Discovering Object-Centric Process Simulation Models

Benedikt Knopp
Chair of Process and Data Science
RWTH Aachen University
Aachen, Germany
knopp@pads.rwth-aachen.de

Mahsa Pourbafrani
Chair of Process and Data Science
RWTH Aachen University
Aachen, Germany
mahsa.bafrani@pads.rwth-aachen.de

Wil M. P. van der Aalst
Chair of Process and Data Science
RWTH Aachen University
Aachen, Germany
wvdaalst@pads.rwth-aachen.de

***Abstract*—Process simulation assesses the impact of changing environmental parameters on a process. To obtain realistic simulation models, process mining techniques can be deployed for a log-based discovery. Such discovery techniques usually rely on a fixed case notion, falling short in capturing the entangled nature of real organizational processes as an interplay of objects and subprocesses. Yet there is a need for such methods, given the requirement for information systems to foresee and adapt to changing environments in an online setting and in a holistic manner. In this work, we approach this research need by elaborating a method for simulation model discovery that is based on the object-centricity paradigm. To implement object-centric simulation, some intrinsic challenges have to be overcome. These include, first, the parametrizable generation of sets of objects having predefined interrelations that structure possible behavior. Second, the generated objects have to be synchronized and routed through a control-flow model. We outline these challenges, describe our solution approach, and evaluate the quality of both object generation and behavior.**

***Index Terms*—Process Mining, Process Simulation, Object-Centricity, Digital Twins.**

I. Introduction

Process simulation is a technique to assess the impact of changing environmental parameters on the course of a process. Process mining provides data-driven techniques for simulation [1], to discover or improve simulation models based on log evidence. Such techniques are well matured for classical log formats [2], [3]. However, these works have a representational bias by using a fixed case notion to describe processes, neglecting the entangled nature of business processes as an interplay of objects of various types, such as resources or goods.

While notions and techniques such as Colored Petri nets (CPNs) can be tailored toward simulation [4] and are in fact able to capture object-centricity, there are, to the best of our knowledge, no methods for automated discovery of object-centric simulation models. A reason could be that CPNs are very expressive, and lacking of a lightweight interface to facilitate automated discovery techniques. The object-centric paradigm in process mining [5] provides such an interface, as well as log [6] and control-flow standards [7] that could work as enablers.

Funded under the Excellence Strategy of the Federal Government and the Länder. Also, we thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

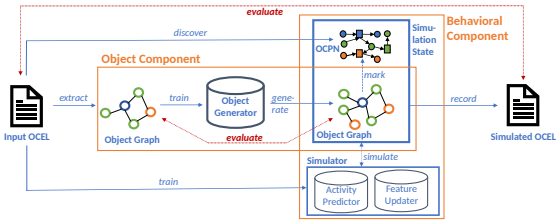

Fig. 1: Our approach for object-centric process simulation. An *object component* generates objects, marking a control-flow driven by a *behavioral component*.

We identify a need for simulation model discovery techniques that respect the object-centric nature of processes based on the following reasons. Firstly, there is a requirement for information systems and business processes to be able to foresee and adapt to changing environments [8]. Secondly, there is a trend for information systems to span holistically across organizations, forming an organizational Digital Twin (DT) [9]. As argued above, object-centricity is a suitable paradigm for such holistic models, while the classical case notion is insufficient. Thirdly, emergent DT technologies aim to automatically recommend or even implement changes in an organization's environment. For this, such models need simulation capabilities [8], [10]. In the online nature of DTs, we also find a motivation for automated, data-driven simulation techniques.

In this work, we present *Object-Centric Process Simulation Models (OCPS)*. Our solution (Fig. 1) relies on an *object* component with an object generator and an intertwined *behavioral* component that routes generated objects through a control-flow model. Both components are derived from one input log in a semi-automated way. As an interface to pose what-if questions, we provide the possibility to parameterize the patterns of object interrelations in the object generator. The quality of generated objects, generated behavior and performance aspects are

then compared against corresponding observations in the input log.

We proceed as follows. In Sec. II, we introduce background for object-centric logs and models. Sec. III describes how the two components of our solution are structured (Sec. III-A) and discovered (Sec. III-B). In Sec. IV, we evaluate simulation runs on two data sets concerning the quality of generated objects, behavior and performance. In Sec. V, we discuss work related to data-driven simulation model construction and object-centric forward-looking techniques. We conclude in Sec. VI.

II. Preliminaries

We make use of set and multiset notations as follows. Let X be a set. $\mathcal{P}(X)$ is the power set of X. A multiset over X is a function $Y : X \to \mathbb{N}$ that assigns a frequency to each $x \in X$. With $\mathcal{B}(X)$, we denote the set of all multisets over X. We make use of the alternative notation $[x_1^{Y(x_1)}, ...]$ for a multiset Y, for example $[x_1^3, x_2]$ is a multiset Y over $X = \{x_1, x_2\}$ with $Y(x_1) = 3, Y(x_2) = 1$.

As indicated in Fig. 1, we rely on an input log based on which we derive a simulation model centered around a control-flow model. In the following, we describe both the log and control-flow model. First, we introduce the set of universes which logs, control-flow as well as the later introduced framework components base upon.

Definition 1 (Universes) We make use of the following universes and functions.

- $\mathbb{U}_{ei}$ are event identifiers,
- $\mathbb{U}_{act}$ are activity names,
- $\mathbb{U}_{time}$ are totally ordered timestamps,
- $\mathbb{U}_{ot}$ are object types,
- $\mathbb{U}_{oi}$ are object identifiers,
- $type \in \mathbb{U}_{oi} \to \mathbb{U}_{ot}$ assigns an object type to each object identifier,
- $\mathbb{U}_{event} = \mathbb{U}_{ei} \times \mathbb{U}_{act} \times \mathbb{U}_{time} \times (\mathcal{P}(\mathbb{U}_{oi}) \setminus \{\emptyset\})$ are events.
- $\mathbb{U}_F$ with $time \in \mathbb{U}_F$ are features,
- $\mathbb{U}_{Fval}$ with $\mathbb{U}_{time} \subseteq \mathbb{U}_{Fval}$ are feature values,
- $\mathbb{U}_{asg} = \{\gamma \in \mathbb{U}_F \nrightarrow \mathbb{U}_{Fval} \mid time \in dom(\gamma)\}$ are feature value assignments.

Definition 2 (Event Log [6]) An *object-centric event log* (OCEL), or short *log*, is a set of events $L \subset \mathbb{U}_{event}$ such that for all $e_1 = (ei_1, a_1, tm_1, b_1), e_2 = (ei_2, a_2, tm_2, b_2) \in L : ei_1 = ei_2 \implies e_1 = e_2$. $\mathbb{U}_{OCEL}$ is the universe of OCELs.

Tab. I depicts an exemplary log L_0 as a collection of events ordered by their *time*, each described by an *activity* and a set of *objects* of types such as *orders*.

The behavior recorded in an OCEL can be modeled by an *object-centric Petri net* (OCPN) [7].

Definition 3 (Object-Centric Petri Net) An *object-centric Petri net* (OCPN) is a tuple $ON = (N, pt, F_{var})$ where $N = (P, T, F, l)$ is a labeled Petri net where $l \in T \to \mathbb{U}_{act}$ is a labeling function, $pt \in P \to \mathbb{U}_{ot}$ maps places onto object types, and $F_{var} \subseteq F$ is the set of variable arcs. Furthermore, for each $t \in T$,

- $tpl(t) = \{pt(p) \mid p \in \bullet t \cup t\bullet\}$ are the object types,
- $tpl_v(t) = \{pt(p) \mid p \in \bullet t \cup t \bullet \wedge \big(\{(p,t),(t,p)\} \cap F_{var}\big) \neq \emptyset\}$ are the variable object types,
- $tpl_{nv}(t) = \{pt(p) \mid p \in \bullet t \cup t \bullet \wedge \big(\{(p,t),(t,p)\} \cap F \setminus F_{var}\big) \neq \emptyset\}$ are the non-variable object types at t.

ON is called *well-formed* if for each $t \in T$, $tpl_{nv}(t) \cap tpl_v(t) = \emptyset$. With $\mathbb{U}_{OCPN}$, we denote the universe of OCPNs. In the following, we assume each OCPN to be well-formed.

A *marked* OCPN has a distribution of tokens across its places. Tokens in an OCPN carry the identifier of a typed object, and reside only in accordingly typed places.

Definition 4 (Marking) Let $ON = (N, pt, F_{var}) \in \mathbb{U}_{OCPN}$ with $N = (P, T, F, l)$. $Q_{ON} = \{(p, o) \in P \times \mathbb{U}_{oi} \mid type(o) = pt(p)\}$ is the set of possible tokens. $M \in \mathcal{B}(Q_{ON})$ is called a marking. A *marked net* is a tuple (ON, M).

OCPNs can be discovered in a perfectly fitting manner based on any given OCEL [7]. Fig. 2 (left) depicts an exemplary marked net (ON_0, M_0), mined based on L_0, with $M_0 = [(x_2, o_1), ...]$. Transitions $t_1, ..., t_4$ carry labels, e.g., $l(t_1) =$ *place order*. In the net, variable arcs are printed in bold.

The execution semantics of OCPNs are similar to those of standard Petri nets except that through variable arcs, arbitrarily many tokens are bound.

Definition 5 (Binding) Let $ON = (N, pt, F_{var})$ be an OCPN with transitions T. $B = \{(t, b) \in T \times (\mathcal{P}(\mathbb{U}_{oi}) \setminus \{\emptyset\}) \mid \cup_{o \in b} type(o) \subseteq tpl(t) \wedge \forall ot \in tpl_{nv}(t) : |\{o \in b \,|\, type(o) = ot\}| = 1\}$ is the set of possible bindings over *ON*. For a binding $(t, b) \in B$, $cons(t, b) = [(p, o) \in Q_{ON} \mid p \in \bullet t \wedge o \in b]$ are the consumed, $prod(t, b) = [(p, o) \in Q_{ON} \mid p \in t \bullet \wedge o \in b]$ are the produced tokens. (t, b) is enabled in marking $M \in \mathcal{B}(Q_{ON})$ if $cons(t, b) \leq M$. The occurrence of an enabled binding (t, b) in M leads to a new marking $M' = M - cons(t, b) + prod(t, b)$.

For example, in Fig. 2 (left), transition t_3 could fire either with the occurrence of the enabled binding $(t_3, \{p_1\})$ or with $(t_3, \{p_1, i_1\})$.

Based on these concepts of object-centric logs and Petri nets, we build our simulation framework.

TABLE I: An exemplary OCEL L_0. Objects of the types *orders, items* and *packages* interfere at various activites.

Event	Activity	Time	orders	items	packages
e_1	place order	10:00	o_1	i_1, i_2	-
e_2	pick item	10:30	o_1	i_1	-
e_3	pick item	10:35	o_1	i_2	-
e_4	ship package	13:30	-	i_1, i_2	p_1
e_5	notify customer	13:30	o_1	-	-

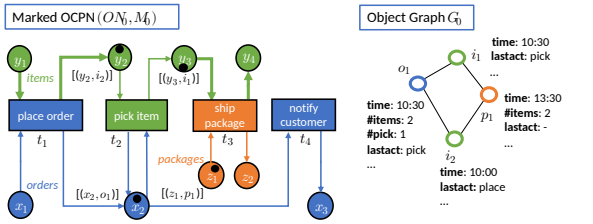

Fig. 2: A marked net (ON_0, M_0) capturing a control-flow state (left). Together with an object graph G_0 (right), a simulation state $s_0 = (M_0, G_0)$ is described.

III. Approach

For the creation of a simulation model, we identify two main components, as outlined in Fig. 1.

1) *Object Component:* object graph and generator,
2) *Behavioral Component:* control-flow model, stochastic models

Consider the exemplary log L_0 in Tab. I, describing an order management process. There are three object types, namely *orders, items* and *packages*, interacting at various activities. Our goal is to derive a simulation model from such event logs (Fig. 1, left) which generates comparable behavior, i.e. , a similar event log (Fig. 1, right). For this, we need an *object component* to create sets of objects that resemble the observed objects and their interplay. The task of the *behavioral component* is to plausibly route related objects through a control-flow model, i.e. , through an enriched object-centric Petri net.

A. Components

In the following, we formalize these two components. After that, we propose an approach to discover the components from an input log.

1) Object Component: In an input log, objects of different types interfere at event horizons. For instance, in L_0, *order* o_1 is associated with *items* i_1, i_2 at event e_1 (*place order*). We make two assumptions in order to structure possible behavior and to facilitate object generation. Firstly, such relations between objects are fixed, for example, o_1 is not placed again with different items. Secondly, these relations constrain the possible bindings in the process course: for example, o_1 again relates to i_1 at e_2 and i_2 at e_3. A possible abstraction of such fixed and recurring relations is an *object graph*.

We introduce object graphs firstly with the goal of capturing these object relations, and secondly to track features at objects.

Definition 6 (Object Graph) An object graph is a tuple $G = (OI, R, \Gamma)$ with nodes $OI \subseteq \mathbb{U}_{oi}$, undirected edges $R \subseteq \{\{o_1, o_2\} \mid o_1, o_2 \in OI, o_1 \neq o_2\}$ and features $\Gamma \in OI \to \mathbb{U}_{asg}$. $\mathbb{U}_{OG}$ is the universe of all object graphs.

An exemplary object graph G_0 is depicted in Fig. 2 (right). G_0 has feature assignments over the following features: for each $a \in \mathbb{U}_{act}$, the number of executions per activity, $\#a$ (e.g., $\#pick$ for *pick item*); for each $ot \in \mathbb{U}_{ot}$, the number of related objects of that type, $\#ot$ (e.g., o_1 is related to 2 *items*); also, the last performed activity is tracked (*lastact*). For the sake of brevity, Fig. 2 shows only some of the assigned values.

To generate new object graphs, we first extract graphs from the input log to use as a training basis for generation. To this end, we translate object interaction patterns observed in the log to object relations. In L_0, there are various such patterns. For instance, at e_1, order o_1 interacts with two items i_1 and i_2, but only with one item each at e_2 and e_3, and with none of them at e_5. Assume that o_1 has an object relationship with i_1 and i_2, as in G_0. For simplification, we allow only the three aforementioned cases: an object either interacts with the whole of its related objects of a certain type, with exactly one, or with none. The structure of an underlying OCPN that complies with this requirement will feature suitable arc multiplicities, as illustrated in Tab. II.

TABLE II: Object co-occurrence patterns and correlation with the net structure.

Event	*items* related to o_1 involved	Arcs for *item* at transition
e_1	i_1, i_2 (all related *items*)	variable arc
e_2	i_1 (single)	non-variable arc
e_3	i_2 (single)	non-variable arc
e_5	- (none)	no arc

We also require that for each activity, there is an object type, called the *leading type*, such that any activity occurrence includes exactly one object of that type, called the *leading object*, around which these assumptions are satisfied.

Definition 7 (Leading Types) Let $L \in \mathbb{U}_{OCEL}$ with activities $A_L = \{a \in \mathbb{U}_{act} \mid \exists\ (ei, a, tm, b) \in L\}$. Furthermore, let $lead \in A_L \to \mathbb{U}_{ot}$. $lead$ is called a *leading type assignment* for L, if for all $(ei, a, tm, b) \in L : |\{o \in b \mid type(o) = lead(a)\}| = 1$. In this case, there is a unique function $lead_{OI} \in L \to \mathbb{U}_{oi}$ where for all $e = (ei, a, tm, b) \in L : lead_{OI}(e) \in b$ and $type(lead_{OI}(e)) = lead(a)$. $lead_{OI}$ is called the *leading object assignment* corresponding to $lead$.

In Fig. 2 (left), the color of each transition reflects a possible leading type for the corresponding log activity. Object relationships then emerge by means of object co-occurrences at events between the leading object and all other involved objects (Fig. 2, right). In the following, we assume that each event log has such a leading type

assignment. Based on that assignment, we extract an object graph from the event log.

Definition 8 (Graph Extraction) Let $L \in \mathbb{U}_{OCEL}$ with objects $OI_L = \{o \in \mathbb{U}_{oi} \mid \exists\, (ei, a, tm, b) \in L : o \in b\}$. Let furthermore $G = (OI, R, \Gamma) \in \mathbb{U}_{OG}$. G is called an *object graph extraction from* L, if $OI = OI_L$ and there is a leading type assignment *lead* for L such that for all $o_1, o_2 \in OI, o_1 \neq o_2 : \{o_1, o_2\} \in R$ iff there is an $e = (ei, a, tm, b) \in L$ with $o_1, o_2 \in b$ and $lead_{OI}(e) \in \{o_1, o_2\}$.

For example, G_0 from Fig. 2 is an object graph extraction from L_0 by the given leading type assignment.

As mentioned, we use such an extracted object graph as a training basis for the generation of new graphs. Before turning to generation and discovery procedures, we complete the description of the desired target simulation models, using the above graph extraction as an illustration for a simulation state. Also, we show how our assumptions on leading types and co-occurence patterns provide a structure to control object behavior.

2) Behavioral Component: A *state* of a simulation run is composed of the marking of the control-flow model (OCPN) and a corresponding graph capturing properties of the objects residing in the net.

Definition 9 (Simulation State) Let $ON \in \mathbb{U}_{OCPN}$, $M \in \mathcal{B}(Q_{ON})$, and $G = (OI, R, \Gamma) \in \mathbb{U}_{OG}$ such that $OI = \{\, o \in \mathbb{U}_{oi} \mid (p, o) \in M \,\}$. $s = (M, G)$ is called a *simulation state* over *ON*. We denote $\mathcal{S}_{ON}$ to be the set of all simulation states over *ON*.

For example, M_0 together with G_0 as depicted in Fig. 2 constitutes the simulation state $s_0 = (M_0, G_0)$, reflecting the state of the process after e_1 and e_2 occurred.

Next, we describe a simulation engine to transform states. For this, we make use of activity predictors to choose bindings, and feature updaters to transform the state of the object graph. As a basis for choosing bindings of multiple objects, we rely on predictions on the level of *single objects*.

Definition 10 (Flat Activity Predictor) Let ON be an OCPN with transitions T. A *flat activity predictor* over ON is $J \in \mathbb{U}_{asg} \to (T \to [0, 1])$, such that for all $\gamma \in \mathbb{U}_{asg}$, $\sum_{t \in T} J(\gamma)(t) \leq 1$.

Example. In the running example, one can choose values for a flat predictor J_0 that reflect the continuation of the simulation state. For instance, we can chose $J_0\big((\Gamma(o_1))\big)(t_2) = 1$, as the next activity for o_1 surely is *pick item*.

In general, we aim at discovering activity predictors that capture the business logic as an interplay of features. In the running example, for *orders*, one can predict *pick item* as long as the number of executed *pick* activities

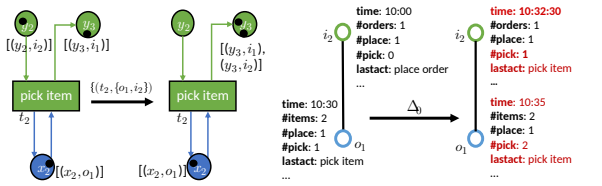

Fig. 3: Transformations on parts of marking (left) and object graph (right) of s_0 by means of a binding execution.

does not equal the number related objects of type *items*. In the discovery section (Sec. III-B), we will describe how a simple stochastic predictor to account for such a business logic is extracted from a log.

Definition 11 (Feature Updater) Let ON be an OCPN with transitions T. A feature updater is a function $\Delta \in (T \times \mathbb{U}_{asg}) \nrightarrow \mathbb{U}_{asg}$.

Example. We define a feature updater Δ_0 that operates as follows: If an object of type ot is bound at a transition t with $l(t) = a$, the counter feature $\#a$ is incremented, and we set *lastact* to a. Also, Δ_0 defines a simple timestamp update at t as the average delay from the last activity *lastact* until a, over all observations on objects of type ot in the input log L_0.

Activity predictor and feature updater together with an OCPN make a simulation model.

Definition 12 (Simulation Model) Let $ON \in \mathbb{U}_{OCPN}$. $OS = (ON, J, \Delta)$ where

- J is a flat activity predictor over *ON*,
- Δ is a feature updater over *ON*,

is called an *Object-Centric Process Simulation Model (OCPS)* over *ON*.

We put a simulation model into effect by repeatedly selecting and executing bindings. Given a state $s = (M, G) \in \mathcal{S}_{ON}$ with $G = (OI, R, \Gamma)$, J provides predictions for all $t \in T$ and single objects $o \in OI$ based on Γ. For each $t \in T$ and $O \subseteq OI$, flat prediction values can be aggregated to a *joint* prediction value $\mathcal{J}_{(t,O)}$ by taking the minimum flat prediction value.

Candidate bindings (t, O) are identified as follows. Firstly, (t, O) should be enabled in *(ON, M)*. Secondly, according to our observations on object relations in Tab. II, there should be exactly one leading object $o \in O$ with $type(o) = lead(l(t))$, such that the requirements given in Tab. III are satisfied for all $ot \in \mathbb{U}_{ot} \setminus \{type(o)\}$, depending on the arc type of ot at t.

TABLE III: We restrict the set of candidate bindings in an execution step.

Arcs for ot at t	Bound objects of type ot in O
variable	all neighbors of o of type ot
non-variable	one neighbor of o of type ot
none	none

A binding to be executed is selected based on the following heuristics. (1) All candidate bindings (t, O)

according to above restrictions are identified. (2) The bindings are iterated ascending by the time at which they would be executed. Here, the maximum timestamp of any object $o \in O$ determines the execution time. (3) A binding is picked for execution with the joint probability $\mathcal{J}_{(t,O)}$. Thus, the first such binding with a positive outcome for execution selection terminates the selection procedure.

Executing the selected binding results in a new marking M'. The object graph $G' = (OI, R, \Gamma')$ of the new state is given by the feature updater Δ_0. Fig. 3 illustrates this simulation runtime by the realization of the binding $(t_2, \{o_1, i_2\})$ in the state s_0.

B. Discovery

In the following, we describe an approach for discovering the object component and the behavioral component defined above. The approach described in this section will yield, starting from an input log L, a simulation model $OS = (ON, J, \Delta)$ over an OCPN *ON* as well as an initial simulation state $s = (M, G)$.

For both components, we make use of *flat logs per object type*. Such a flat log is derived from the input OCEL by collecting all traces of individual objects of that type [5]. As depicted in Tab. IV for type *orders* in the running example, these logs are then enriched with object features. As a design choice, we use counters for activity executions and track the last activity, as described in Sec. III-A1. Also, we fix a leading type assignment on the input log and use the resulting number of object relations per type as features, e.g. $\#items$. These features are added at case level (object relations counter) or event level (activity counter, last activity) to the flattened logs. Also, for the simulation of timing behavior, we compute the *delay* in between events.

1) Object Component: For the object component, we now derive requirements for an object graph generator and then describe a generation procedure.

Object Graph Extraction. In the initial state of a simulation, there should be an object graph that under default parametrization is structurally comparable to the object graph extracted from the input log. For this extraction $G' = (OI, R, \Gamma)$, we derive *OI, R* by Def. 8. For the feature assignment Γ, we extract, for every object, features from its initial event in the enriched flattened log of the corresponding type. For instance, o_1 in the running example initially has $\Gamma(o_1)(\#items) = 2, \Gamma(o_1)(time) = 10{:}00$ etc. (cf. Tab. IV).

Object Relationship Multiplicity Distributions. In the extracted object graph, objects of different types relate to each other with different multiplicities, defining *object relationship multiplicity distributions (ORMDs)*. For example, assume that some log (Fig. 4, left) has objects of type *orders* relating to either $2, 3$ or 4 *items*, each

Fig. 4: By means of the leading type assignment, an OCEL (left) exhibits patterns of object relations (middle left). Based on the original or parametrized ORMDs, an object generator (middle right) can generate new sets of objects (right).

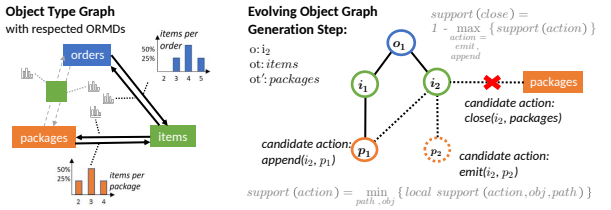

Fig. 5: The generation procedure is guided by ORMDs (left). At each generation step (right), new relations emerging from an action affect the ORMDs in the evolving graph. Assessing the likelihood of that effect w.r.t. the given ORMD defines the *support* of an action.

with a certain probability. These probabilities define the ORMD *items per order*. ORMDs also emerge transitively for greater *depths*, for instance, by counting the *packages per items per order* at a depth of 2. An ORMD is then, based on domain knowledge about the process, selected or not selected to be respected during generation. Also, to pose what-if scenarios, one may adapt these distributions. The selected ORMDs then shape the evolving object graph, as described in the following.

Object Graph Generation. To implement an object generator that satisfies above requirements we proceed as follows, by iteratively generating connected components of objects as illustrated in Fig. 5. The procedure starts with creating a *seed* object of an arbitrary type. In each step, a random combination of an existing object o of some type ot and an object type ot' is picked. Then, either (1) o is *appended* (i.e., a relation emerges) to an existing object of type ot'; (2) a new object of type ot' *emits* and is connected to o; or (3) o is *closed* towards ot', meaning that no more relations will emerge between o and any object of type ot'. Each candidate action has a *support* value which is used as a weight for sampling.

The *support* values are based on the ORMDs passed as parameters. We illustrate the used distributions at an *object type graph* (Fig. 5, left) that connects types between which object relationships exist. Fig. 5 (right) shows an exemplary intermediary generation step where an *item* i_2 is to be connected or not connected to a *package*. Choosing a generative action, for example *appending* i_2 to the existing package p_1, will create new *paths* in the object graph, for instance, the paths (p_1, i_2) and (o_1, i_2, p_1). Each path may affect the respective object relationship multiplicities in the evolving graph,

TABLE IV: L_0 flattened on *orders*, yielding flat log L_0^{orders} (1st - 3rd column), enriched with features (from 4th column).

orders	activity	time	delay	#items	#place	#pick	#notify	lastact
o_1	place order	10:00	0:00	2	0	0	0	-
o_1	pick item	10:30	0:30	2	1	0	0	place
o_1	pick item	10:35	0:05	2	1	1	0	pick
o_1	notify customer	13:30	2:55	2	1	2	0	pick

for instance, here the *items per package* at p_1 are incremented from 1 to 2. The parameter ORMD *items per package* then defines a natural *local support* for the appendix action at p_1 via the path (p_1, i_2) as the conditional probability of a package that has *at least 1* item to have *at least 2* items. The total support of the action is computed as the minimal local support at any respected ORMD that is affected by the action.

Feature Assignment. As object features Γ, we set the *counters* of related objects of each type (cf. Sec. III-B1). Also, *arrival times* are assigned in an iterative manner. For this, we pick as a *seed type* the same object type used as a seed type for the generation procedure. For that type, absolute arrival rates from the input data and fitted against an exponential distribution. Also, *relative arrival times* between related object types are learned. Then, for some object of the seed type, an arrival time is sampled. The object graph is traversed breadth-first and new objects are assigned an arrival time relative to the type of the node on the prior depth level. If another seed type object is encountered, it is assigned an arrival time following the absolute rate of the seed type.

2) Behavioral Component: As a result of above generation, we obtain an object graph G, which is to be put into action by a simulation model. Then, the simulation model components (Def. 12) are discovered as follows.

Control-Flow. We use the inductive miner for the discovery of Petri nets which are then merged to an OCPN [7]. Note that discovering an OCPN this way may violate our assumption of having only labeled transitions. We drop this assumption as follows: next activities are predicted as described in Sec. III-A2. Then, the feasibility of predicted bindings at labeled transitions is assessed modulo reachability of the labeled transition via paths of silent transitions.

As a result of this step, we obtain an OCPN *ON* which is marked with the objects found in G, yielding an initial marking M and an initial simulation state $s = (M, G)$.

Activity Predictor. The stochastic models are based on flattened event logs enriched with our design-choice features, as in Tab. IV. Concerning activity prediction, the *flat predictor* J is a log frequency-based classifier with the target variable *activity*. That is, for a feature assignment γ, the probability $J(\gamma)(t)$ is the relative frequency of events in the enriched log that have both target feature $l(t)$ and the enrichment γ, modulo timing information. If there is no event in that log having γ, we consider the most proximate assignments γ' with respect to numerical features that are supported in the log.

Feature Updater. As described in Sec. III-A2, the feature updater Δ increments activity counters and tracks the last performed activity. Also, timestamps are updated based on the described activity-to-activity delay function.

Activity predictor and feature updater together with the OCPN constitute our simulation model $OS = (ON, J, \Delta)$. This model is then iteratively applied to the initial state s as described in Sec. III-A2.

Logging. Running the simulation, we build an event log by mapping each executed binding naturally to an event. To obtain the timestamp of an event occurrence, we average the timestamps of all bound objects after applying the delay function.

IV. Evaluation

A tool that implements the approach, realizing all indicated parametrization points, can be found on GitHub[1]. Using the tool, we evaluated simulation runs[2] based on two datasets: an artificial log of an order management process[3] (OM) as well as an artificial procure-to-pay process[4] (P2P). The OM log was projected on the object types *orders (ord), items (itm)* and *packages (pck)*, filtering out *customers* and *products* because they are less interesting from a behavioral perspective. Also, *orders* were filtered from all events involving a *package*. The P2P log was projected to two activities and three object types *purchase order (ord), material (mat)* and *goods receipt (gsr)*. We evaluated with regards to three aspects: quality of generated objects, behavioral conformance and timing behavior. OM was considered for all three aspects. P2P was merely used for a study on object quality.

A. Simulation Runs

Tab. V shows the number of objects per object types in the input, namely in the object graph extractions G_0, as well as in various generated graphs. For both logs, object graphs G_1 were generated fitting all ORMDs of depth 1. The graphs G_2 were generated respecting some distributions of depth 2. For OM, G_2^o respects *orders per items per package* and vice versa *packages per items per order*. For P2P, G_2^p respects *purchase orders per materials per*

[1] https://github.com/beneknopp/OCPS
[2] https://github.com/beneknopp/OCPS/tree/main/evaluation
[3] https://ocel-standard.org/
[4] https://github.com/ocpm/ocpa

TABLE V: For both datasets: the number of generated objects for each type.

(a) Number of objects per type for OM.

otype	G^o_0	G^o_1	G^o_2	G^o_+
ord	*2000*	2002	2001	2003
itm	*8159*	8765	8745	11163
pck	*1325*	1324	1281	1668

(b) Number of objects per type for P2P.

otype	G^p_0	G^p_1	G^p_2	G^p_{full}
ord	*80*	84	80	80
mat	*414*	450	497	410
gsr	*80*	82	80	80

Fig. 6: Different ORMDs based on different object type graphs were respected for the generation of G^p_1, G^p_2, G^p_{full} (from left to right), having a great impact on the graph quality.

goods receipt and vice versa. Thirdly, for OM, a graph G^o_+ was generated under adapted parametrization by fitting the *items per order* against a normal distribution and then increasing the average items per order by 1. For P2P, G^p_{full} was generated also to depth 1, but with a different configuration concerning object type per activity filtering and leading type assignment, as depicted below in Tab. VI (leading types printed in italics).

TABLE VI: Input configuration for runs of the P2P log.

Activity	G^p_1, G^p_2	G^p_{full}
Create Purchase Order	*ord*, mat	*ord*, mat
Issue Goods Receipt	*gsr*, mat	*ord*, mat, gsr

B. Graph-Structural Conformance

We use a variant of the *earth-movers distance (EMD)* for histogram comparison [11], to assess the similarity of ORMDs. Tab. VIIa shows average EMD values between the ORMDs of depths $d = 1, 2, 3$ found in the simulated object graphs against those found in the extraction from the input log. For OM, we did not evaluate G^o_+, because deviations between simulated and original distributions are intended. At G^o_1, G^o_2, one can see that respecting ORMDs of greater depth (2 instead of 1) balances out discrepancies between the different depths.

For P2P, the mere numbers of generated objects as shown in Tab. Vb suggest a decent level of similarity to the input objects. However, this is only a superficial similarity, as the EMD values in Tab. VIIa show, especially for G^p_1. The reason lies in disrespecting a transitive 1-to-1 relationship (Fig. 6, left). For G^p_2, the corresponding ORMDs of depth 2 are respected during generation (middle), leading to a significantly better score, but also introducing a trade-off with the satisfaction of direct relationships towards *material*. By a suitable choice of leading type assignment and included object types at

TABLE VII: Evaluation details concerning object graph quality and quality of simulated behavior.

(a) EMD values of ORMDs in the generated object graphs against those in the original object graph extraction, averaged over all paths between object types for various depths d.

	OM		P2P		
d	G^o_1	G^o_2	G^p_1	G^p_2	G^p_{full}
1	0.265	**0.265**	0.128	0.222	**0.013**
2	0.296	**0.234**	0.901	0.187	**0.015**
3	1.308	**0.659**	4.283	0.222	**0.013**

(b) EMD values of flat output logs against flat input logs for all object types.

	OM			
otype	G^o_0	G^o_1	G^o_2	G^o_+
ord	0.180	**0.165**	**0.165**	0.265
itm	0.204	0.139	**0.137**	0.140
pck	0.009	0.005	**0.003**	0.003

(c) Cycle times mean / standard deviation for all object types, in days.

	OM			
otype	L	G^o_0	G^o_1	G^o_+
ord	16.9 | 12.7	11.1 | 6.8	13.9 | 9.1	16.6 | 10.2
itm	15.2 | 12.7	12.2 | 7.2	14.7 | 9.3	16.5 | 10.7
pck	1.81 | 1.47	2.14 | 0.75	1.79 | 0.60	1.77 | 0.58

activities, the transitivity of the 1-to-1 relationship can be turned into a local relationship (right). The resulting graph G^p_{full} has the highest conformance.

C. Behavioral Conformance

We use a variant of the earth-movers distance (EMD) to compare trace variant distributions [12]. Tab. VIIb shows, for the OM log, such EMD values between the simulated logs against the input logs. The scores are comparable between the run using the original graph and the runs using the three generated graphs. Especially *packages,* on the one hand, show a behavior that is highly similar to the original process. This is because that type exhibits a rather sequential control-flow with few trace variants. Note that also the parametrized run based on G^o_+ shows similarly high conformance for *items* and *packages*. This is because, while the number of *items per order* was increased indeed, this has little effect on the behavior of each individual object of those types.

D. Timing Behavior

Tab. VIIc shows, for the OM log, mean and standard deviation of cycle times over all objects of a type, for some simulated runs and in the input log L^o. As expected, *orders* in G^o_+ have comparatively long cycles due to more related *items* and thus more events per object. In general, simulated cycle times tend to be shorter than in the original process, while showing a lower standard deviation. Note that our approach does not respect timing and resource aspects such as congestion [13] and availability or arrival calendars [14]. In the original OM process underlying the simulation, such calendars are used. However, we see an adaption of refined timing and resource modeling as future work.

In conclusion, our implementation succeeds in adequately simulating observed behavior, especially with re-

gards to trace variants, and is also capable of generating realistic sets of objects, given a suitable configuration of the input log.

V. Related Work

This paper is concerned with the data-driven process simulation model construction in an object-centric setting. General challenges in data-driven construction methods are outlined in [15]. Many of these challenges, such as resource scheduling, cannot be addressed in this paper due to its proof-of-concept nature, but they apply readily to our case. In [16], a comprehensive set of metrics is provided to assess simulation model quality, covering quality aspects we discussed as well as the time and resource aspects we plan to address.

Simulation models can either be designed in a top-bottom manner by manual configuration, or in a bottom-up manner with the help of mining techniques. As a contribution to the latter, this paper takes up work initiated in [17], where a CPN is discovered from event data in a fully automated way. More recently, a tool was proposed [2] that covers many challenges beyond control-flow discovery, such as distribution fitting and automated accuracy optimization. Furthermore, the authors deploy advanced resource availability modeling [14]. Another work [3] provides a discovery tool with the additional option of adapting the discovered process models. Future work may take up these methods to further advance object-centric simulation.

To the best of our knowledge, this is the first work on fully automated, data-driven discovery as well as execution of object-centric process simulation models. However, there are initial works on predictive process analytics [18], [19] and feature engineering [20] for that setting. For their predictive models, [19] also choose as features the notion of relations between object types. [18] defines transitive object relations over several succeeding events. In our work, we do not only make predictions for single objects, but synchronize objects of multiple types.

VI. Conclusion and Future Work

This paper describes a strategy for implementing object-centric process simulation. We proposed a generic approach for a log-based discovery of an object generator and a process simulation model with simple log frequency-based statistical models to simulate behavior for the generated objects.

By using log-to-log and graph-to-graph conformance measures, we showed the quality of our approach and identified recommendable strategies for the configuration of the input to improve output quality. As the next step, we aim at improving the quality of our framework by advancing the deployed statistical models, timing and resource modeling and object generation procedures, and also with the help of case studies.

References

[1] W. M. P. van der Aalst. *Process mining: Data science in action*. Springer Berlin Heidelberg, 2016.

[2] M. Camargo, M. Dumas, and O. González-Rojas. Automated discovery of business process simulation models from event logs. *Decision Support Systems*, 134:113284, 2020.

[3] M. Pourbafrani, S. Jiao, and W. M. P. van der Aalst. Simpt: process improvement using interactive simulation of time-aware process trees. In *International Conference on Research Challenges in Information Science*, pages 588–594. Springer, 2021.

[4] K. Jensen and L. M. Kristensen. Colored petri nets: a graphical language for formal modeling and validation of concurrent systems. *Communications of the ACM*, 58(6):61–70, 2015.

[5] W. M. P. van der Aalst. Object-centric process mining: Dealing with divergence and convergence in event data. In *Software Engineering and Formal Methods: 17th International Conference, SEFM 2019, Oslo, Norway, September 18–20, 2019, Proceedings 17*, pages 3–25. Springer, 2019.

[6] A. F. Ghahfarokhi, G. Park, A. Berti, and W. M. P. van der Aalst. Ocel: A standard for object-centric event logs. In *European Conference on Advances in Databases and Information Systems*, pages 169–175. Springer, 2021.

[7] W. M. P. van der Aalst and A. Berti. Discovering object-centric petri nets. *Fundamenta informaticae*, 175(1-4):1–40, 2020.

[8] I. Beerepoot et al. The biggest business process management problems to solve before we die. *Computers in Industry*, 146:103837, 2023.

[9] R. Parmar, A. Leiponen, and L. D. Thomas. Building an organizational digital twin. *Business Horizons*, 63(6):725–736, 2020.

[10] M. Dumas. Constructing digital twins for accurate and reliable what-if business process analysis. In *Problems@ BPM*, pages 23–27, 2021.

[11] O. Pele and M. Werman. Fast and robust earth mover's distances. In *2009 IEEE 12th International Conference on Computer Vision*, pages 460–467. IEEE, September 2009.

[12] S. J. J. Leemans, A. F. Syring, and W. M. P. van der Aalst. Earth movers' stochastic conformance checking. In *Business Process Management Forum: BPM Forum 2019*, pages 127–143. Springer, 2019.

[13] A. Senderovich, J. C. Beck, A. Gal, and M. Weidlich. Congestion graphs for automated time predictions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4854–4861, 2019.

[14] B. Estrada-Torres, M. Camargo, M. Dumas, L. García-Bañuelos, I. Mahdy, and M. Yerokhin. Discovering business process simulation models in the presence of multitasking and availability constraints. *Data Knowl. Eng.*, 134:101897, 2021.

[15] N. Martin, B. Depaire, and A. Caris. The use of process mining in business process simulation model construction: structuring the field. *Business & Information Systems Engineering*, 58:73–87, 2016.

[16] D. Chapela-Campa, I. Benchekroun, O. Baron, M. Dumas, D. Krass, and A. Senderovich. Can I trust my simulation model? measuring the quality of business process simulation models. In *Business Process Management - 21st International Conference, BPM 2023*, pages 20–37. Springer, 2023.

[17] A. Rozinat, R. S. Mans, M. Song, and W. M. P. van der Aalst. Discovering simulation models. *Information systems*, 34(3):305–327, 2009.

[18] R. Galanti, M. De Leoni, N. Navarin, and A. Marazzi. Object-centric process predictive analytics. *Expert Systems with Applications*, 213:119173, 2023.

[19] W. Gherissi, J. El Haddad, and D. Grigori. Object-centric predictive process monitoring. In *International Conference on Service-Oriented Computing*, pages 27–39. Springer, 2022.

[20] J. N. Adams, G. Park, S. Levich, D. Schuster, and W. M. P. van der Aalst. A framework for extracting and encoding features from object-centric event data. In *International Conference on Service-Oriented Computing*, pages 36–53. Springer, 2022.