Object Synchronizations and Specializations with Silent Objects in Object-Centric Petri Nets

Jan Niklas van Detten[1,2] ✉, Pol Schumacher[2], and Sander J.J. Leemans[1,3]

[1] RWTH University, Aachen Germany
[2] Celonis, Munich Germany
[3] Fraunhofer FIT, Aachen Germany

Abstract. Processes involve interacting objects of different types, such as orders, items and machines. Object-centric event logs capture the execution of activities with the involved objects in such processes. Discovery algorithms use these logs to construct process models in standardized formalisms, such as object-centric Petri nets. Conformance checking techniques quantify how fittingly and precisely these models represent a log. Existing techniques for object-centric Petri nets only consider explicitly recorded objects. However, unobserved objects might still influence the behaviour of the observed objects in a log. Observed objects may belong to the same unobserved group and therefore remain synchronized across multiple activities. Additionally, observed objects might have an unobserved property and hence specialize on certain activities. In this paper, we propose the notion of silent objects in object-centric Petri nets, which are unobserved objects that nonetheless affect observed objects. We provide a novel algorithm to automatically detect silent objects that cause synchronizations and specializations in object-centric event logs. We show that the inclusion of silent objects in the discovery of object-centric Petri nets preserves fitness and increases precision for logs that include them. We apply our method to multiple public logs and consistently find silent objects to be present in them. Additionally, we observe our approach's runtime to remain feasible for large input logs.

Keywords: process mining · object-centric modeling · process discovery

1 Introduction

The research area of process mining provides techniques to analyze and optimize business processes. In many cases the structure of a process first needs to be specified in a process model [5]. Such models are compact representations of processes in standardized formalisms, such as BPMN diagrams or Petri nets.

Ideally, a process model should allow exactly the behaviour of the process that it represents. However, it might not be possible to construct such an ideal model for any given process, since the chosen modelling formalism induces a representational bias through its modeling constructs. The insufficient quality of a model may impede the subsequent application of analysis techniques. Therefore, it is important to be aware of relevant behaviour that cannot be modeled.

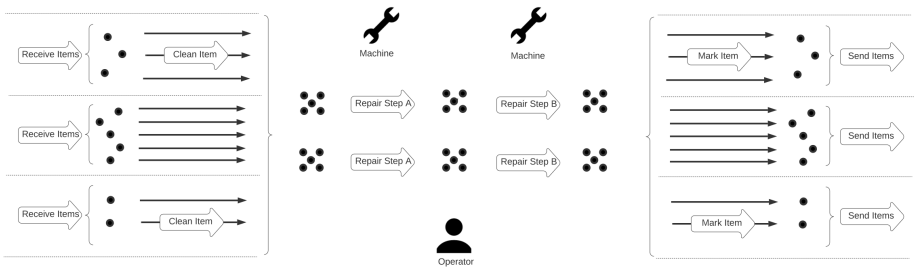

Fig. 1. Informal description of an example repair process.

Figure 1 depicts a simple repair process. Groups of broken items are received, and then repaired by an employee using specialized machines. Afterwards, each group of repaired items is returned. Items that are received in too bad conditions are cleaned before the repair and marked afterwards. This process already exhibits behaviour that cannot be specified properly in some existing formalisms.

In *object-centric* process models, all entities that participate in a process are considered *objects* of a certain type [1]. In our example repair process above, the objects of the type items, machines and employees can be seen. Object-centric modelling formalisms, such as object-centric Petri nets, can be used to describe the control flow for each type and the interactions between them.

However, some objects that participate in our process are not explicitly specified. We can observe that for each received set of items, the same items are also returned together. The actual repair steps are performed in groups of items that are independent from the deliveries. Both of these *object synchronization*s correspond to conceptual group objects, like, in our example, an item delivery and a repair batch that synchronize a set of items for different parts of the process. Furthermore, only some items, based on their condition, are cleaned at the beginning and marked at the end. We refer to objects of the same type being involved in different activities due to their unobserved properties as an *object specialization*. Based on the source and goal of the model, object synchronizations and specializations may or may not be included as objects in a model explicitly.

The application of most process mining techniques is based on execution track records, called event logs. Object-centric event logs can be extracted from ERP systems of a process [7]. Such logs capture the execution of process activities in relation to the involved objects. Object synchronizations and specializations may not be included in these logs, for instance due to the ERP extraction strategy, incomplete recordings and data complexity considerations. Missing group objects and properties, such as those discussed for our example, cause under-specification of automatically discovered object-centric Petri nets. Therefore, we propose to extend object-centric Petri nets and their discovery with objects that are not explicitly present in the log, but emerge as patterns, to increase precision.

For this purpose, we introduce *silent object*s in object-centric Petri nets, which, in a model, describe unobserved objects that nevertheless may affect

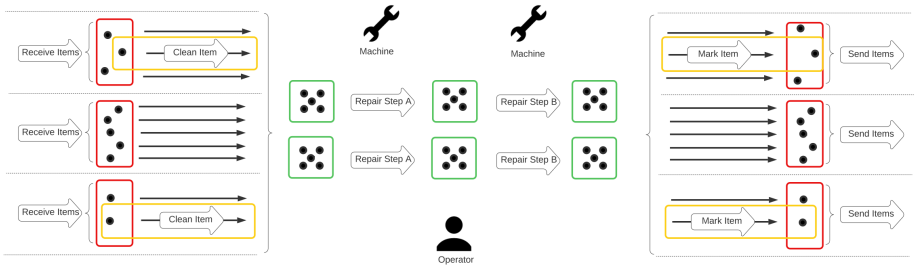

Fig. 2. Silent objects in our repair process.

the behaviour of other, observed, objects. For the scope of this paper we focus on silent objects that synchronize or specialize observed objects in a log. Figure 2 shows some silent objects for our example repair process. The object synchronizations of items that are deliveries are marked in red, while the object synchronizations of items for repair batches are marked in green. The object specialization of the items that are in bad conditions are marked in yellow. Including these silent objects in a model enforces the synchronization and specialization of the observed objects, addressing the previously discussed under-specification.

In this paper, we define silent objects and integrate them into object-centric Petri nets. We introduce an algorithm to automatically detect two categories of silent objects in object-centric event logs and to construct an object-centric Petri net that includes them. We discuss how conformance checking techniques account for these objects and show that the inclusion of these objects, if they are present in a log, improves the precision of a model without impeding fitness. We apply our approach to a range of public logs and find silent objects to be present in them. Our approach shows feasible runtimes, even on large input logs.

2 Preliminaries

In this section, we summarize required background information on event logs, modeling formalisms and conformance checking techniques. We utilize sets $\{\ldots\}$, multisets $[\ldots]$, powersets $\mathbb{P}(\{\ldots\})$, crossproducts $\{\ldots\} \times \{\ldots\}$ and sequences $\langle\ldots\rangle$. We write $\langle\ldots\rangle[i]$ for the element at position i, starting at one, and $|\langle\ldots\rangle|$ for the total length. We always include zero in the set of natural numbers $\mathbb{N}$.

Table 1. Object-centric event log of an repair process.

$$
\begin{array}{lll}
\langle \text{RECEIVE } \{\mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3\}, & \text{CLEAN } \{\mathtt{i}_2, \mathtt{o}_1\}, & \text{RECEIVE } \{\mathtt{i}_4, \ldots, \mathtt{i}_8\}, \\
\quad \text{STEP A } \{\mathtt{i}_1, \ldots, \mathtt{i}_5, \mathtt{o}_1, \mathtt{m}_1\}, & \text{RECEIVE } \{\mathtt{i}_9, \mathtt{i}_{10}\}, & \quad \text{CLEAN } \{\mathtt{i}_{10}, \mathtt{o}_1\}, \\
\quad \text{STEP B } \{\mathtt{i}_1, \ldots, \mathtt{i}_5, \mathtt{o}_1, \mathtt{m}_2\}, & \quad \text{MARK } \{\mathtt{i}_2, \mathtt{o}_1\}, & \quad \text{SEND } \{\mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3\}, \\
\quad \text{STEP A } \{\mathtt{i}_6, \ldots, \mathtt{i}_{10}, \mathtt{o}_1, \mathtt{m}_1\}, & \text{STEP B } \{\mathtt{i}_6, \ldots, \mathtt{i}_{10}, \mathtt{o}_1, \mathtt{m}_2\}, & \quad \text{SEND } \{\mathtt{i}_4, \ldots, \mathtt{i}_8\}, \\
\quad \text{MARK } \{\mathtt{i}_{10}, \mathtt{o}_1\}, & \quad \text{SEND } \{\mathtt{i}_9, \mathtt{i}_{10}\}\rangle &
\end{array}
$$

2.1 Event Logs

Traditionally, event logs are considered multisets of case traces, which are sequences of activity labels. We write Σ for all activity labels in an event log.

Object-centric event logs drop case notions and capture the involvement of objects. They are defined as a sequence of events, which are an activity label and an object set $L = \langle (a_1, O_1), \dots (a_n, O_n) \rangle$. We write Θ for all objects in a log and Ω for all object types. Table 1 shows an example for our repair process with machines $(\mathtt{m}_1, \mathtt{m}_2)$, operators $(\mathtt{o}_1)$ and items $(\mathtt{i}_1, \dots, \mathtt{i}_{10})$. The function $\omega \colon \Theta \mapsto \Omega$ maps each object to its type. Object-centric logs can be projected onto the sequence of activity labels from events that involve an object $o \in \Theta$. We write $L\!\downarrow_o$ for such an *object trace*. The projection of an object set $O \subseteq \Theta$ on an object type $ot \in \Omega$ is defined as $O\!\downarrow_{ot} = \{o \in O \mid \omega(o) = ot\}$. An activity a is *related* to a type ot if one of its objects is involved in an event with a: $\exists (a, O) \in L : O\!\downarrow_{ot} \neq \emptyset$.

Three important object-centric properties are the convergence, divergence and deficiency of object types [1]. We define these properties for $ot \in \Omega$ and $a \in \Sigma$. *Convergence* of ot and a describes the involvement of multiple objects of ot in an event with a: $\exists (a, O) \in L : |O\!\downarrow_{ot}| > 1$. *Divergences* of a and ot is the existence of different object sets for events with a that share any object of ot: $\exists (a, O), (a, O') \in L \colon O \neq O' \wedge (O \cap O')\!\downarrow_{ot} \neq \emptyset$. *Deficiency* of ot and a expresses that some event with a does not involve any object of ot: $\exists (a, O) \in L : O\!\downarrow_{ot} = \emptyset$.

We can observe these properties in the example from Table 1. Multiple items can be received at once, resulting in convergence. An operator can, over time, work on different item sets, resulting in divergence on these activities. The repair machines and operators are both deficient for the receiving and sending of items.

2.2 Petri Nets

Petri nets are graph structures with two different node types, called places and transitions, that are connected by directed arcs. They offer the possibility to replay transition sequences by passing conceptual *tokens* through the places of the net. Formally, a labeled Petri net is defined as a tuple (P, T, F, l), with a finite set of places P, a disjoint finite set of transitions T and a set of directed arcs F. The directed arcs in F always connect a place and a transition with $F \subseteq (T \times P) \cup (P \times T)$. The labeling function $l : P \mapsto \Sigma$ maps each transition to an activity label or the special character τ, representing unobserved behaviour.

Places and transitions are the two node types of the net, of which only places can contain tokens. We denote the number of tokens in each place of

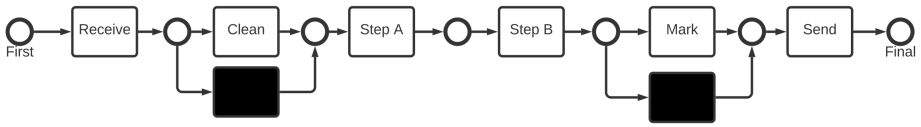

Fig. 3. Accepting Petri net for the behaviour of isolated items from Table 1.

the net in a *marking* $M \in P \mapsto \mathbb{N}$. We write $\bullet t = \{p \in P \mid (p,t) \in F\}$ and $t\bullet = \{t \in P \mid (t,p) \in F\}$ for the input and output places of a transition. A transition is *enabled* in a marking if tokens are available in all input places, i.e $\forall p \in \bullet t : M(p) > 0$. Firing an enabled transition t consumes and produces a token for each input and output place respectively, leading to a potentially different marking with $\forall p \in P : M'(p) = M(p) + |\{(p,t) \in F\}| - |\{(t,p) \in F\}|$.

We write $M \rightarrow^{t_1} \ldots \rightarrow^{t_n} M'$ for a *firing sequence* of the transitions $t_1, \ldots, t_n$ that start in a marking M, leading to M'. A firing sequence can be projected onto a trace of activity labels with $\langle l(t_1), \ldots l(t_n) \mid l(t_i) \neq \tau \rangle$. This mechanic can be used to define a language of traces. An accepting Petri net is a Petri net with the initial and final marking M_i and M_f. The language of such a net contains the projected traces of all firing sequences that lead form M_i to M_f.

Figure 3 shows an accepting Petri nets that represent the isolated behaviour items in our repair process. The first and final place have a single token in the initial and final marking of the net. Black boxes represent τ-transitions.

2.3 Object-Centric Petri Nets

Object-centric Petri nets extend Petri nets by offering the option to model multiple types at the same time, instead of specifying their behaviour in isolation [3]. For this purpose, an object type is assigned to each place of an underlying Petri net. Additionally, arcs can be marked as *variable* arcs, allowing the consumption and production of multiple tokens. Tokens are used to represent objects.

Formally, an object-centric Petri net (P, T, F, l, V, pt) is a labeled Petri net (P, T, F, l) with variable arcs $V \subseteq F$ and an object type mapping $pt : P \mapsto \Omega$. A marking in an such a net is a distribution of tokens that represent objects with $M : P \times \Theta \mapsto \mathbb{N}$. Each object's tokens can only reside in those places of the net that have the corresponding type: $\forall o \in \Theta : M(p,o) > 0 \implies \omega(o) = pt(p)$.

The notion of firing sequences is extended to account for the objects represented by tokens. We write $\bullet_v t = \{p \in P \mid (p,t) \in V\}$ and $t\bullet_v = \{p \in P \mid (t,p) \in$

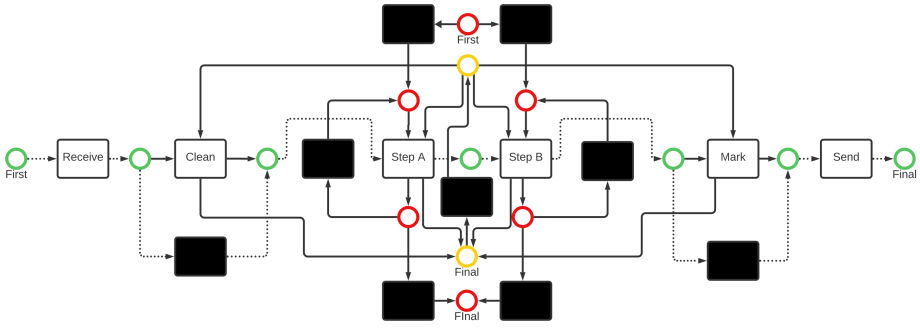

Fig. 4. Object-centic Petri net for our example repair process from Table 1.

$V\}$ for the input and output places of a transition t that are connected by variable arcs. A transition t is enabled by an object set O with $\forall ot \in \Omega : \forall p \in \bullet t : pt(p) = ot \iff O\lfloor_{ot} \neq \emptyset$ in a marking M if all objects of O are available in the input places of the transition: $\forall ot \in \Omega : \forall o \in O\lfloor_{ot} : \forall p \in \bullet t : (pt(p) = ot \implies M(p,o) > 0) \wedge (p \notin \bullet_v t \implies |O\lfloor_{ot}| = 1)$. A transition t that is enabled for an object set O in a marking M may fire and change the marking to the new marking M', with $\forall p \in P : \forall o \in O : M'(p,o) = M(p,o) + |\{(p,t) \in F\}| - |\{(t,p) \in F\}|$. A firing sequence with transitions and object sets $M \rightarrow^{t_1,O_1} \ldots \rightarrow^{t_n,O_n} M'$ can then be projected onto the log $\langle (l(t_1), O_1), \ldots (l(t_n), O_n) \mid l(t_i) \neq \tau \rangle$. An accepting object-centric Petri net is a net with an initial and final marking for a given set of objects. It defines a language of object-centric logs with these objects.

We consider nets with start and end place sets P_s and P_e. To replay a log, the initial and final marking are defined as $\forall o \in \Theta, p \in P_s : pt(p) = \omega(o) \implies M_i(p,o) = 1$ and $\forall o \in \Theta, p \in P_e : pt(p) = \omega(o) \implies M_f(p,o) = 1$. Figure 4 shows an example net, with colors for the types items (green), machines (red) and operators (yellow). Variable arcs are represented by dotted lines.

2.4 Object-Centric Conformance Checking

Conformance checking techniques quantify how well a model and a log match. The results of such methods are commonly used to quantitatively evaluate a model's ability to represent a given log. Specialized conformance checking methods for accepting object-centric Petri nets and object-centric logs have been proposed in [4] and [15].The general idea is to execute a firing sequences on the model that represents the log up until an event (a_i, O_i). Then, the method determines the labels of transitions enabled in the model $en(i, M)$ and the activity labels of the following events in the log $en(i, L)$. Fitness of the event describes how much behaviour contained in the log is enabled in the model, i.e $\frac{|en(i,L) \cap en(i,M)|}{|en(i,M)|}$. Analogously, precision of the event measures how much behaviour enabled in the model is contained in the log, i.e. $\frac{|en(i,L) \cap en(i,M)|}{|en(i,M)|}$. The fitness and precision of the full log is then the average fitness and precision of all events respectively.

3 Silent Objects

In this section, we define silent objects for synchronizations and specializations in object-centric event logs. Additionally, we introduce a modelling extension of object-centric Petri nets that includes both types of silent objects. Lastly, we show that the inclusion of these silent object in an object-centric Petri net preserves fitness and only increases precision for logs in which they are present.

3.1 Object Synchronizations & Object Specializations

Silent objects are unobserved objects that nonetheless affect observed objects. In a given log, we can, by definition, only observe their effect on the observed objects. The two effects of silent objects that we investigate in this paper, are

object synchronisations and object specializations. Objects that belong to the same unobserved group object, remain synchronized across a set of activities.

Definition 1. *An object set $O \subseteq \Theta$ in an object-centric log $L = \langle (a_1, O_1), \dots (a_n, O_n) \rangle$ remains synchronized for the activities $A \subseteq \Sigma$ if $\forall a \in A : \exists (a, O_i) \in L : O \subseteq O_i$ and $\forall (a_i, O_i) \in L : a_i \in A \wedge O \cap O_i \neq \emptyset \implies O \subseteq O_i$ hold.*

If we consider the example log from Table 1 we can see many object sets that fit this definition of object synchronizations. The item set $\{\mathtt{i}_4, \mathtt{i}_5, \mathtt{i}_6, \mathtt{i}_7, \mathtt{i}_8\}$, for example, remains synchronized across the two activities $\{\text{RECEIVE}, \text{SEND}\}$. Object specialization, in contrast, describe individual objects that participate in a set of optional activities because of an unobserved property of them.

Definition 2. *Let $L = \langle (a_1, O_1), \dots (a_n, O_n) \rangle$ be a log, with the object $o \in \Theta$ and $\Sigma'_o = \{a \in \Sigma \mid \exists o' \in \Theta|_{\omega(o)} : \forall (a, O_i) : o' \notin O_i\}$ the optional activities for $\omega(o)$. Then, $\{o\}$ specializes on $A \subset \Sigma'_o$ if $\forall a \in A : \exists (a, O_i) \in L : o \in O_i$.*

In our example from Table 1 we can observe object specializations as well. The item $\{\mathtt{i}_2\}$, for example, specializes on the activities $\{\text{CLEAN}, \text{MARK}\}$. In existing work on object-centric Petri nets, these two effects are not considered, leading to to the automated discovery of imprecise models. We use the net in Figure 4 for the log in Table 1 to illustrate the problem. Consider the first part of the log for our repair process, up to sending back the first set of items: $\langle \text{RECEIVE}\{\mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3\}, \text{CLEAN}\{\mathtt{i}_2, \mathtt{o}_1\}, \text{RECEIVE}\{\mathtt{i}_4, \dots, \mathtt{i}_8\}, \text{STEP A}\{\mathtt{i}_1, \dots, \mathtt{i}_5, \mathtt{o}_1, \mathtt{m}_1\}, \text{RECEIVE}\{\mathtt{i}_9, \mathtt{i}_{10}\}, \text{CLEAN}\{\mathtt{i}_{10}, \mathtt{o}_1\}, \text{STEP B}\{\mathtt{i}_1, \dots, \mathtt{i}_5, \mathtt{o}_1, \mathtt{m}_2\}, \text{MARK}\{\mathtt{i}_2, \mathtt{o}_1\}, \text{SEND}\{\mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3\}, \dots \rangle$ At this point, in the log, the items $\mathtt{i}_4, \mathtt{i}_5$ are already repaired and the items $\mathtt{i}_6, \dots, \mathtt{i}_{10}$ are waiting to be repaired. It is not possible to observe a second set of items to be send back yet. The items $\mathtt{i}_4, \mathtt{i}_5$ are already repaired, but they will only be sent out once the items $\mathtt{i}_6, \mathtt{i}_7, \mathtt{i}_8$ are also ready.

However, this is not reflected in the model. If we replay the log excerpt above, we see an important deviation of the model in the resulting marking. Figure 5 shows the item tokens in the corresponding marking. The tokens of the items

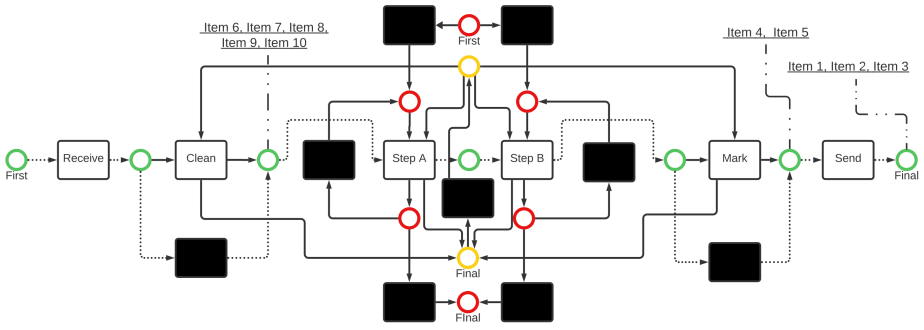

Fig. 5. Accepting object centric Petri net from Figure 4 with item tokens in the marking that occurs after replaying the log from Table 1 up until the event $(\text{SEND}, \{i_1, i_2, i_3\})$.

$\mathtt{i}_4, \mathtt{i}_5$ enable the model to send out this incomplete set of items, decreasing the precision of the model. The conceptual problem here, is that the object synchronization of the items $\{\mathtt{i}_4, \mathtt{i}_5, \mathtt{i}_6, \mathtt{i}_7, \mathtt{i}_8\}$ is not enforced. An analogue problem arises because the object specialization of item $\{\mathtt{i}_2\}$ is not enforced as well.

We propose to consider the unobserved group objects and object properties that cause these synchronizations and specializations as objects themselves. We introduce these silent objects as a modelling extension for object-centric Petri nets. Including them allows us to model object synchronizations and specialization, increasing the precision of the model for logs that include such effects.

3.2 Silent Object Types In Event Logs

Our example from the previous section illustrates that multiple object synchronizations and specialization might be present in a given log. Therefore, we need to differentiate between different object synchronizations and specialization. We do so by introducing the notion of silent object types. A silent object type represents one specific kind of object synchronization or object specialization. It is defined by the types of the affected objects, and the activities it applies to.

Definition 3. *A silent object type is a combination of a set of visible object types and a set of activities. We write $\Omega_\tau = \{ot_{C,A} \mid C \in \mathbb{P}(\Omega) \wedge A \in \mathbb{P}(\Sigma)\}$ for the set of silent object types for the visible object types Ω and the activities Σ.*

Formally, if $O \subseteq \Theta$ synchronizes or specializes on $A \subseteq \Sigma$, it has the silent object type $ot_{C,A}$ with $C = \{\omega(o) \mid o \in O\}$. Based on that rule, we can now assign silent object types to the examples discussed in the previous section. The object synchronization for the items that are received and send out, have the silent object type ot_{C_1,A_1} with $C_1 = \{\text{ITEMS}\} \wedge A_1 = \{\text{RECEIVE}, \text{SEND}\}$. Analogously, the object specialization of items that are cleaned and marked, corresponds to the silent object type ot_{C_2,A_2} with $C_2 = \{\text{ITEMS}\} \wedge A_2 = \{\text{CLEAN}, \text{MARK}\}$.

Given a silent object type and an event log, we can use the definitions of object synchronizations and specializations to identify all object sets that fit a silent object type. That is, they synchronize or specialize on the activities specified in the type. Note that in case of object specializations, these sets contain exactly one visible object. This leads to the notion of silent objects themselves.

Definition 4. *An object set $O \subseteq \Theta$ in a log L is a silent object of the type $ot_{C,A}$, if O is an object synchronization (1) or specialization (2) on A and the involved types match, i.e. $C = \{\omega(o) \mid o \in O\}$. We write $\omega(O) = ot_{C,A}$ in that case.*

This definition allows us, for a given log and silent object type, to determine all object sets that represent a silent object of this type. To do so, we check for every object set $O \subseteq \Theta$ if it has the corresponding silent object type. Applying this strategy to our running example reveals additional silent objects for the silent objects types discussed earlier. For ot_{C_1,A_1} we can, for example, identify the object sets of items $\{\mathtt{i}_1, \mathtt{i}_2, \mathtt{i}_3\}$ and $\{\mathtt{i}_9, \mathtt{i}_{10}\}$ as silent object of this type.

Formally, the object sets $\{\mathtt{i}_1, \mathtt{i}_2\}$, $\{\mathtt{i}_1, \mathtt{i}_3\}$ and $\{\mathtt{i}_2, \mathtt{i}_3\}$ also fulfill the properties for silent objects of ot_{C_1,A_1} in our log. However, their utility is highly questionable, since using them would result in the convergence of the silent object type. We refer to these kinds of silent objects as *covered* silent objects. For this paper, we only consider silent objects that are not covered. Future work will investigate if hierarchies of covered silent objects offer any modelling benefits.

Definition 5. *A silent object O in an object-centric log L is covered if there is a superset O' that is a silent object of the same type: $\omega(O) = \omega(O') \wedge O \subset O'$.*

Lastly, we need to determine if a log, potentially randomly, just includes a few instances of a silent object type, or if it actually adheres to it systematically. This tells us if we can use such a silent object type to accurately describe a log.

Definition 6. *An object-centric event log $L = \langle (a_1, O_1), \dots (a_n, O_n)$ adheres to a silent object type $ot_{C,A} \in \Omega_\tau$ if there is at least one object of this type $\exists O \subseteq \Theta : \omega(O) = ot_{C,A}$ and $\forall (a_i, O_i) \in L : a_i \in A \implies \omega(\cup_{ot \in C} O_i \lfloor_{ot}) = ot_{C,A}$.*

If a log adheres to a silent object type, it expresses that objects of the visible types that are involved in the affected activities always exhibit the corresponding object specialization or synchronization on them. In our repair process, this is for example the case for the two silent object types discussed above. Hence they should be expressed and enforced in a model for this process.

3.3 Silent Object Types in Object-Centric Petri Nets

Next, we formally extend object-centric Petri nets to include silent object types in them. We do so by introducing so-called synchronization and specialization-sensitive object-centric (SYSSOC) Petri nets. A SYSSOC Petri net is an object-centric Petri net, with an additional set of silent object types.

Definition 7. *Let (P, T, F, l, V, pt) be an object-centric Petri net for the visible object types Ω and the activities Σ. A SYSSOC Petri net $(P, T, F, l, V, pt, \Omega_s)$ has an additional set of silent object types $\Omega_s \subseteq \{ot_{C,A} \mid C \in \mathbb{P}(\Omega) \wedge A \in \mathbb{P}(\Sigma)\}$.*

Next, we define the semantics of SYSSOC nets by adjusting the underlying object-centric Petri net based on Ω_s. As a result we get an adjusted net (P', T, F', l, V, pt') on which we define the replay semantics. First, we introduce a new place for each silent object type with $P' = P \cup \{p_{C,A} \mid ot_{C,A} \in \Omega_s\}$. The types of these places are the silent object types that they represent. They are bidirectionally connected to transitions with activities that involve the silent type with $F' = F \cup \{(p_{C,A}, t), (t, p_{C,A}) \mid ot_{C,A} \in \Omega_s \wedge t \in T \wedge l(t) \in A\}$.

A marking in these nets is a distribution of tokens to represent sets of objects in each place: $M : P' \times \mathbb{P}(\Theta) \mapsto \mathbb{N}$. All places that are mapped to a visible object type, can only contain sets with a single object of the correct object type, i.e. $\forall p \in P' : pt'(p) \in \Omega \implies \forall O \in \mathbb{P}(\Theta) : (M(p, O) = 0 \vee (O = \{o\} \wedge \omega(o) = pt'(p)))$. Places of silent object types can contain any sets, but the types of

the visible objects contained in them need to be correct: $\forall p \in P' : (pt'(p) \in \Omega_s \wedge pt'(p) = ot_{C,A}) \implies (\forall O \in \mathbb{P}(\Theta) : (M(p,O) = 0 \vee (\{\omega(o) \mid o \in O\} = C)))$.

Next we define the notion of enabled transitions in a given marking. We again use $\bullet_v t = \{p \in P' \mid (p,t) \in V\}$ and $t\bullet_v = \{p \in P' \mid (t,p) \in V\}$ to denote the input and output places of a transition t that are connected by variable arcs. Similar to object-centric Petri nets, transitions in a marking are enabled by an object set O. This set must contain objects that correspond to the visible types of the input places $\forall p \in \bullet t : pt(p) \in \Omega \iff O\restriction_{ot} \neq \emptyset$. If an input place is not connected by a variable arcs, then only one object of the corresponding type can be used $\forall p \in \bullet t : p \notin \bullet_v t \implies |O\restriction_{pt(p)}| = 1$. A transition is enabled by such an object set O in a marking M, if all objects of O are individually available in the input places of the transition with visible types: $\forall ot \in \Omega : \forall o \in O\restriction_{ot} : \forall p \in \bullet t : (pt(p) = ot \implies M(p,\{o\}) > 0)$. Additionally, a token for the objects that are covered by any involved silent object type, must be available in the corresponding place as well: $\forall ot_{C,A} \in \Omega_s : \forall p \in \bullet t : (pt(p) = ot_{C,A} \implies M(p, \cup_{ot \in C} O\restriction_{ot}) > 0)$.

A transition t that is enabled for an object set O in a marking M may fire and change the marking of the visible objects into M', with $\forall p \in P : \forall o \in O : M'(p,\{o\}) = M(p,\{o\}) + |\{(p,t) \in F\}| - |\{(t,p) \in F\}|$. The marking in the places of silent object types never changes, since they are bi-directionally connected to any transition they are involved in. The notions of firing sequences, log projections and languages in SYSSOC nets is then defined analogue to those concepts in object-centric Petri nets with the use of initial and final markings.

To define an initial and final marking for the replay of a specific log, we employ a similar technique as described for object-centric Petri nets in Section 2.3. Let P_s and P_e contain the designated start and end places for each visible object type of the underlying object-centric Petri net. The initial marking of the net for the replay of a specific logs contains one token for each visible object in the respective start place: $\forall o \in \Theta : \forall p \in P_s : pt(p) = \omega(o) \implies M_i(p,\{o\}) = 1$.

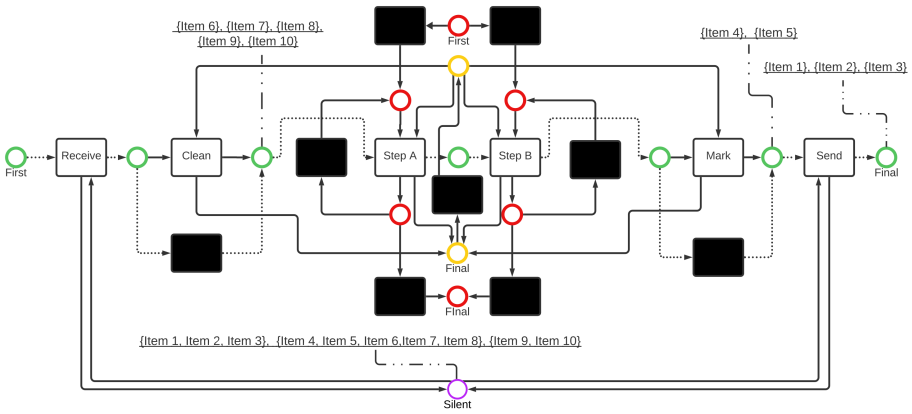

Fig. 6. Adjusted SYSSOC Petri net with item tokens.

For the final marking, we analogously have $\forall o \in \Theta : \forall p \in P_e : pt(p) = \omega(o) \implies M_f(p, \{o\}) = 1$. Additionally, we determine the set of silent objects for each silent object types and put a token for the respective object set in the silent object types place. This is simple task, given that we only consider object sets that are not covered and have the silent object types given to us by the model with Ω_s. For each silent object type $ot_{C,A} \in \Omega_s$ and potential object set $O \subseteq \Theta$ with $\exists (a_i, O_i) \in L : O = \cup_{ot \in C} O_i \restriction_{ot}$ we check if O is an object synchronization or specialization for the activities A with the definition form Section 3.1. If this is the case, we adjust the markings of the net, i.e. $M_i(p_{C,A}, O) = M_f(p_{C,A}, O) = 1$.

For our repair process, we can now solve the problem discussed in Section 3.1. For our example of item groups that are received and sent out together, we do so by including the corresponding silent object type ot_{C_1,A_1} in a SYSSOC Petri net that is based on the object-centric Petri net in Figure 4. Figure 6 shows the SYSSOC net with the tokens for items and silent objects in the marking that arises if we replay the problematic situation described in Section 3.1. The difference here is that the transition for sending out activities is not enabled.

Lastly, we discuss the influence of silent object types on conformance checking methods and results. In principle, specialized conformance checking techniques for object-centric Petri nets, such as those from [4] can also be applied to SYSSOC nets with minor adjustments. The key conceptual idea of how to measure precision and fitness in terms of $en(i, M)$ and $en(i, L)$ as explained in Section 2.4 does not need to change. Only the calculation of $en(i, M)$ needs to be adapted, by accounting for the new notion of when a transition is considered enabled.

For any conformance checking techniques that follows this approach, we can provide the following guarantees. Let $N = (P, T, F, l, V, pt)$ be an object-centric Petri net, L be an object-centric event log and Ω_s be a set of silent object type that adhere to L. Then, the fitness of the SYSSOC net $S = (P, T, F, l, V, pt, \Omega_s)$ for L is identical to the fitness of N for L. Additionally, the precision of S for L is equal or higher than the precision of N for L for the following reasons:

The enabled transitions in the model can only be reduced by the inclusion of silent object types, i.e. $en(i, S) \subseteq en(i, N)$. However, since every silent object type that might restrict a transition adheres to the log, none of the correctly enabled transitions in the model are affected: $a \in en(i, N) \cap en(i, L) \implies a \in en(i, S) \cap en(i, L)$. It follows directly that $en(i, S) \cap en(i, L) = en(i, N) \cap en(i, L)$, i.e. the fitness value are identical. Additionally, $en(i, S) \subseteq en(i, N)$ implies $|en(i, S)| \leq |en(i, N)|$ and hence $\frac{en(i,S) \cap en(i,L)}{en(i,S)} \geq \frac{en(i,N) \cap en(i,L)}{en(i,N)}$. That is, the precision can only be identical or increase due to adding silent object types.

4 Discovering Silent Object Types

In this section, we specify our algorithm to discover SYSSOC nets from object-centric logs. That is, given an object-centric log L, we can use [3] to discover an object-centric Petri net (P, T, F, l, V, pt) and our technique to discover the set of silent object types Ω_s for L, resulting in the SYSSOC net $(P, T, F, l, V, pt, \Omega_s)$. Additionally, we discuss our approach's guarantees and runtime complexity.

4.1 Silent Object Type Discovery

The number of potential silent object types that adhere to a log grows exponentially in the size of Ω and Σ. We focus on the identification of silent object types that do not exhibit unnecessary convergence, divergence and deficiency.

We propose a three staged approach for the detection of such silent object types. First, we exploit the property of divergence to detect combinations of visible object types that are covered by a silent object type. Then, we determine the activity sets for which these object combinations are relevant. Lastly, we filter the detected silent object types, to make sure no redundancies are included.

Object sets of visible types, that do not diverge for an activity a in combination, are by definition sufficient to identify all unique events with a.

Definition 8. *A set of object types $C \subseteq \Omega$ is an* identifier *for an activity a in the log L if all projections of object sets in events with a on the types in C are unique:* $|\{O' \mid \exists (a,O) \in L : \bigcup_{ot \in C} O\lfloor_{ot} = O'\}| = |\{O \mid \exists (a,O) \in L\}|$.

There might be multiple object type combinations C, C' that are identifiers for the same activity, with $C' \subset C$. This indicates that the additional types included in C are not actually needed for the identification of events. Therefore, we are only interested in minimal identifier types for the involved visible types, i.e. identifiers C for which there is no other identifier C' such that $C' \subset C$.

All minimal identifiers can be found using an iterative approach for each activity $a \in \Sigma$. We start with the set of object types that is an identifier for all activities: $C = \Omega$. Then, we separately remove each visible type from C resulting in the candidates of a smaller size $T_{|\Omega|-1} = \{C' \mid |C'| = |C| - 1 \wedge C' \subset C\}$. We check which object type combinations in $T_{|\Omega|-1}$ are indeed identifiers, denoted by $V_{|\Omega|-1}$. We iteratively decrease the size of the object type combinations until no more identifiers are found. We then pick all found minimal identifiers.

Next, we provide a method to detect the maximal activity sets for each minimal identifier C, i.e. the sets of activities in which the same object combinations of the types in C are used. This can be done in polynomial time complexity. Let V_a be the set of minimal identifiers for each activity $a \in \Sigma$ and $\Sigma_C = \{a \in \Sigma \mid C \in V_a\}$. For each identifier $C \in \bigcup_{a \in \Sigma} V_a$ and each activity pair $(a, a') \in \Sigma_C \times \Sigma_C$, we check if the same object sets of the types in C are used.

Definition 9. *Let $C \subseteq \Omega$ be a minimal identifier for the activities $a, a' \in \Sigma$ in the log L with $C \in V_a \cap V_{a'}$. C is a shared minimal identifier for the activity pair (a, a') if $\{O' \mid \exists_{(a,O) \in L} : \bigcup_{ot \in C} O\lfloor_{ot} = O'\} = \{O' \mid \exists_{(a',O) \in L} : \bigcup_{ot \in C} O\lfloor_{ot} = O'\}$. We write $V_{a,a'}$ for the set of all shared minimal identifiers.*

We then construct an undirected graph for each of the minimal identifiers $C \in \bigcup_{a \in \Sigma} V_a$, to detect maximal activity sets that share the identifier. These graphs include nodes for each activity in Σ_C and edges between $a, a' \in \Sigma_C$ if $C \in V_{a,a'}$. Each connected component in these graphs corresponds to a maximal activity set for which C is a shared minimal identifier. We denote the set of maximal activity sets for a minimal identifier C with M_C. The combinations

of minimal identifiers with their maximal activity set define the detected silent object type for a given input log: $\{ot_{C,A} \mid \exists a \in \Sigma : C \in V_a \wedge A \in M_C\}$.

Lastly, we need to check if any of the detected silent types corresponds to a visible type in the log L. For this purpose, we check if a detected $ot_{C,A}$ can be replaced with an object type such that $\forall a \in \Sigma : (\exists (a, O) \in L : O \restriction_{ot} \neq \emptyset \iff a \in A)$. If this is the case, we drop the detected silent object type.

4.2 Formal Guarantees

Our detection strategy guarantees all silent object types $ot_{C,A}$ discovered from a log to not be divergent, convergent or deficient for the activities in A. The convergence of $ot_{C,A}$ for an activity $a \in A$ requires the existence of two silent objects of this type in the same event of the log, i.e $\exists (a, O) \in L : \exists O', O'' \subseteq O : \omega(\tau_{O'}) = ot_{C,A} \wedge \omega(\tau_{O''}) = ot_{C,A}$. However, due to the same type of the two silent objects we have $O' = \bigcup_{ot \in C} O \restriction_{ot} = O'$ and $O'' = \bigcup_{ot \in C} O \restriction_{ot} = O'$, which implies $O' = O''$. Hence, the silent object type cannot be convergent. Furthermore, the object type combination C is by definition an identifier for all activities in A. Therefore, we have $|\{O' \mid \exists (a, O) \in L : \bigcup_{ot \in C} O \restriction_{ot} = O'\}| = |\{O \mid \exists (a, O) \in L\}|$ for all $a \in A$. This proves that the silent object type cannot be divergent or deficient, since the definition of these properties both imply $|\{O' \mid \exists (a, O) \in L : \bigcup_{ot \in C} O \restriction_{ot} = O'\}| \neq |\{O \mid \exists (a, O) \in L\}|$.

Our approach has an exponential worst-case time complexity of $O(2^{|\Omega|} \cdot |\Theta| \cdot |L| \cdot |\Sigma| + |\Omega| \cdot |\Theta| \cdot |L| \cdot |\Sigma|^2)$. For each activity ($|\Sigma|$), we need to check at most $2^{|\Omega|}$ object type combinations for identifiers, which takes at most $|L| \cdot |\Theta|$ steps per combination. Out of these combinations at most $|\Omega|$ identifiers can be minimal, leading to the construction of the undirected graph for shared identifiers between at most $|\Sigma|^2$ activity pairs, with each check taking at most $|L| \cdot |\Theta|$ steps again. The detection of connected components can be done in quadratic time complexity. However, we do not expect real-life logs to actually require this worst-case time complexity for the following reasons: In practice, the number of identifiers for an activity is usually much lower than the number of visible object type combinations. Once our technique detects a non-identifier combination of the size $|C|$, all $2^{|C|} - 2$ subsets of this combination are not considered anymore. Similarly, the subset of activities that contain all object types in a given C are often small. Hence, the undirected graphs that represent the shared identifiers for any activity pair become smaller in size. Therefore, we expect $2^{|\Omega|}$ and $|\Sigma|^2$ to decrease significantly when applying our technique in practice.

The worst-case space complexity of our approach is $O(min(|L|, 2^{|\Omega|}) \cdot |\Theta| + |\Sigma| \cdot |\Omega| + |\Sigma|^2 + |\Sigma|)$. We only ever check one type set for being a (shared) identifier at a time. Such a check requires us to store the set of all unique object sets for a set of events. There can be at most $min(|L|, 2^{|\Theta|})$ of such events with unique object sets, with the size of each set being bound by $|\Theta|$. For each activity, we store the minimal identifiers, resulting in a storage load of at most $|\Omega| \cdot |\Sigma|$. The same storage capacity is needed to store the maximal activity sets for each minimal identifier. The graphs used to determine these sets require us to store at most $|\Sigma|$ nodes and $|\Sigma|^2$ edges. We only ever store one of these graphs.

5 Evaluation

In this section, we evaluate our approach. We apply our detection algorithm to public logs to check if there are any silent object types in them. We investigate the runtime of our algorithm on these logs, to evaluate it's feasibility. Lastly, we manually investigate some examples and discuss limitations. Our implementation is available at https://github.com/Nik314/BPM_2024_Silent_Objects.

We apply our implementation to six public logs, which are part of the log library for the OCEL file format at https://www.ocel-standard.org/1.0/ or the MIMIC-III database as specified in [12, 14]. Table 2 shows relevant properties of the utilized logs. We apply our algorithm to all of these logs and observe the runtime for the discovery of silent object types. We use a computer with an Intel Core i5-8265U CPU and 16GB of working memory for this experiment. Additionally, we manually inspect some discovered silent object types.

5.1 Experimental Results

We observed the runtime in seconds for each of the logs, as specified in Table 2. These results indicate that our approach is successful in not approaching its exponential upper bound in runtime complexity. Despite the order-to-cash log having seventy times more objects, three times more types and twice the activities of the HR recruiting log, runtime only increases by a factor of nine.

Additionally, we can see that in all of these logs, silent object types are indeed present. Most of the detected silent object types are conceptually similar to those discussed in our repair process. The order logistic log contains, for example, a silent object type for items and the activities OUT OF STOCK and REORDER. This silent object type expresses that only out of stock items can be reordered.

Some more interesting examples for multiple visible object types being involved in a silent object type were found in the HR recruiting log. It contains, for example, a silent object type that combines the visible types of applicants and vacancies for the activities INVITE FOR INTERVIEW and CONDUCT INTERVIEW. The unobserved objects here are clearly the interviews themselves. The inclusion of a corresponding silent object type in a model, makes sure that an applicant is only interviewed for those vacancies for which they were invited for.

Table 2. Properties, runtime and silent object types for logs from [7, 12–14].

Log	Events	Activities	Objects	Types	Runtime	Silent
P2P	24854	32	74489	8	71s	73
O2C	98350	23	107767	19	182s	23
Recruiting	6980	16	1505	6	20s	21
Logistics	22367	11	11521	5	77s	12
Transfer	10319	3	2500	5	50s	3
MIMIC-III	3007	3	13410	3	11s	3

5.2 Discussion

Our evaluation shows that there are silent object types in public object-centric logs that we can detect automatically in a feasible runtime with our approach. Furthermore, we can guarantee these types to be free of convergence, divergence and deficiency for all activities in which they are involved. Additionally, we can guarantee that the inclusion of these types in a SYSSOC nets preserves fitness and can only improve precision, as discussed in previous sections. However, these guarantees comes at the cost of excluding all potential instances for covered silent objects, as explained in Section 3.2. Future work should investigate if hierarchies of covered silent object offer modeling benefits. Our approach, in principle, could be used to detect those hierarchical objects, by including a threshold for the allowed degree of divergence, convergence and deficiency.

However, further types of unobserved objects might exist in real-life processes that exhibit a different, or no, effect on the observed objects besides synchronizing or specializing them. These would require different detection strategies. Furthermore, we do not know to which extent the detected silent object types actually correspond to unobserved objects. We did perform some manual sanity check, as for the silent object types that we discussed in the previous section, to see to which kind of unobserved real-life objects the silent object types could correspond to. However, there is no guarantee that these unobserved objects are actually there. It seems unlikely to us, but still possible, that objects randomly exhibit synchronizations or specializations across multiple activities.

6 Related Work

We summarize existing approaches to model and discover constructs with similar effects as silent objects in business processes with interacting object types.

Proclet-based [2] and artifact-centric [8] approaches contain models for each object type that interact through specialized constructs. They can be used to model various communication and interaction patterns between objects, including synchronizations. Extensions of such formalism to cover many-to-many relations between objects have been proposed and discussed in [9] and [10]. For this purpose, many-to-many relations between observed objects are manifested as objects themselves, similar to silent objects. However, these relations can have their own lifecycle and control flow attached to them, giving them a dynamic nature. Additionally, objects in a many-to-many relation never directly interact with each other, but only through the manifested object of the shared relation, decoupling them to some extent in the model. Conceptually, this is very different from the notion proposed in this paper, which is entirely static and strongly connected. That is, silent objects are fixed a-priori in the initial marking, based on observations in the data, and the objects they affect directly interact in shared transitions. As a result, they are less expressive but easier to integrate into a joint process model, following the line of research on object-centric Petri nets.

Due to the similarities between silent object type and many-to-many relation manifestations in artifact-centric models, our detection strategy for silent objct

types is conceptually similar to the discovery algorithm for artifacts discussed in [16]. However, our formalization rooted in object-centric properties allows us to provide guarantees on the absence of divergence, convergence and deficiency.

Another line of research focuses on explicitly representing and accounting for object identifiers with regards to the control flow of a model. Corresponding approaches include v-nets [17] and a range of extensions, such as Petri nets with identifiers [11] and typed Jackson nets [6]. In these formalisms, object synchronizations are naturally represented by sets of object identifiers in designated places. In principle, this is very similar to our approach, as we apply the same idea with silent object types to object-centric Petri nets, in which identifiers, so far, have not been considered to restrict the control flow. However, as discussed above already, our approach does not dynamically create and resolve these object sets for synchronizations. This allows us to integrate silent object types into existing discovery techniques for object-centric Petri nets, enabling an automated discovery of SYSSOC nets. The automated discovery of typed Petri nets with identifiers and typed Jackson nets mostly remains an open research topic.

In summary, our approach applies existing ideas for the representation of object sets, in the form of silent objects, to object-centric Petri nets. As such, it enhances the expressiveness of this formalism and enables the construction of joint process models for multiple object types with improved precision. However, it sacrifices expressiveness in comparison to formalisms such as [6], by not dynamically creating and resolving the included object sets. In return, it gains the ability to automatically discover models with silent object types from logs.

7 Conclusion

In this paper, we introduced silent objects for object-centric Petri nets. These objects describe unobserved entities that have an observable effect on other objects. In a log, they are not explicitly recorded but can be inferred due to object synchronizations and specializations on activities. We sketched the minimal changes necessary for existing conformance checking techniques to take silent objects into account. We showed that the inclusion of silent objects in a model preserves fitness and increases precision if they are present in a log. Additionally, we introduced an algorithm to automatically construct object-centric Petri nets with silent object types from a given log. We proved that the inclusion of silent object types does not suffer from divergence, convergence or deficiency. In our evaluation, we showed that our approach is feasible in terms of runtime and scalability. Additionally, we illustrated the presence of silent object types in publicly available object-centric logs. Future work will investigate if hierarchies between covered silent objects offer any modeling benefits and if they can be integrated into existing process discovery algorithms. Additionally, we will observe if dynamic behaviour of silent object types can be added to object-centric Petri nets, without sacrificing the ability to automatically discover process models.

References

1. van der Aalst, W.M.P.: Object-centric process mining: Dealing with divergence and convergence in event data. In: 17th SEFM Proceedings. Springer (2019)
2. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A framework for lightweight interacting workflow processes. Int. J. Cooperative Inf. Syst. **10**(4), 443–481 (2001). https://doi.org/10.1142/S0218843001000412
3. van der Aalst, W.M.P., Berti, A.: Discovering object-centric petri nets. Fundam. Informaticae **175**(1-4), 1–40 (2020). https://doi.org/10.3233/FI-2020-1946
4. Adams, J.N., van der Aalst, W.M.P.: Precision and fitness in object-centric process mining. In: 3rd ICPM Proceedings. IEEE (2021)
5. Aguilar-Savén, R.S.: Business process modelling: Review and framework. International Journal of Production Economics (2004). https://doi.org/https://doi.org/10.1016/S0925-5273(03)00102-6
6. Barenholz, D., Montali, M., Polyvyanyy, A., Reijers, H.A., Rivkin, A., van der Werf, J.M.E.M.: There and back again - on the reconstructability and rediscoverability of typed jackson nets. In: 44th PETRI Proceedings. Lecture Notes in Computer Science, Springer (2023). https://doi.org/10.1007/978-3-031-33620-1_3
7. Berti, A., Park, G., Rafiei, M., van der Aalst, W.M.P.: An event data extraction approach from SAP ERP for process mining. In: 1st ICPM. Springer (2021)
8. Fahland, D.: Artifact-centric process mining. In: Encyclopedia of Big Data Technologies. Springer (2019). https://doi.org/10.1007/978-3-319-63962-8_93-1
9. Fahland, D.: Describing behavior of processes with many-to-many interactions. In: PETRI - 40th International Conference (2019), `https://doi.org/10.1007/978-3-030-21571-2_1`
10. Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.M.P.: Many-to-many: Some observations on interactions in artifact choreographies. In: 3rd ZEUS Workshop. CEUR-WS.org (2011), `https://ceur-ws.org/Vol-705/paper1.pdf`
11. van Hee, K.M., Sidorova, N., Voorhoeve, M., van der Werf, J.M.E.M.: Generation of database transactions with petri nets. Fundam. Informaticae **93**(1-3), 171–184 (2009). https://doi.org/10.3233/FI-2009-0095
12. Johnson, A., Pollard, T., Shen, L., Lehman, L.w., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L., Mark, R.: Mimic-iii, a freely accessible critical care database. Scientific Data **3**, 160035 (05 2016). https://doi.org/10.1038/sdata.2016.35
13. Koren, I., Adams, J.N., Berti, A., van der Aalst, W.M.P.: OCEL 2.0 resources - www.ocel-standard.org. In: Doctoral Consortium ICPM). CEUR-WS.org (2023)
14. Leemans, S.J.J., Mannel, L.L., Sidorova, N.: Significant stochastic dependencies in process models. Inf. Syst. **118**, 102223 (2023). https://doi.org/10.1016/J.IS.2023.102223
15. Liss, L., Adams, J.N., van der Aalst, W.M.P.: Object-centric alignments. In: Conceptual Modeling - 42nd International Conference. Springer (2023)
16. Popova, V., Fahland, D., Dumas, M.: Artifact lifecycle discovery. Int. J. Cooperative Inf. Syst. **24**(1), 1550001:1–1550001:44 (2015). https://doi.org/10.1142/S021884301550001X
17. Rosa-Velardo, F., de Frutos-Escrig, D.: Decidability and complexity of petri nets with unordered data. Theor. Comput. Sci. **412**(34), 4439–4451 (2011). https://doi.org/10.1016/J.TCS.2011.05.007