event logs*. In Definition 1, we define events, and in Definition 2, we formally define *event logs* that can be used for both traditional and object-centric processes.

Definition 1 (Event). *Let $\mathcal{U}_{ev}$ be the universe of events, $\mathcal{U}_{ot}$ the universe of object types, and $\mathcal{U}_{oi}$ the universe of object identifiers. We define the event $e = (ei, act, time, omap, vmap)$, such that $\pi_{ei}(e) = ei$ is the event id, $\pi_{act}(e) = act$ is the event activity, $\pi_{time}(e) = time$ is the timestamp of the event, $\pi_{omap}(e) = omap$ ($omap \in \mathcal{U}_{ot} \rightarrow \mathcal{P}(\mathcal{U}_{oi})$) is a function mapping each object type to the objects involved in the event, and $\pi_{vmap}(e) = vmap$ is a function assigning values to each of the event attributes.*

Definition 2 (Event Log [2]). *$L = (E, \preceq_E) \in \mathcal{U}_L$ is an event log with $E \subseteq \mathcal{U}_{ev}$ and $\preceq_E \subseteq E \times E$ such that:*

- *$\preceq_E$ defines a partial order (reflexive, antisymmetric, and transitive)*
- *$\forall_{e_1,e_2 \in E}\, \pi_{ei}(e_1) = \pi_{ei}(e_2) \implies e_1 = e_2$, and*
- *$\forall_{e_1,e_2 \in E}\, e_1 \preceq_E e_2 \implies \pi_{time}(e_1) \leq \pi_{time}(e_2)$.*

We use OT_L to denote all object types in L.

In process mining, the concept of process executions is essential for many techniques. Depending on the complexity, process executions are represented as totally or partially ordered events. For example, in traditional process mining, event logs are represented as a set of traces, where each trace is a sequence of events. Here, we call such event logs *simple event logs* and write $L_S \in \mathcal{P}(\mathcal{U}_{ev}^*)$.

To obtain a simple event log L_S from $L = (E, \preceq_E) \in \mathcal{U}_L$, we group the events on a certain object type $ot \in \mathcal{U}_{ot}$ and order them. For this, we use $flat \in \mathcal{U}_L \times \mathcal{U}_{ot} \to \mathcal{P}(\mathcal{U}_{ev}^*)$ such that $flat(L, ot) = \{\rho_{cid} \in \mathcal{U}_{ev}^* \mid cid \in \bigcup_{e \in E} \pi_{omap}(e)(ot) \wedge \forall_{e \in E}(cid \in \pi_{omap}(e)(ot) \implies e \in \rho) \wedge \forall_{1 \leq i < j \leq |\rho|}(\pi_{time}(\rho_i) \leq \pi_{time}(\rho_j))\}$. In case two events have the same timestamp, we assume some order.

Process Models. The behavior recorded in logs can be modeled using different notations, such as DFG, process trees, BPMN, Petri nets, etc. In this work, we focus on Petri nets, and more precisely on *labeled Petri nets* which we define in Definition 3 and *object-centric Petri nets* as defined in Definition 4.

Definition 3 (Labeled Petri Net). *A* labeled Petri net *is a tuple* $N = (P, T, F, l)$ *with* P *the set of places,* T *the set of transitions, such that* $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ *the flow relation and* $l \in T \to \mathcal{A} \cup \{\tau\}$ *a labeling function.*

Definition 4 (Object-centric Petri nets). *An* Object-centric Petri net *is a tuple* $ON = (N, pt, F_{var})$ *where* $N = (P, T, F, l)$ *is a labeled Petri net,* $pt \in P \to \mathcal{U}_{ot}$ *maps places onto object types, and* $F_{var} \subseteq N.F$ *is the subset of variable arcs.*

Generally, process models define a language, commonly used in process mining to measure how well the model aligns with the collected event data. For both labeled and object-centric Petri nets, exist notions like tokens and markings, necessary to define their language. Because of space restrictions, we refer to [2] on how the language is obtained.

In this work, we consider LPMs as labeled Petri nets and OCLPMs to be OCPNs. However, LPMs and OCLPMs do not cover the entire event log, since they do not represent the entire process. Therefore, we define $E_{oclpm} \subseteq E$ to be the events in the event log $L = (E, \preceq_E)$ that are covered by the OCLPM *oclpm*. More details about how LPMs are matched to event logs, can be found in previous works [22,23].

5 OCLPM Discovery

In this section, we present a two-phase discovery approach for OCLPMs given an OCEL. In Fig. 2 we visualize the two phases of the approach: *preparation* and *discovery*, together with the input, output, and the intermediate results. In Algorithm 1.

5.1 Preparation (Phase 1)

The preparation covers the first two lines in Algorithm 1. As previously discussed, the discovered OCLPMs should describe local patterns occurring in the process but also incorporate object interactions. Therefore, with the preprocessing, we extract dominant local dependencies between activities for each object type (place net discovery) on the one hand, and identify meaningful object interactions (event log transformation), on the other.

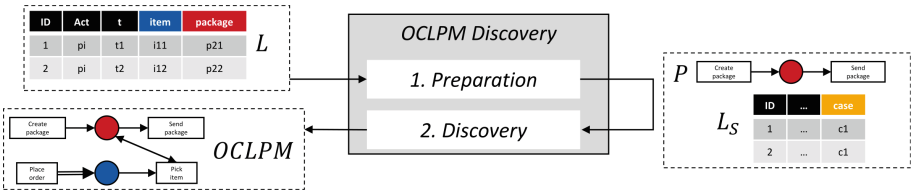


Fig. 2. Overview of the steps in the OCLPM framework, depicted with their inputs and outputs.

Algorithm 1: Discovery algorithm for OCLPMs.

input : $L = (E, \preceq_E)$
1 // Preparation (Phase 1)
2 $PT \leftarrow \bigcup_{ot \in OT_L} \texttt{po}(\texttt{flat}(L, ot)) \times \{ot\}$;
3 $L_S \leftarrow \texttt{flat}(\texttt{peo}(L))$;
4 // Discovery (Phase 2)
5 $P \leftarrow \{N_p \mid (N_p, ot) \in PT\}$;
6 $LPM \leftarrow \texttt{lpmd}(L_S, P)$;
7 $OCLPM \leftarrow \emptyset$;
8 **for** $lpm \in LPM$ **do**
9 $\quad pt \leftarrow \{(p, ot) \mid p \in lpm.P \wedge (lpm_{\restriction_p}, ot) \in PT\}$[1];
10 $\quad F_{var} \leftarrow \texttt{vararc}(lpm, pt, L_S)$;
11 $\quad OCLPM \leftarrow OCLPM \cup \{(lpm, pt, F_{var})\}$;
12 **return** $OCLPM$

Place Net Discovery. We model local dependencies between activities by using place nets, i.e., the simplest LPMs containing only one place (see output of *Preparation* in Fig. 2). To obtain local dependencies for each object type, we flatten the event log for each object type (using *flat* from Sect. 4) and execute a place net oracle *po* on the flattened event log. As a place net oracle, we can use any discovery algorithm that returns a Petri net or a set of place nets. However, algorithms unrestricted to end-to-end trace fitness, as discussed in [22], are more suitable when we are interested in local dependencies. The result is the set PT, a set of place nets together with the object type for which they were discovered (Line 2 in Algorithm 1).[2]

Event Log Transformation. To focus on object interactions, we enhance the event log with a new object type, used to group interacting objects, whose goal is to mimic the case notion from traditional event logs. Each object of the new object type represents a group of original objects that have met during the process. By then flattening the event log on the newly added object type, we extract process executions representing the process from the viewpoint of the interacting objects.

[2] $lpm_{\restriction_p} = (\{p\}, T_p, F_p, l_p)$ such that $T_p = \{t \in lpm.T \mid (t, p) \in lpm.F \vee (p, t) \in lpm.F\}$, $F_p = \{(x, y) \in lpm.F \mid x = p \vee y = p\}$, and $l_p = lpm.l_{\restriction_{T_p}}$.

In Algorithm 1, we compute such object type and enhance the event log with it, by assuming a process execution oracle (see *peo* in Line 3) and extract the process executions by flattening (see *flat* in Line 3). One way of discovering such object types and extracting process executions has been proposed in [5]. There are a multitude of ways one can define object interactions. One example is to consider sharing an event an interaction. Moreover, the new object type must not combine all interacting objects. It is up to the process execution oracle to decide what is an interaction and where to put the boundaries between the interacting objects. By abstracting from the concrete computation, we allow flexibility when it comes to which object interactions are meaningful.

5.2 Discovery (Phase 2)

In the discovery step, the event log given as input and the intermediate results from the preparation are used to build OCLPMs. First, we discover traditional LPMs (LPM discovery), and then construct the OCLPMs by enhancing the discovered LPMs with place to object type mapping (object type annotation) and variable arcs (variable arc identification). Below, we describe each in more detail and finish the algorithm by returning the set of computed OCLPMs.

LPM Discovery. We discover traditional LPMs on the simple event log we created by utilizing an existing LPM discovery technique [22]. The technique used, starts with a precomputed set of local dependencies, represented as place nets (Line 5 in Algorithm 1), and merges those into larger LPMs only when there is evidence in the provided event log that the LPM occurs in the process. Additionally, the occurrence of the LPM should be for interacting objects. Let us consider the OCLPM in Fig. 2. A claim that the OCLPM occurred in the event log means it occurred for related packages and items, and not random pairs of items and packages. The discovered set of LPMs *LPM* (Line 6 in Algorithm 1) satisfies this requirement because the simple event log focuses on interacting objects as explained above.

Object Type Annotation. An important advantage of OCLPMs is depicting object type interactions. Therefore, for each place of an LPM, we identify the object type they represent. Since we used the computed place nets for each object type P as a starting point for the LPM discovery, we use the original place net to object type mapping PT to compute the place to type mapping (Line 9 in Algorithm 1).

Variable Arc Identification. Variable arcs allow for modelling many-to-one interactions between objects of different types. Therefore, the final step is for each LPM to identify the variable arcs. In our approach, we identify variable arcs as proposed in [2]. More concretely, given an LPM (which is a labeled Petri net) $lpm = (P, T, F, l)$, we define $F_{var} = \{(p,\ \mathrm{t}) \in F \cap (P \times T) \mid score(l(t), pt(p)) < \tau\} \cup \{(t,\ \mathrm{p}) \in F \cap (T \times P) \mid score(l(t), pt(p)) < \tau\}$, where $score \in \mathcal{A} \times \mathcal{U}_{ot} \not\to [0, 1]$ computes the fraction of events of the specified activity that contain exactly one

object of the object type in question, as defined below, and τ is a user-defined threshold.

$$score(act, ot) = \frac{|\{e \in E_{oclpm} \mid \pi_{act}(e) = act \wedge |\pi_{obj}(e)(ot)| = 1\}|}{|\{e \in E_{oclpm} \mid \pi_{act}(e) = act\}|}$$

Note, for the variable arc computation, we use E_{oclpm} since OCLPMs do not cover all events in an event log. Finally, the *vararc* in Line 10 returns F_{var} computed as described before.

With this we covered all steps of the discovery algorithm.

6 Evaluation

In this section, we evaluate the proposed approach qualitatively and quantitatively. First, we demonstrate applicability on two case studies. Then, we report runtime statistics and count of discovered models for different event logs. This is the first method for discovering OCLPMs, so we are unable to compare with previous work. All experiments are performed using the open-access implementation we provide in the ProM framework[3]. As a place net oracle, we use the SPECpp plugin in ProM[4], and as an LPM discovery approach we use the approach from [22] also available in ProM[5]. For each event log, we build OCLPMs between 2 and 7 places, 3 and 10 transitions, all activities and discovered place nets, and window size 7.

6.1 Case Studies

BPI Challenge 2017. The event log is recorded from a loan application process of a Dutch financial institute [10]. It involves objects of types *application* and *offer*. The event log is available in both traditional xes and OCEL format, hence we use it to compare OCLPM discovery to LPM discovery on the same event log.

We discover LPMs using [22] on the traditional event log, and OCLPMs on the available OCEL with the proposed approach. In Fig. 3, we display two OCLPMs and two LPMs. The OCLPM at the top depicts how after an application is created and then accepted, a new offer is created and sent. The interaction between the application and offer object types is clearly shown in the discovered OCLPM. The highest-ranked LPM also shows the move from application to offer, denoted with activity names prefixed with A and O. However, this is achieved with preprocessing of the event log. Emphasizing object types visually and treating them as first-class citizens gives a much clearer picture of what the pattern is describing. Moreover, it is not just the visual appeal that is gained with OCLPMs, but also expressiveness. This is illustrated with the help of the

[3] https://github.com/promworkbench/ObjectCentricLPMs.
[4] https://github.com/promworkbench/SPECpp.
[5] https://github.com/promworkbench/LocalProcessModelDiscoveryByCombiningPlaces.

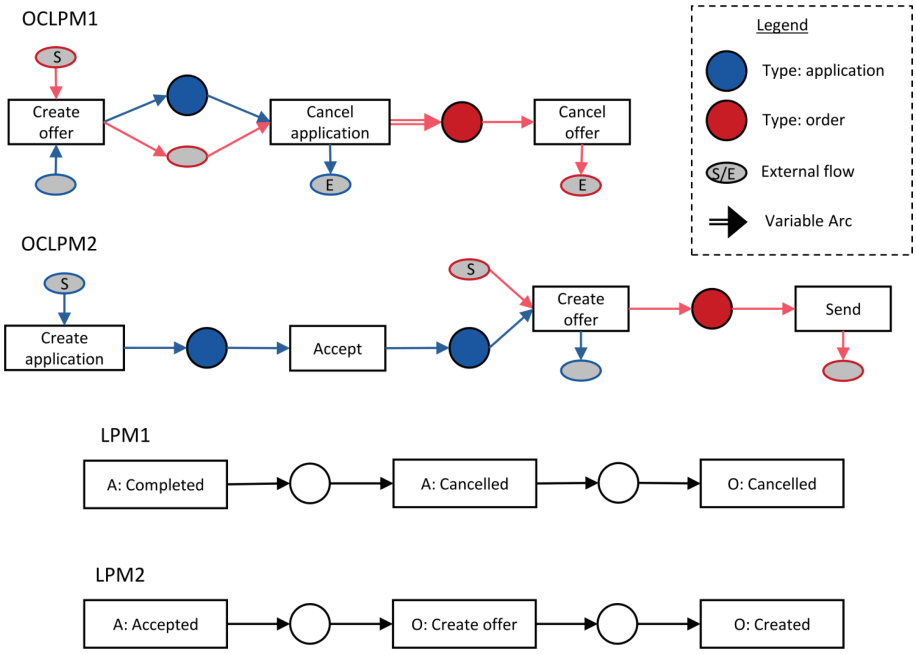
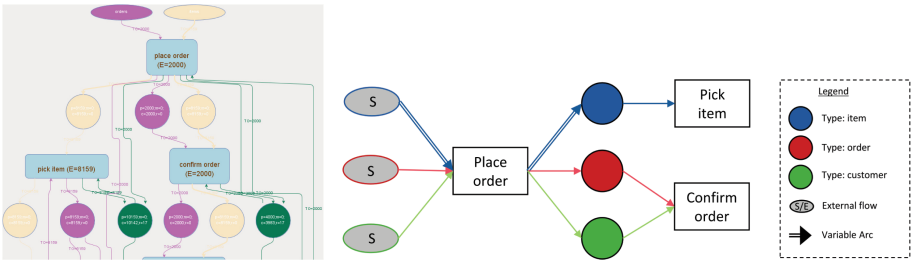


Fig. 3. Discovered OCLPMs with the proposed approach and LPMs with [22] on the BPIC2017 event log. In OCLPMs, we model external flow of the objects with elipses, where S/E denote object Start/End position.

OCLPM and LPM concerned with a cancellation of application. The shown OCLPM, with the help of variable arcs, clearly illustrates that one application might connect to multiple offers. Meaning, once the application is cancelled, all of the offers should be cancelled as well. The discovered LPM, although depicting that offer should be cancelled after an application is cancelled, does not contain information on whether one or multiple offers are cancelled.

Order Management. The event log, as its name hints, describes a process for managing orders in which objects of types *order*, *item*, *package*, *customer*, and *product* are involved. The main flow of the process is that a customer places an order, which is then confirmed by an employee, continuing into two disentangled subprocesses. On the one hand, we have, collecting order items, packing them, and sending the package (possibly multiple times if the deliveries have failed) until a successful delivery, and on the other hand, paying the order and possibly sending multiple reminders before the payment was completed.

The approach proposed run about 40 s and discovered in total 375 models. We show the highest ranked OCLPM in Fig. 4b. The model clearly shows the relationship between orders, customer, and items. One order is made by one



(a) Part of the full end-to-end process model describing the same behavior as the OCLPM.

(b) Discovered OCLPM showcasing the interactions between items, orders, and customers, focusing on the activities *place order*, *pick item*, and *confirm order*.

Fig. 4. Part of the end-to-end model and one OCLPM for the Order Management event log.

Table 1. Event log description with number of discovered OCLPMs and runtime (https://www.ocel-standard.org/1.0/#eventlogs).

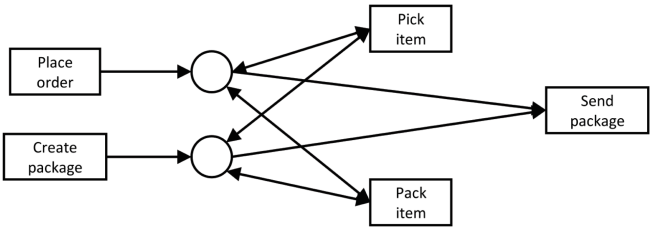
Name	Events	Objects	Object Types	Models	Runtime(s)
Order Management	22367	11521	5	846	161
O2C	98350	107767	19	1046	258
P2P	24854	74489	8	2923	195
Transfer	10319	2500	5	8	6
Recruiting	6980	1505	6	109	43
Github	1798	532	3	2314	92
BPIC2017	31203	8416	2	918	16

customer, and one order can correspond to multiple items. Furthermore, it shows that *place order* is a starting activity for orders, customers, and items.

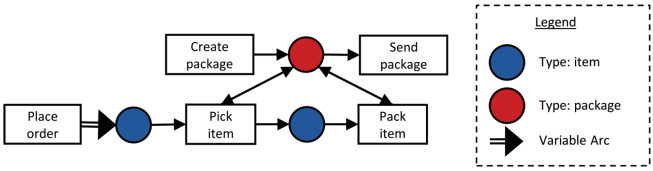
The end-to-end process model discovery took about 4 s to discover the original model. Although, the process itself is not too complex, the model was very cluttered and spaghetti like. In Fig. 4a, we show a part of the end-to-end process model. We had to filter on the orders, customers, and items object types, the activities, and discard most of the paths, to spot the relationships described by the OCLPM in Fig. 4b, despite it being frequent and highly-ranked. The approach returned additional 374 OCLPM, bringing valuable information that otherwise would have been lost in the complexity of the end-to-end model.

6.2 Results for Other Event Logs

In this section, we report different statistics regarding the discovery of OCLPMs on various OCEL. In Table 1 we list the event logs used in this part of the evaluation together with the number of events, objects, and object types they include. We run the *Object-Centric Local Process Model Discovery given OCEL*



(a) Example LPM for the motivational example log.



(b) Example OCLPM for the motivational example log.

Fig. 5. Example LPM and OCLPM for the motivational example log.

plug-in in *ProM* with default parameters. In Table 1, we report the number of models discovered and the time necessary to do so for each of the event logs. The three largest event logs *Order Management*, *O2C*, and *P2P* have the highest running time. We can also note that usually the runtime proportionally increases with the number of models built. One exception is the *Github* event log, that has more OCLPMs discovered in less time, compared to the *O2C* and the *Order Management* event logs. However, *Github* has significantly fewer events and objects than those two event logs.

7 Discussion

In this section, we cover the strengths and weaknesses of the proposed approach. An alternative to our approach, would be to directly use traditional LPM discovery on the flattened version of the event log and skip all the other steps we presented. In Fig. 5 we show one LPM and one OCLPM discovered for the event log given in Sect. 2. The obvious arguments in favour of OCLPMs, also exhibited in the evaluation of the approach, are the explicitness of object types and the expressiveness of the variable arcs. However, examining further, we compare both models from the perspectives of convergence and divergence. First, note that the missing dependency between *Pick item* and *Pack item* in the LPM is present in the OCLPM. This is due to the *Preparation* step of our approach, in which we discover local dependencies per object type. Second, the improper dependency between *Place order* and *Send package* is avoided in the OCLPM, again as a result of the *Preparation* step. Both of these examples demonstrate *divergence* problems for traditional LPMs discovered for OCELs and the ability of our approach to avoid them. Additionally, the improper dependency also creates the untrue impression that for each *Place order* a *Send package* is executed,

leading to *convergence* problems. In the OCLPM, the two activities are executed for different object types, allowing the OCLPM to be matched to one package and two orders as included in the log. In conclusion, the proposed approach resolves the convergence and divergence problems that would be introduced if a traditional LPM discovery was used.

8 Conclusion

In this work, we introduce OCLPMs as OCPNs, and present a discovery algorithm for building them from OCELs. We adapt and utilize existing work on computing local dependencies between activities and LPM discovery to support the OCLPM discovery. Moreover, we implement the proposed algorithm in the open-source process mining tool ProM and evaluate the usefulness of the proposed approach on one real-world and one artificial event log. Additionally, we report runtime statistics and discovered models on multiple event logs. Finally, we have discussed strengths and limitations of the proposed approach.

Next steps in this area would be to enhance the discovered OCLPMs with object and event attributes or allow for guided discovery similar to traditional LPMs. Furthermore, more elaborate filtering techniques for discarding or giving less weight on specific object types would allow more advanced discovery. Finally, exploring alternative methods for discovering OCLPMs would be valuable.

References

1. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Workflow modeling using proclets. In: CoopIS 2000, vol. 1901, pp. 198–209 (2000)
2. van der Aalst, W.M.P., Berti, A.: Discovering object-centric petri nets. Fundam. Informaticae **175**(1-4), 1–40 (2020)
3. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: balancing between flexibility and support. Comput. Sci. Res. Dev. **23**(2), 99–113 (2009)
4. Acheli, M., Grigori, D., Weidlich, M.: Efficient discovery of compact maximal behavioral patterns from event logs. In: CAiSE (2019)
5. Adams, J.N., Schuster, D., Schmitz, S., Schuh, G., van der Aalst, W.M.P.: Defining cases and variants for object-centric event data. In: ICPM 2022, pp. 128–135 (2022)
6. Agrawal, R., Srikant, R.: Mining sequential patterns. In: ICDE 1995, pp. 3–14 (1995)
7. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: BPM 2007, vol. 4714, pp. 288–304 (2007)
8. Brunings, M., Fahland, D., Verbeek, E.: Discover context-rich local process models (extended abstract). In: ICPM-D (2022)
9. Cohn, D., Hull, R.: Business artifacts: a data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull. **32**(3), 3–9 (2009)
10. van Dongen, B.: BPI challenge 2017 (2017)
11. Fahland, D.: Describing behavior of processes with many-to-many interactions. In: PETRI NETS 2019, vol. 11522, pp. 3–24 (2019)

12. Fahland, D.: Oclets - scenario-based modeling with petri nets. In: PETRI NETS 2009, vol. 5606, pp. 223–242 (2009)
13. Fahland, D.: Process mining over multiple behavioral dimensions with event knowledge graphs. In: Process Mining Handbook, vol. 448, pp. 274–319 (2022)
14. Ghahfarokhi, A.F., Park, G., Berti, A., van der Aalst, W.M.P.: OCEL: a standard for object-centric event logs. In: ADBIS 2021 Short Papers, vol. 1450, pp. 169–175 (2021)
15. Haarmann, S., Lichtenstein, T., Weske, M.: Fragment-based service choreographies. In: IEEE SCC 2022, pp. 164–173 (2022)
16. Haarmann, S., Montali, M., Weske, M.: Refining case models using cardinality constraints. In: CAiSE 2021, vol. 12751, pp. 296–310 (2021)
17. Hewelt, M., Weske, M.: A hybrid approach for flexible case modeling and execution. In: BPM Forum 2016, vol. 260, pp. 38–54 (2016)
18. Leemans, M., van der Aalst, W.M.P.: Discovery of frequent episodes in event logs. In: SIMPDA 2014, Revised Selected Papers, pp. 1–31 (2014)
19. Li, G., de Carvalho, R.M., van der Aalst, W.M.P.: Automatic discovery of object-centric behavioral constraint models. In: BIS 2017, vol. 288, pp. 43–58 (2017)
20. Lu, X., Nagelkerke, M., van de Wiel, D., Fahland, D.: Discovering interacting artifacts from ERP systems. IEEE Trans. Serv. Comput. **8**(6), 861–873 (2015)
21. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data Min. Knowl. Discov. **1**(3), 259–289 (1997)
22. Peeva, V., Mannel, L.L., van der Aalst, W.M.P.: From place nets to local process models. In: PETRI NETS (2022)
23. Tax, N., Sidorova, N., Haakma, R., van der Aalst, W.M.P.: Mining local process models. J. Innov. Digit. Ecosyst. **3**(2), 183–196 (2016)