Object-Centric Alignments

unknown author

unknonw institute
unknown email

Abstract. Processes tend to interact with other processes and operate on various objects of different types. These objects can influence each other creating dependencies between sub-processes. Analyzing the conformance of such complex processes challenges traditional conformance-checking approaches because they assume a single-case identifier for a process. To create a single-case identifier one has to flatten complex processes. This leads to information loss when separating the processes that interact on some objects. This paper introduces an alignment approach that operates directly on these object-centric processes. We introduce alignments that can give behavior-based insights into how closely related the event data generated by a process and the behavior specified by an object-centric Petri net are. The contributions of this paper include a definition for object-centric alignments, an algorithm to compute them, a publicly available implementation, and a qualitative and quantitative evaluation. The qualitative evaluation shows that object-centric alignments can give better insights into object-centric processes because they correctly consider inter-object dependencies. Findings from the quantitative evaluation show that the run-time grows exponentially with the number of objects, the length of the process execution, and the cost of the alignment. The evaluation results motivate future research to improve the run-time and make object-centric alignments more applicable for larger processes.

Keywords: Process mining · Object-centric process mining · Alignments.

1 Introduction

Process mining provides insights into processes by analyzing event data generated by these processes. When analyzing a process, one standard pipeline consists of extracting data, discovering a process model, and checking the conformance of the process with specifications [13]. This paper presents an approach to compute alignments to check the conformance of object-centric processes for which traditional conformance-checking methods fail to give correct insights.

Traditional process mining approaches depend on the assumption that a process is defined by a single case notion meaning that all actions created for one object define a process execution. Processes in the real world tend not to fit that assumption. An example of that is a typical supply chain process. Supply processes are happening on raw materials, production processes on raw materials

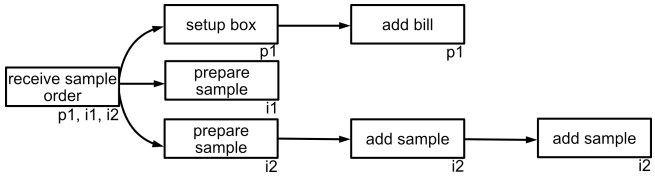
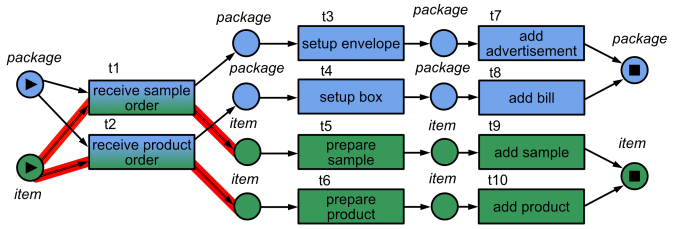


Fig. 1. A process execution of our running examples. Events are associated with objects of type package (prefix p) or item (prefix i). The process execution describes the partial order of events induced by the individual objects.



Package p1:

receive sample order	>>	setup box	add bill
>>	t2: receive product order	t4: setup box	t8:add bill

Item i1:

receive sample order	prepare sample	>>
t1: receive sample order	t5: prepare sample	t9: add sample

Item i2:

receive sample order	prepare sample	add sample	add sample
t1: receive sample order	t5: prepare sample	t9: add sample	>>

Fig. 2. Top: De-jure model as an object-centric Petri net. Variable arcs in the model are marked red. Bottom: The process execution of Figure 1 is flattened to the individual objects and aligned to the de-jure model's subnet of the object's types.

and products, shipping processes on products and orders, and payment processes operating on orders and customers. One execution of the supply chain is not defined by a single object. Real-world processes consist of multiple sub-processes operating on multiple object instances from various types. These sub-processes can have synchronization points and long-term dependencies between different objects.

Recently, approaches have been made to generalize the notion of a process so that one can describe these complex processes. Those approaches fall under the umbrella term of *object-centric process mining* [20]. Object-centric process mining generalizes traditional process mining techniques such that one does not follow one object through one process, but multiple objects through multiple, connected sub-processes. This generalization increases the complexity. In traditional process mining, one execution of a process is called *case* and is defined by a sequence of events [1]. An object-centric *process execution* is a graph describing the partial order between events in different sub-processes [7]. So far, a fitness notion based on replay has been proposed for object-centric conformance

checking, but the approach is constrained to replayable behavior[5]. However, process owners are typically interested in aligning observed behavior to modeled behavior, finding deviations in their execution of the process, i.e., using alignments [8]. The notion, calculation, implementation, and feasibility analysis for alignments on object-centric process mining are, so far, missing.

The running example used in this paper describes a packaging process with cross-object dependencies. In our example, one process execution refers to multiple items and one package. Figure 1 depicts a process execution as a graph of events describing the partial order between item and package objects. Figure 2 shows an object-centric Petri net [4] of the packaging process. An object-centric Petri net differs from traditional Petri nets by introducing place types and variable arcs. Tokens are typed and places of one type can only hold tokens of the same type. Variable arcs can consume an arbitrary amount of tokens. Places are typed according to either item (green) or package (blue). We, furthermore, colored transitions with the colors of the types that are involved in this transition for clarity and highlighted the variable arcs with red. In the given process, the path of the package depends on the path of items and vice-versa. The package and the items can either be part of a sample order or a product order, but never both, depending on whether *receive sample order* or *receive product order* fires. Although the objects do not share any more events after the start, their allowed behavior still depends on whether they belong to a sample or product order.

If a process owner would like to find deviations in their object-centric processes today, they would need to *flatten* [2] the observed process executions and apply traditional alignments to the object-centric Petri net's subnets of the same type. We show this for our example process execution of Figure 1 in Figure 2. If flattened to one trace per object, the three resulting traces get aligned to the type's subnet in a way that is not possible in the composed model. As mentioned, activity *receive sample order* and *receive product order* can never happen both in one process execution since the de-jure model forces a decision between product orders and sample orders. But the flattened alignments do not agree on which activity should happen. The alignment for *p1* has *receive product order* in the model part whereas the alignments for *i1* and *i2* have *receive sample order* in their model part. As shown by the running example, computing alignments on object-centric processes requires more than just finding alignments for each object individually. Aligning the sub-processes for all objects by respecting their object dependencies creates a computationally complex problem that we tackle in this paper.

This paper presents four contributions to enable and investigate object-centric alignments. First, we generalize the notion of an alignment to object-centric processes. Second, we present an algorithm to compute optimal object-centric alignments. Third, we implemented our algorithm and make it publicly accessible as an open-source project[1] based on the open-source object-centric process mining library OPCA [6]. Fourth, we evaluate the quality and the com-

[1] https://github.com/LukasLiss/object-centric-alignments

putation time of object-centric alignments on real-world event data. Thereby, we gain insights into the scalability and suitability of the approach.

This paper is structured in the following way. We present related work in Section 2 and preliminaries in Section 3. Then, we define object-centric alignments in Section 4. Our algorithm to compute alignments consists of two parts: constructing the synchronous product net (Section 5) and finding an optimal alignment in the synchronous product net (Section 6). We present a qualitative and quantitative evaluation in Section 7 and conclude the paper in Section 8.

2 Related Work

Process mining includes discovery, conformance checking, and enhancement of business processes [1]. Our approach belongs to the category of conformance checking, where behavior from the event log is compared to allowed behavior that is specified by a de-jure model [13]. For traditional processes, there exists a variety of conformance checking approaches [19]. The majority of them use either a token-based replay [25] approach, or an alignment [8] approach. Both have been used to derive quality metrics like precision [9] and fitness [10]. Unlike token-based replay, alignments are independent of the structure of the de-jure model [8]. Like our calculation, the traditional alignment calculation defined by Adriansyah et al. uses a two-step algorithm to compute alignments [8]. Adriansyah et al.'s approach creates a synchronous product net such that finding optimal alignments relates to finding a shortest path in that net. This is a well-studied problem that can be solved with the Dijkstra [16] or A^* [14] algorithm. Different ways to speed up the calculation have been researched [18][27]. However, the alignment algorithm assumes the process to have a single case identifier and can therefore not be used for compositions of processes that operate on multiple objects.

Multiple extensions to the traditional alignment algorithm have been made over the years that consider additional dimensions together with the workflow dimension [11][12]. Thereby they use higher-order nets to represent the additional dimensions. The data and resource-aware conformance checking approach from de Leoni et al. uses data Petri nets [23]. Felli et al. use data Petri nets together with satisfiability modulo theories to compute data-aware alignments [21]. Sommers et al. constructed a ν-Petri net to calculate resource-constrained alignments [26]. But all of the approaches above assume the process to have a single-case identifier.

There are approaches that lift this generalization and model processes as interacting sub-processes. Multi-agent process models describe the behavior of agents and their interaction by composing Petri nets [24]. Object-centric process mining [2] extends the notion of processes so that they can interact and operate on objects from different types. Adams et al. defined the notion of cases and variants for object-centric processes [7] as event graphs instead of event sequences. The defined process executions serve as input for our alignment calculation. The other data type that we use as input is an object-centric Petri net [4] that can describe allowed behavior. Object-centric Petri nets can be discov-

ered from object-centric event logs using the discovery algorithm from van der Aalst and Berti [4]. Precision and fitness metrics to evaluate the quality of a model have, recently, been proposed [5]. However, techniques to check conformance to a de-jure model and spot deviations, such as object-centric alignments, are so far missing.

3 Preliminaries

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

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

- *$E \subseteq \mathbb{U}_{event}$ is a set of events, $O \subseteq \mathbb{U}_{obj}$ is a set of objects,*
- *$OT = \{\pi_{type}(o) | o \in O\}$ is a set of object types,*
- *$\pi_{act} : E \rightarrow \mathbb{U}_{act}$ maps each event to an activity,*
- *$\pi_{obj} : E \rightarrow \mathcal{P}(\mathbb{U}_{obj}) \setminus \{\emptyset\}$ maps each event to at least one object,*
- *$\pi_{time} : E \rightarrow \mathbb{U}_{time}$ maps each event to a timestamp, and*
- *$\pi_{trace} : O \rightarrow E^*$ maps each object onto a sequence of events such that $\pi_{trace}(o) = \langle e_1, ..., e_n \rangle$ with $\{e_1, ..., e_n\} = \{e \in E | o \in \pi_{obj}(e)\}$ and $\forall_{i \in \{1,...,n-1\}} \pi_{time}(e_i) \leq \pi_{time}(e_{i+1})$*

Event logs can contain events from multiple process executions. When analyzing the behavior we want to extract one process execution.

Definition 2 (Process Execution). *Let $L = (E, O, OT, \pi_{act}, \pi_{obj}, \pi_{time}, \pi_{trace})$ be an object-centric event log. The object graph $OG_L = (O, I)$ with $I = \{\{o, o'\} | \exists_{e \in E} \{o, o'\} \subseteq \pi_{obj}(e) \wedge o \neq o'\}$ connects objects that share events. The connected components $con(L) = \{X \subseteq O | X$ is a connected component in $OG_L\}$ of the object graph are sets of inter-dependent objects. Each set $X \in con(L)$ defines a process execution of L. A process execution is a graph $P_X = (E_X, D_X)$ with nodes $E_X = \{e \in E | X \cap \pi_{obj}(e) \neq \emptyset\}$ and edges $D_X = \{(e, e') \in E_X \times E_X | \exists_{o \in X, 1 \leq i < n} \langle e_1, ..., e_n \rangle = \pi_{trace}(o) \wedge e = e_i \wedge e' = e_{i+1}\}$. The set $px(L) = \{P_X | X \in con(L)\}$ contains all process executions of event log L.*

Figure 1 shows the process execution of the running example. Object-centric Petri nets describe object-centric behavior by using types like a colored Petri net [22].

$\mathcal{B}(A)$ is used to represent all multisets for a set A. Given multiset M for set A, the number of instances of element $a \in A$ in M is $M(a)$. We overload the notation $M = [a^k | a \in A]$ to state that there are k instances of element a in multiset M.

Definition 3 (Object-centric Petri Net [4]). *An object-centric Petri net is a tuple $ON = (N, pt, F_{var})$ where $N = (P, T, F, l)$ is a labeled Petri net with places P and transitions T. $F \in \mathcal{B}((P \times T) \cup (T \times P))$ is the multiset of arcs between places and transitions. Transitions are labeled with activities or τ by $l : T \to \mathbb{U}_{act} \cup \{\tau\}$ with invisible activity $\tau \notin \mathbb{U}_{act}$. $pt : P \to \mathbb{U}_{typ}$ maps places to object types and $F_{var} \leq F$ is the sub-multiset of variable arcs.*

Note that we label all transitions to activities or τ with function l. Other common definitions for object-centric Petri nets define l as a partial function. This can be translated into our definition by assuming $l(t) = \tau$ for all t without a label. We define the following derived notations for object-centric Petri nets

- $\bullet t = \{p \in P | (p, t) \in F\}$ *is the preset of transition* $t \in T$.
- $t\bullet = \{p \in P | (t, p) \in F\}$ *is the post set of transition* $t \in T$.
- $pl(t) = \bullet t \cup t\bullet$ *are the input and output places of* $t \in T$, $pl_{var}(t) = \{p \in P | \{(p, t), (t, p)\} \cap F_{var} \neq \emptyset\}$ *are places that are connected through variable arcs and* $pl_{nv}(t) = \{p \in P | \{(p, t), (t, p)\} \cap (F \setminus F_{var}) \neq \emptyset\}$ *are places that are connected through non-variable arcs.*
- $tpl(t) = \{pt(p) | p \in pl(t)\}$, $tpl_{var}(t) = \{pt(p) | p \in pl_{var}(t)\}$, *and* $tpl_{nv}(t) = \{pt(p) | p \in pl_{nv}(t)\}$ *are object types related to transitions.*

Figure 2 shows the object-centric Petri net for the running example. It has two object types and variable arcs for transitions *receive sample order* and *receive products order*.

Definition 4 (Well-Formed Object-Centric Petri Net [4]). *Let $ON = (N, pt, F_{var})$ be an object-centric Petri net with $N = (P, T, F, l)$. ON is well-formed if for each transition $t \in T : tpl_{var}(t) \cap tpl_{nv}(t) = \emptyset$.*

In a well-formed object-centric Petri net arcs, connected to the same transition and places with the same object type, are either all variable or none of them is. We assume for the following that all the object-centric Petri nets we use are well-formed. Similar to colored Petri nets, object-centric Petri nets use the notion of markings and bindings to describe the semantics of a Petri net.

Definition 5 (Marking of object-centric Petri Net [4]). *Let $ON = (N, pt, F_{var})$ be an object-centric Petri net with $N = (P, T, F, l)$. $\mathcal{Q}_{ON} = \{(p, o) \in P \times \mathbb{U}_{obj} | pt(p) = \pi_{type}(o)\}$ is the set of possible tokens. A marking M of ON is a multiset of tokens $M \in \mathcal{B}(\mathcal{Q}_{ON})$.*

A binding describes which transition fires and what object instances are consumed and produced per object type.

Definition 6 (Binding of object-centric Petri Net [4]). *Let $ON = (N, pt, F_{var})$ be an object-centric Petri net with $N = (P, T, F, l)$. The set of all possible bindings is $B = \{(t, b) \in T \times (\mathbb{U}_{type} \not\to \mathcal{P}(\mathbb{U}_{obj})) | dom(b) = tpl(t) \wedge \forall_{ot \in tpl_{nv}(t)} \forall_{p \in pl_{nv}(t), pt(p) = ot} |b(ot)| = F(p, t)\}$. A binding $(t, b) \in B$ corresponds to firing transition t in Petri net ON. The object map b describes what object instances are consumed and produced. The multiset of consumed tokens given*

binding $(t,b) \in B$ *is* $cons(t,b) = [(p,o) \in \mathcal{Q}_{ON} | p \in \bullet t \wedge o \in b(pt(p))]$. *The multi-set of produced tokens given binding* $(t,b) \in B$ *is* $prod(t,b) = [(p,o) \in \mathcal{Q}_{ON} | p \in t \bullet \wedge o \in b(pt(p))]$.

Binding $(t,b) \in B$ *is enabled in marking* $M \in \mathcal{B}(\mathcal{Q}_{ON})$ *if* $cons(t,b) \leq M$. *Applying binding* (t,b) *in marking* M *leads to new marking* $M' = M - cons(t,b) + prod(t,b)$. *We use the notation* $M \xrightarrow{(t,b)} M'$ *for applying* (t,b) *in* M. *This implies that* (t,b) *was enabled in* M *and* M' *is the result of applying* (t,b) *in* M.

This notation can be extended to a sequence of bindings $\sigma = \langle (t_1,b_1),(t_2,b_2),...,(t_n,b_n) \rangle \in B^*$ *such that* $M_0 \xrightarrow{(t_1,b_1)} M_1 \xrightarrow{(t_2,b_2)} M_2 ... \xrightarrow{(t_n,b_n)} M_n$. *We use the notation* $M \xrightarrow{\sigma} M'$ *to show that* M' *can be reached from* M *by applying the bindings in* σ *after another. The transitions can be mapped to activities using the label function* l. *This results in the visible binding sequence* $\sigma_v = \langle (l(t_1),b_1),(l(t_2),b_2),...,(l(t_n),b_n)) \rangle$ *where* $(l(t_i),b_i)$ *is omitted if* $l(t_i) = \tau$.

Definition 7 (Accepting object-centric Petri Net [4]). *An accepting object-centric Petri net is a tuple* $AN = (ON, M_{init}, M_{final})$ *where* $ON = (N, pt, F_{var})$ *is a well-formed object-centric Petri net.* $M_{init} \in \mathcal{B}(\mathcal{Q}_{ON})$ *and* $M_{final} \in \mathcal{B}(\mathcal{Q}_{ON})$ *indicate the initial and final markings of the net.*

Accepting object-centric Petri nets accept some binding sequences and some not. The set of all binding sequences that are accepted form a language.

Definition 8 (Language of an Accepting Petri Net [4]). *The language* $\phi(AN) = \{\sigma_v | M_{init} \xrightarrow{\sigma} M_{final}\}$ *of an accepting object-centric Petri net* $AN = (ON, M_{init}, M_{final})$ *contains all the visible binding sequences starting in* M_{init} *and ending in* M_{final}.

4 Alignment

Alignments show how process executions and the allowed behavior of a de-jure model relate to each other. We use moves to represent whether something occurs in the process execution, the de-jure model, or both of them. Note that we do not allow alignments to alter the set of objects. We assume them to be fixed.

Definition 9 (Moves). *Let* $L = (E, O, OT, \pi_{act}, \pi_{obj}, \pi_{times}, \pi_{trace})$ *be an object-centric event log and* $P_X = (E_X, D_X) \in px(L)$ *a process execution. Let* $AN = (((P,T,F,l), pt, F_{var}), M_{init}, M_{final})$ *be an accepting object-centric Petri net. The set of all moves is* $moves(P_X, AN) \subseteq (\{\pi_{act}(e) | e \in E_X\} \cup \{\gg\}) \times \mathcal{P}(O) \times (T \cup \{\gg\}) \times \mathcal{P}(O)$ *with skip symbol* $\gg \notin \mathbb{U}_{act} \cup T$. *A move* $(a_{log}, o_{log}, t_{mod}, o_{mod}) \in moves(P_X, AN)$ *is one of the following three types:*
Log move - for an $e \in E_X$*:* $a_{log} = \pi_{act}(e)$, $o_{log} = \pi_{obj}(e)$, $t_{mod} = \gg$, *and* $o_{mod} = \emptyset$.
Model move - for a $(t,b) \in \sigma$ *with* $\sigma_v \in \phi(AN)$*:* $t_{mod} = t$, $o_{mod} = \bigcup_{o \in range(b)} o$, $a_{log} = \gg$, *and* $o_{log} = \emptyset$.
Synchronous move - for an $e \in E_X$ *and a* $(t,b) \in \sigma$ *with* $\sigma_v \in \phi(AN)$*:* $a_{log} = \pi_{act}(e) = l(t)$, $t_{mod} = t$, *and* $o_{log} = o_{mod} = \pi_{obj}(e) = \bigcup_{o \in range(b)} o$.

Each type of move is presented in Figure 3. For synchronous moves, the activity and objects of the model and log part have to be exactly the same. For log and model moves, only one part has an activity and objects while the other parts are skipped. The skip symbol $\gg$ represents that nothing happened in that part. The upper part is the log part a_{log} and o_{log}. The lower block is the model part that contains t_{mod}, the activity $l(t_{mod})$ it is labeled with, and o_{mod}.

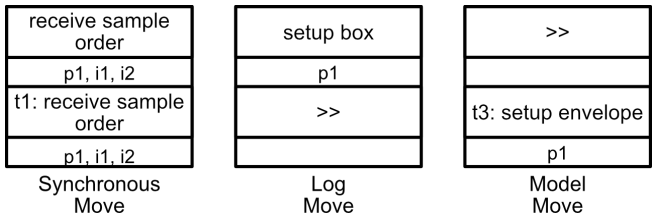


Fig. 3. One synchronous move, one log move, and one model move for the running example with the accepting object-centric Petri net in Figure 2 and the process execution in Figure 1.

We define the following projections on moves.

Definition 10 (Move Projections).
Given a move $m = (a_{log}, o_{log}, t_{mod}, o_{mod}) \in moves(P_X, AN)$ *with process execution* P_X *and accepting object-centric Petri Net* $AN = (((P, T, F, l), pt, F_{var}), M_{init}, M_{final})$. *We use the following projections to map moves to their attributes:*
$\pi_{la}(m) = a_{log}$ *maps moves to their log activity.*
$\pi_{lo}(m) = o_{log}$ *maps moves to their log objects.*
$\pi_{mt}(m) = t_{mod}$ *maps moves to their model transition.*
$\pi_{ma}(m) = l(t_{mod})$ *maps moves to the activity the transition is labeled with.*
$\pi_{mo}(m) = o_{mod}$ *maps moves to their model objects.*

When reasoning about the model or log behavior individually, we want to ignore skipped behavior in that part. An alignment, which we define in Definition 12, is a directed acyclic graph of moves. We need to reason about the model and log behavior individually to define alignments. Therefore, we introduce the following reductions that remove moves with skipped behavior in a given part from a directed acyclic graph of moves while maintaining the partial order defined by the acyclic graph.

Definition 11 (Reduction to Log and Model Part). *Let* $MG = (M, C)$ *be a directed acyclic graph with vertices* $M \subseteq moves(P_X, AN)$ *and edges* $C \subseteq M \times M$ *with process execution* P_X *and accepting object-centric Petri Net* AN. *The reduction to moves with visible activity in the log part is* $MG_{\downarrow log} = (M_{\downarrow log}, C_{\downarrow log})$ *with:*

- $M_{\downarrow log} = \{m \in M | \pi_{la}(m) \neq \gg\}$ *synchronous and log moves.*
- $C_{\downarrow log} = \{(m_1, m_n) \in M_{\downarrow log} \times M_{\downarrow log} | \exists_{<m_1,\ldots,m_n> \in M^*} \forall_{1 \leq i < n} (m_i, m_{i+1}) \in C \wedge \forall_{1<i<n} \pi_{la}(m_i) = \gg\}$ *edges between synchronous and log moves and new edges where model moves were removed.*

The reduction to moves with visible activity in the model part is $MG_{\downarrow mod} = (M_{\downarrow mod}, C_{\downarrow mod})$ *with:*

- $M_{\downarrow mod} = \{m \in M | \pi_{ma}(m) \neq \gg\}$ *synchronous and model moves*
- $C_{\downarrow mod} = \{(m_1, m_n) \in M_{\downarrow mod} \times M_{\downarrow mod} | \exists_{<m_1,\ldots,m_n> \in M^*} \forall_{1 \leq i < n} (m_i, m_{i+1}) \in C \wedge \forall_{1<i<n} \pi_{ma}(m_i) = \gg\}$ *edges between synchronous and model moves and new edges where log moves were removed.*

In Figure 4 both $MG_{\downarrow log}$ and $MG_{\downarrow model}$ are visualized for an arbitrary directed acyclic graph of moves. $MG_{\downarrow log}$ describes a directed acyclic graph after removing all model moves and related edges. New edges are added when two movements used to be connected via removed model moves in the movement graph. $MG_{\downarrow model}$ behaves simultaneously for the model part.

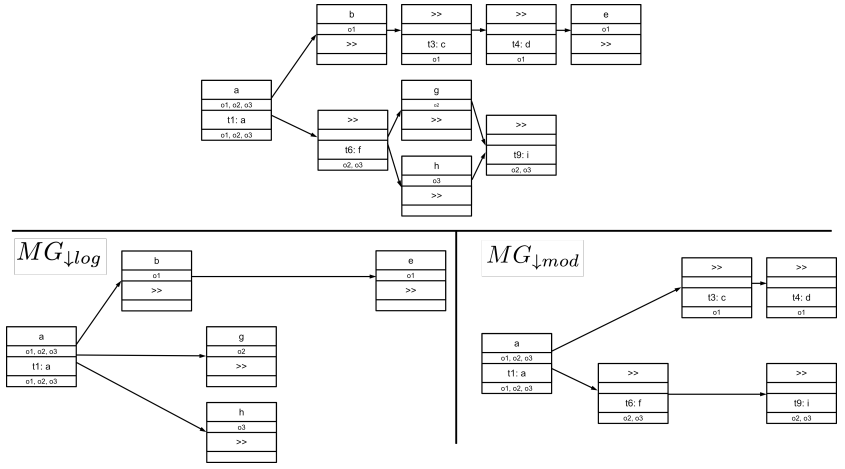


Fig. 4. Reductions $MG_{\downarrow log}$ and $MG_{\downarrow model}$ for a directed acyclic graph of moves.

An alignment is a directed acyclic graph of moves that requires the log part to contain the process execution and the model part to be in the language of the de-jure model.

Definition 12 (Alignment). *Let* $L = (E, O, OT, \pi_{act}, \pi_{obj}, \pi_{times}, \pi_{trace})$ *be an object-centric event log and* $P_X = (E_X, D_X) \in px(L)$ *a process execution. Let* $AN = (((P, T, F, l), pt, F_{var}) M_{init}, M_{final})$ *be an accepting object-centric Petri net. An alignment* $AL_{P_X,AN} = (M, C)$ *is a directed acyclic graph on* $M \subseteq moves(P_X, AN)$ *such that:*

The alignment contains the process execution behavior in the log parts: P_X is isomorphic to $AL_{P_X,AN_{\downarrow log}}$ with bijective function $f : E_X \to M_{\downarrow log}$ such that $\forall_{e \in E_X} \pi_{act}(e) = \pi_{la}(f(e)) \wedge \pi_{obj}(e) = \pi_{lo}(f(e))$.

The alignment contains behavior that is accepted by the Petri net in the model parts: There exists a binding sequence $\sigma = \langle (t_1, b_1), (t_2, b_2), ..., (t_n, b_n) \rangle \in B^*$ with $\sigma_v \in \phi(AN)$ and a bijective function $f' : B \to M_{\downarrow mod}$ such that:

- $\forall_{(t,b) \in \sigma} t = \pi_{mt}(f'(t,b)) \wedge \bigcup_{o \in range(b)} o = \pi_{mo}(f'(t,b))$
- $\forall_{m_1,m_2 \in M_{\downarrow mod}} (m_1, m_2) \in C_{\downarrow mod} \Rightarrow \exists_{1 \leq i < j \leq n} m_1 = f'(t_i, b_i) \wedge m_2 = f'(t_j, b_j)$

There can be multiple alignments for a process execution and an accepting object-centric Petri net. $al(P_X, AN)$ is the set of all these alignments.

An alignment for the running example can be seen in Figure 5. The graph of moves is directed and acyclic which creates a partial order of moves. When reducing the graph to log and synchronous moves, it is isomorphic to the given process execution in Figure 1. This ensures that the behavior of the given process execution is contained in the alignment. The reduction to the model part relates to a binding sequence that has to be in the language of the accepting object-centric Petri net. This ensures that the model part describes behavior that is accepted by the model. Note that an alignment does not put any other requirement on the model part than to be allowed by the model. Therefore, alignments can also contain model behavior that is very different from the process execution. Those alignments will end up with more model and log moves and fewer synchronous moves. Cost functions for moves allow us to search for special behavior from the model that we want to align. Most of the time we are looking for the allowed behavior that is the most similar to the given process execution. That is what the standard cost function formalizes.

Definition 13 (Standard Cost of Move Function).
Let $AL_{P_X,AN} = (M, C) \in al(P_X, AN)$ be an alignment with process execution P_X and accepting object-centric Petri Net AN. The cost function $cost_{move} : moves(P_X, AN) \to \mathbb{R}$ is defined as:

$$cost_{move}(m) = \begin{cases} 0 \text{ if } m \text{ is a synchronous move,} \\ |\pi_{lo}(m) \cup \pi_{mo}(m)| \text{ if } m \text{ is a model or log move} \wedge \pi_{ma}(m) \neq \tau, \\ \varepsilon \text{ if } \pi_{ma}(m) = \tau \wedge a_{log} = \gg, \\ +\infty \text{ else} \end{cases}$$

With ε being a positive very small number. The cost of a complete alignment is the sum over all alignment moves: $cost_{alignment}(AL_{P_X,AN}) = \sum_{m \in M} cost_{move}(m)$.

The cost of the alignment in Figure 5 is 6 because there are 3 model and 3 log moves that have one object each. The lower the cost the better model and log part match, because synchronous moves are cheaper than model and log moves. We call one of the cheapest alignments an optimal alignment.

Definition 14 (Optimal Alignment). *Let L be an object-centric event log and $P_X \in px(L)$ be a process execution. Let AN be an accepting object-centric*

Petri net. An alignment $AL_{P_X,AN} = (M, C) \in al(P_X, AN)$ *is optimal if* $\forall_{a \in al(P_X,AN)} cost_{alignment}(AL_{P_X,AN}) \leq cost_{alignment}(a)$.

Note that there can still be multiple optimal alignments for a given process execution and a de-jure Petri net as long as they are all equally similar to the process execution.

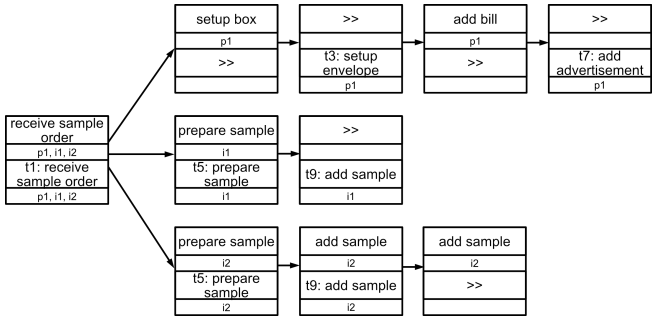


Fig. 5. Alignment for process execution in Figure 1 and accepting object-centric Petri net in Figure 2.

An optimal object-centric alignment for the running example can be seen in Figure 5. The inter-object dependencies that are defined in the object-centric Petri net in Figure 2 are respected by the object-centric alignment. For example, the object-centric alignment agreed on one shared start activity for *p1*, *i1*, and*i2* which keeps the alignment consistent with inter-object dependencies. This differentiates object-centric alignments and traditional alignments which can violate this requirement.

5 Object-Centric Synchronous Product Net

The first part of our approach to calculate optimal alignments is to create a synchronous product net for a given process execution and an accepting object-centric Petri net. That synchronous product net is designed to generate all possible alignments. Since alignments can have model, log, and synchronous moves, the synchronous product net consists of three parts that directly relate to them. In the synchronous product net in Figure 8 these parts are marked. First, we construct the log part from the process execution. Then, we pre-process the de-jure model to finally merge them together to the synchronous product net and add the synchronous part.

5.1 Process Execution Net Construction

The process execution net will be the part of the synchronous product net that guarantees that the process execution is contained in the alignment. The construction of the process execution net relates to Petri net runs [15] and causal

nets [3]. For each object, each edge in the process execution defines a precondition for an event. The process execution net in Figure 6, therefore, has a place for each condition defined in the process execution.

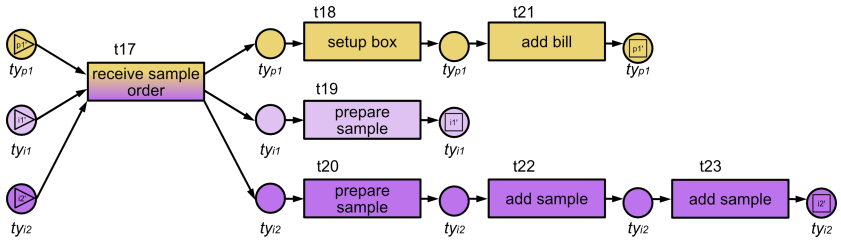


Fig. 6. Process Execution Net

Because object-centric Petri nets are under-specified and we will merge the created net with a given accepting object-centric Petri net, we can not use the same objects and types for both, the log and model part. The process execution contains *directly follows* relations on the level of object instances. For example, the running example process execution in Figure 1 shows that on item *i1* *add sample* never happens, whereas on item *i2* activity *add sample* happens twice. Although they both have the same object type, the process execution differentiates them. An accepting object-centric Petri net is under-specified in that regard because it only differentiates places by type. Clearly, it is important to separate the object instances because otherwise, deviations in one object instance could compensate for a contrary deviation in another. For example, the missing *add sample* activity on *i1* could be compensated by the additional *add sample* activity on *i2* if one would not strictly separate which event happened on which object instance. To enable an accepting object-centric Petri net to separate object instances, we have to create a new individual type for each object instance in the process execution. Since each object instance must have exactly one type, we then need to replace the original object instances with new individual placeholder object instances in the process execution net.

Definition 15 (Generation of new Objects and Types). *Let* $L = (E, O, OT, \pi_{act}, \pi_{obj}, \pi_{times}, \pi_{trace})$ *be an object-centric event log and* $P_X = (E_X, D_X) \in px(L)$ *a process execution containing objects* X*. Let* $O_{new}(P_X) \in \mathbb{U}_{obj}$ *with* $O_{new}(P_X) \cap O = \emptyset$ *and* $|O_{new}(P_X)| = |X|$ *be a set of new object instances and* $OT_{new}(P_X) \in \mathbb{U}_{type}$ *with* $OT_{new}(P_X) \cap \{\pi_{type}(o) | o \in X\} = \emptyset$ *and* $|OT_{new}(P_X)| = |X|$ *be a set of new types.*

Let $new_{obj} : X \to O_{new}(P_X)$ *be a bijective function that renames objects from* X *to unused objects in* $O_{new}(P_X)$*. Let* $new_{type} : X \to OT_{new}(P_X)$ *be a bijective function that relates each object to its unique new type.*

We define the following derivative concepts:
$orob : O_{new}(P_X) \to X$ *with* $orob = new_{obj}^{-1}$ *returns the original object.*

orty : $OT_{new}(P_X) \rightarrow \{\pi_{type}(o)|o \in X\}$ *with* $orty(ot) = \pi_{type}(new^{-1}_{type}(ot))$ *returns the original type of the object that is associated with the given new type.*

We create the process execution net with the new objects and types. The conditions defined in the process execution are the *directly follows* relation per object. For each condition, the Petri net contains a place. Also, start and end places are added for each object. The transitions relate to the events of the process execution.

Definition 16 (Process Execution Net). *Let* $L = (E, O, OT, \pi_{act}, \pi_{obj}, \pi_{time}, \pi_{trace})$ *be an object-centric event log. Let* $P_X = (E_X, D_X) \in px(L)$ *be a process execution from the event log. The process execution net* $PX_{net} = (((P, T, F, l), pt, F_{var}), M_{init}, M_{final})$ *is an accepting object-centric Petri net with:*

- $P = \{p_{o;i}|o \in X \wedge \pi_{trace}(o) = \langle e_1, ..., e_n\rangle \wedge 1 \leq i \leq n-1\}$
  $\cup \{p_{o;s}|o \in X\} \cup \{p_{o;e}|o \in X\}$
- $T = \{t_e|e \in E_X\}$
- $F = [(t_e, p_{o;i}) \in T \times P|\pi_{trace}(o) = \langle e_1, ..., e_n\rangle \wedge \exists_{1 \leq i \leq n-1} e_i = e]$
  $\cup\, [(p_{o;i}, t_e) \in P \times T|\pi_{trace}(o) = \langle e_1, ..., e_n\rangle \wedge \exists_{1 \leq i \leq n-1} e_{i+1} = e]$
  $\cup\, [(p_{o;s}, t_e) \in P \times T|\pi_{trace}(o) = \langle e_1, ..., e_n\rangle \wedge e_1 = e]$
  $\cup\, [(t_e, p_{o;e}) \in T \times P|\pi_{trace}(o) = \langle e_1, ..., e_n\rangle \wedge e_n = e]$
- $l(t_e) = \pi_{act}(e)$; $pt(p_{o;i}) = new_{type}(o)$; $F_{var} = \emptyset$
- $M_{init} = \{(p_{o;s}, new_{obj}(o))|o \in X\}$; $M_{final} = \{(p_{o;e}, o')|new_{obj}(o) \in X\}$

5.2 Pre-processing of the Object-Centric Petri Net

The de-jure behavior is already given as an accepting object-centric Petri net, which is very close to what we need to create the synchronous product net. But an accepting object-centric Petri net can have variable arcs. The Petri net of the running example in Figure 2 has variable arcs for t_1 and t_2. For transitions with variable arcs, we do not know beforehand how many objects they will use. This becomes a problem when we try to find synchronous moves between the process execution net and the de-jure net. A transition can only be synchronous if they use the same object instances which implies that they use the same number of object instances. Therefore we need to know beforehand how many object instances a transition will use.

As mentioned in Section 4, we assume that the set of objects for the alignment is immutable and defined by the process execution. For a predefined set of object instances, the number of objects a variable arc can use is finite. Therefore, we can replace transitions with variable arcs with a set of transitions without variable arcs. For each combination of how many objects a variable arc could consume we can add a new transition to the Petri net that uses exactly that number of objects, but by replacing the variable arc with a number of non-variable arcs. When doing this for the Petri net in Figure 2 the result will be the Petri net without variable arcs in Figure 7.

Definition 17 (Pre-processing of Accepting Object Centric Petri Net). *Let* $L = (E, O, OT, \pi_{act}, \pi_{obj}, \pi_{times}, \pi_{trace})$ *be an object-centric event log and* $P_X = (E_X, D_X) \in px(L)$ *a process execution. Let* $AN = (((P, T, F, l), pt, F_{var}), M_{init}, M_{final})$ *be an accepting object-centric Petri net. Let there be an arbitrary but fixed order of types* $ot_1, ..., ot_n \in OT$*. The pre-processed accepting object-centric Petri net is* $DJ_{net} = (((P', T', F', l'), pt', F'_{var})M'_{init}, M'_{final})$ *with:*

- $P' = P$, $pt' = pt, F'_{var} = \emptyset$,
- $T' = \{t_{c_1,...,c_n} | t \in T \wedge \forall_{1 \leq i \leq n} 0 \leq c_i \leq |\{o \in X | \pi_{type}(o) = ot_i\}|$ *if* $ot_i \in tpl_{var}(t)$ *otherwise* $c_i = 0\}$
- $F' = [(p, t_{c_1,...,c_n}) \in P' \times T' | (p, t) \in F \wedge (p, t) \notin F_{var})]$
  $\cup [(t_{c_1,...,c_n}, p) \in T' \times P' | (t, p) \in F \wedge (t, p) \notin F_{var})]$
  $\cup [(p, t_{c_1,...,c_n})^k \in P' \times T' | (p, t) \in F_{var} \wedge \exists_{1 \leq i \leq n} pt(p) = ot_i \wedge c_i = k]$
  $\cup [(t_{c_1,...,c_n}, p)^k \in T' \times P' | (t, p) \in F_{var} \wedge \exists_{1 \leq i \leq n} pt(p) = ot_i \wedge c_i = k]$
- $l'(t_{c_1,...,c_n}) = l(t)$
- $M'_{init} = [(p, o) \in P' \times X | \forall_{t \in T} (t, p) \notin F \wedge pt(p) = \pi_{type}(o)]$
- $M'_{final} = [(p, o) \in P' \times X | \forall_{t \in T} (p, t) \notin F \wedge pt(p) = \pi_{type}(o)]$

We call the pre-processed accepting object-centric Petri net a de-jure net. The de-jure net of the running example can be seen in Figure 7.

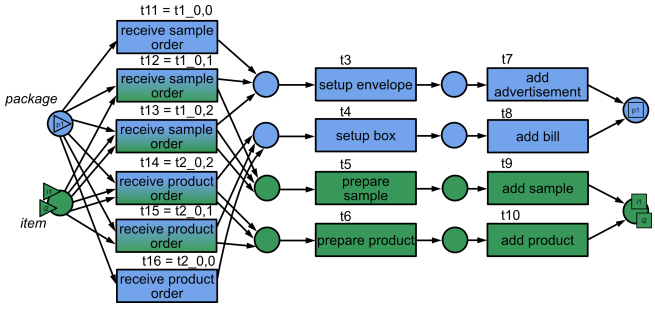


Fig. 7. Preprocessed accepting object-centric Petri net without variable arcs

5.3 Creating the Synchronous Product Net

Now that we have two accepting object-centric Petri nets without variable arcs, one describing the process execution, and one describing the de-jure behavior, we can add them together to create the synchronous product net. The synchronous product net should directly relate to possible alignments. In an alignment, either the process execution and the de-jure parts advance forward separately or they advance forward synchronously. For activities to happen in the process execution and the de-jure part simultaneously, they have to be the same activity on the same object instances. Object-centric Petri nets can only separate between object

types and not between object instances. Therefore, we extend the object-centric Petri net with ν-net requirements that can differentiate between object instances. We decode log, model, and synchronous transitions with $(t_{log}, \gg)$, $(\gg, t_{mod})$, and (t_{log}, t_{mod}) as one can see in the synchronous product net in Figure 8.

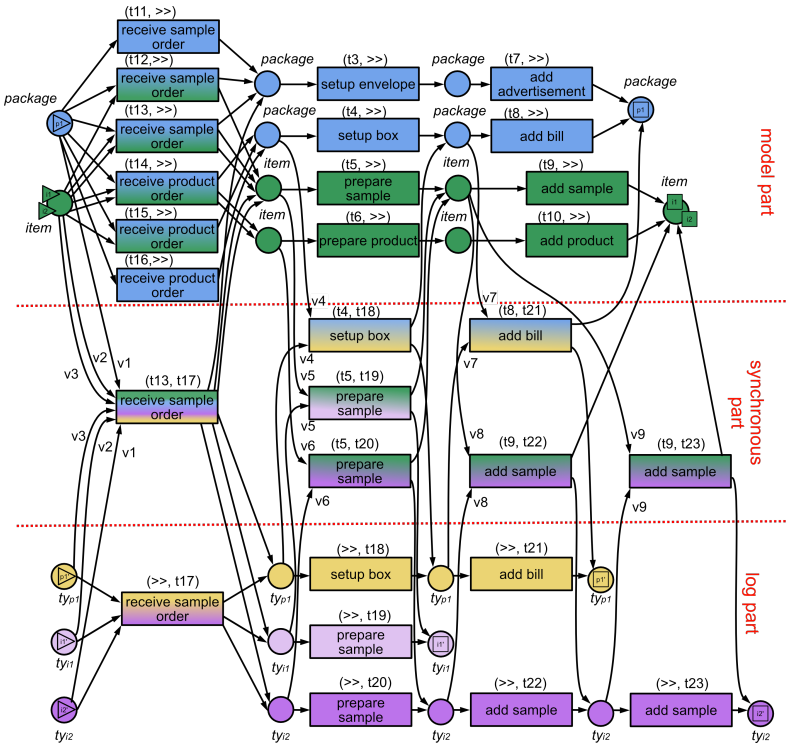


Fig. 8. Synchronous product net for the process execution net in Figure 6 and accepting object-centric Petri net in Figure 7

Definition 18 (Synchronous Product Net).
Let $PX_{net} = (((P_{PX}, T_{PX}, F_{PX}, l_{PX}), pt_{PX}, F_{var_{PX}} = \emptyset), M_{init_{PX}}, M_{final_{PX}})$ *be a process execution net. Let* $DJ_{net} = (((P_{DJ}, T_{DJ}, F_{DJ}, l_{DJ}), pt_{DJ}, F_{var_{DJ}} = \emptyset), M_{init_{DJ}}, M_{final_{DJ}}$ *be a de-jure net. Let* Var *be a set of unused variable names.*

Two transitions can be synchronous if they are labeled with the same activity and consume the same multiset of tokens. This requires them to be related to the same types and consume and produce the same number of tokens.
$T_{syn} = \{(t_{PX}, t_{DJ}) \in T_{PX} \times T_{DJ} | l_{PX}(t_{PX}) = l_{DJ}(t_{DJ}) \wedge \{orty(ty) | ty \in tpl(t_{PX})\} = tpl(t_{DJ}) \wedge \forall_{p_{PX} \in \bullet t_{PX}, p_{DJ} \in \bullet t_{DJ}} (orty(pt_{PX}(p_{PX})) = pt_{DJ}(p_{DJ})) \Rightarrow F_{PX}((p_{PX}, t_{PX})) = F_{DJ}((p_{DJ}, t_{DJ}))) \wedge \forall_{p_{PX} \in t_{PX}\bullet, p_{DJ} \in t_{DJ}\bullet} (orty(pt_{PX}(p_{PX})) = pt_{DJ}(p_{DJ})) \Rightarrow (F_{PX}((t_{PX}, p_{PX})) = F_{DJ}((t_{DJ}, p_{DJ})))\}$ *is the set of synchronous transitions.*

The synchronous product net is the accepting object-centric ν-Petri net $SP_{net} = (ON_{SP}, M_{init_{SP}}, M_{final_{SP}}, \nu)$ with $ON_{SP} = (N_{SP}, pt_{SP}, F_{var_{SP}} = \emptyset)$ with $N_{SP} = (P_{SP}, T_{SP}, F_{SP}, l_{SP})$ such that:

- $P_{SP} = P_{PX} \cup P_{DJ}$ *is the union of places and* $pt_{SP} = pt_{PX} \cup pt_{DJ}$
- $T_{SP} \subset (T_{PX}^{\gg} \times T_{DJ}^{\gg}) = \{(t_{PX}, \gg) | t_{PX} \in T_{PX}\} \cup \{(\gg, t_{DJ}) | t_{DJ} \in T_{DJ}\} \cup T_{syn}$ *is the union of transitions with additional synchronous transitions.*
- $F_{SP} = [(p_{PX}, (t_{PX}, \gg)) | (p_{PX}, t_{PX}) \in F_{PX}] \cup [(p_{DJ}, (\gg, t_{DJ})) | (p_{DJ}, t_{DJ}) \in F_{DJ}] \cup [((t_{PX}, \gg), p_{PX}) | (t_{PX}, p_{PX}) \in F_{PX}] \cup [((\gg, t_{DJ}), p_{DJ}) | (t_{DJ}, p_{DJ}) \in F_{DJ}] \cup [(p_{PX}, (t_{PX}, t_{DJ})) | (p_{PX}, t_{PX}) \in F_{PX} \wedge (t_{PX}, t_{DJ}) \in T_{SP}] \cup [(p_{DJ}, (t_{PX}, t_{DJ})) | (p_{DJ}, t_{DJ}) \in F_{DJ} \wedge (t_{PX}, t_{DJ}) \in T_{SP}] \cup [((t_{PX}, t_{DJ}), p_{PX}) | (t_{PX}, p_{PX}) \in F_{PX} \wedge (t_{PX}, t_{DJ}) \in T_{SP}] \cup [((t_{PX}, t_{DJ}), p_{DJ}) | (t_{DJ}, p_{DJ}) \in F_{DJ} \wedge (t_{PX}, t_{DJ}) \in T_{SP}]$
- $l_{SP} = \{((t_{PX}, \gg), a) | (t_{PX}, a) \in l_{PX}\} \cup \{(\gg, t_{DJ}), a) | (t_{DJ}, a) \in l_{DJ}\} \cup \{((t_{PX}, t_{DJ}), a) | (t_{PX}, a) \in l_{PX}, (t_{DJ}, a) \in l_{DJ}\}$
- $M_{init_{SP}} = M_{init_{PX}} \cup M_{init_{DJ}}$ *and* $M_{final_{SP}} = M_{final_{PX}} \cup M_{final_{DJ}}$
- $\nu : P_{SP} \times T_{SP} \not\rightarrow Var$ *such that:*
  $\forall_{(t_{PX}, t_{DJ}) \in T_{PX} \times T_{DJ}} \forall_{p_{PX} \in \bullet(t_{PX}, t_{DJ})} \exists_{(p_{DJ}, (t_{PX}, t_{DJ})) \in F_{SP}}$
  $(p_{DJ} \in P_{DJ} \wedge pt(p_{DJ}) = orty(pt(p_{PX}))) \Rightarrow$
  $(\nu((p_{DJ}, (t_P X, t_{DJ}))) = \nu((p_{PX}, (t_{PX}, t_{DJ}))))$

In the process execution part, we use the *orob* function when comparing objects from the model and log part. The ν net requirement function assigns variables to the in-going arcs of the synchronous transition. For each in-going arc from the process execution net, there has to be one arc from every place that has the same original type, so that those arcs have the same unique variable assigned by Var. This requires the synchronous move to use the same object instance in the process execution net and the de-jure net. The variables ensure that transitions can only consume the same object instance for arcs with the same variable. This refers to the original objects not the newly created ones in the process execution net.

Definition 19 (Valid Binding in Synchronous Product Net). *Let $SP_{net} = (((P_{SP}, T_{SP}, F_{SP}, l_{SP}), pt_{SP}, F_{var_{SP}}), M_{init_{SP}}, M_{final_{SP}}, \nu)$ be the synchronous product net. Let $SP_{net \setminus \nu}$ be the synchronous product net without ν. A binding sequence $\sigma = \langle (t_1, b_2), ..., (t_n, b_n) \rangle \in B^*$ is valid for SP_{net} if it is valid for $SP_{net \setminus \nu}$ and for every $(t_i, b_i) \in \sigma$ with $t_i \in T_{sny}$ that is connected to an arc with a variable assigned by ν it holds that arcs with the same variable consume the same object:*
$\forall_{p_{DJ} \in P_{DJ}, p_{PX} \in P_{PX}, \{p_{DJ}, p_{PX}\} \subseteq \bullet(t_i)} (\nu((p_{PX}, t_i)) = \nu((p_{DJ}, t_i)) \Rightarrow \{orob(o) | o \in b_i(pt((p_{PX}, t_i)))\} \cap b(pt((p_{DJ}, t_i))) \neq \emptyset)$.

The resulting synchronous Petri net for the running example can be seen in Figure 8.

6 Alignments from Synchronous Product Net

This section describes the second part of our approach to finding object-centric alignments. As an input for this part, we have a synchronous product net in which every transition relates to a move of an alignment. All binding sequences from the initial marking to the final marking in the synchronous product net relate to an alignment. In this section, we now want to find an optimal alignment given a cost function for moves. Every binding in the synchronous product net directly relates to a move. A binding defines the transition and the used objects. As described in Section 5 the transitions already describe the type of move with $(\gg, t)$ being a model move, $(t, \gg)$ being a log move, and (t_{PX}, t_{DJ}) being a synchronous move. The activity is also defined by the transition from the binding. The objects used in the binding are the objects of the move.

Therefore, searching for an optimal alignment relates to searching for an optimal binding sequence from the initial to the final marking of the synchronous product net. Interpreting markings as nodes and bindings as edges between markings, we can set up the search space as a graph. The cost of the edges is then the given cost function applied to the move that relates to the binding. This is a well-defined search problem we can solve with standard search algorithms for finding the cheapest or shortest path in a graph.

Definition 20 (Weighted State Space Graph).
Let $SP_{net} = (((P_{SP}, T_{SP}, F_{SP}, l_{SP}), pt_{SP}, F_{var_{SP}}), M_{init_{SP}}, M_{final_{SP}}, \nu)$ *be a synchronous product net. Let* B_{SP} *be all possible bindings in* SP_{net}*. The weighted search space graph for that Petri net is* $G_{wg} = (V_{wg}, E_{wg}, w_{wg})$ *with* $V_{wg} \subseteq \mathcal{B}(\mathcal{Q}_{ON_{SP}})$*,* $E_{wg} \subseteq \mathcal{B}(\mathcal{Q}_{ON_{SP}}) \times \mathcal{B}(\mathcal{Q}_{ON_{SP}})$ *and* $w_{wg} : E_{wg} \to \mathbb{R}$ *such that:*

- $V_{wg} = \{M \in \mathcal{B}(\mathcal{Q}_{ON_{SP}}) | \exists_{\sigma \in B_{SP}^*} M_{init_{SP}} \xrightarrow{\sigma} M\}$ *is the set of all reachable markings in the synchronous product net.*
- $E_{wg} = \{(M, M') \in \mathcal{B}(\mathcal{Q}_{ON_{SP}}) \times \mathcal{B}(\mathcal{Q}_{ON_{SP}}) | \exists_{(t,b)\ in B_{SP}} M \xrightarrow{(t,b)} M'\}$ *connects each marking to its directly reachable markings.*
- $w_{wg}(M, M') = cost_{move}(a_{log}, o_{log}, t_{model}, o_{model})$ *where* $(t, b) \in B_{SP}$ *with* $M \xrightarrow{(t,b)} M'$ *and* $t = (t_{PX}, t_{DJ}) \in T_{SP}$ *so that:*
  - $a_{log} = l_{SP}(t_{PX})$*;* $t_{model} = t_{DJ}$
  - $o_{log} = range(b)$ *if* $a_{log} \neq \gg$ *else* $o_{log} = \emptyset$
  - $o_{model} = range(b)$ *if* $a_{model} \neq \gg$ *else* $o_{model} = \emptyset$

Theorem 1. *Let* P_X *be a process execution with a finite number of objects and events and a let* $DJ_{net} = (((P_{DJ}, T_{DJ}, F_{DJ}, l_{DJ}), pt_{DJ}, F_{var_{DJ}} = \emptyset), M_{init_{DJ}}, M_{final_{DJ}})$ *be a de-jure net with a finite number of reachable markings from the initial marking. The synchronous product net* SP_{net} *for* P_X *and* DJ_{net} *has a finite number of reachable markings.*

Proof (Finite Search Space). Let $P_X = (E_X, D_X)$ be a process execution with a finite number of objects $o = |X|$ and events $e = |E_X|$. Let $DJ_{net} = (((P_{DJ}, T_{DJ},$

$F_{DJ}, l_{DJ}), pt_{DJ}, F_{var_{DJ}} = \emptyset), M_{init_{DJ}}, M_{final_{DJ}})$ be a de-jure model for the process execution with a finite number of reachable markings $m_{DJ} = |\{M | \exists_{\sigma \in B^*_{DJ}} M_{init_{DJ}(PX)} \xrightarrow{\sigma} M\}|$ with B_{DJ} being the set of possible bindings in DJ_{net}.

Step 1 of our method creates the process execution net $PX_{net} = (((P_{PX}, T_{PX}, F_{PX}, l_{PX}), pt_{PX}, F_{var_{PX}}), M_{init_{PX}}, M_{final_{PX}})$ from the process execution. Let B_{PX} be the set of all bindings in PX_{net}. At most one place is added per event for each object together with one additional start place for each object. Since the number of events and objects is finite the number of places in PX_{net} is finite. Each transition produces as many tokens as it consumes because it is connected to one output place for each input place. So the number of tokens stays consistent all the time. Therefore, the number of reachable markings $m_{PX} = |\{M | \exists_{\sigma \in B^*_{PX}} M_{init_{PX}} \xrightarrow{\sigma} M\}| \leq (e+1)^o$ in PX_{net} is finite.

Step 2 of our method preprocesses DJ_{net}. Thereby, no places are added. Transitions with variable arcs are replaced by a set of transitions that can only reach markings the original transition was able to reach as well. Thus, the number of reachable markings stays the same and is therefore finite.

In Step 3, DJ_{net} and PX_{net} are merged together to form the synchronous product net. No places are added and the synchronous transitions that are added, do not add new reachable markings because there always exists one transition from DJ_{net} and one from PX_{net} that together have the same effect on the markings of the net. Therefore, the number of markings in the synchronous product net SP_{net} for P_X and the preprocessed DJ_{net} is the product of the two nets: $m_{SP} = |\{M | \exists_{\sigma \in B^*_{SP}} M_{init_{SP}} \xrightarrow{\sigma} M\}| = m_{DJ} * m_{PX}$ with B_{SP} being the set of all possible bindings in SP_{net}. Since m_{DJ} and m_{PX} are finite, the synchronous product net has a finite number of reachable markings.

As stated in Theorem 1 and shown in the finite search space proof, the weighted search space graph is finite given that the event log and the de-jure model have a finite size. For a finite weighted graph, it is decidable whether a cheapest path exists, and if one exists, algorithms like the Dijkstra algorithm [14] can find them. The only case in which there is no cheapest path is when there is no path at all. This means that the synchronous product net cannot reach the final state from the initial state. This can only occur if the de-jure model has no option to complete for the set of objects of the process execution. In other words, the de-jure model does not contain any allowed behavior related to the process execution. In this case, there can be no alignment and the user is informed that the de-jure model does not match the process execution. This is similar to traditional alignments with a de-jure model that has no option to complete. In the normal case where the de-jure model describes behavior related to the given process execution, an optimal alignment is determined with the help of the shortest path.

The resulting shortest path is a binding sequence from the initial to the final marking. It relates directly to an object-centric alignment for the given process execution and the de-jure model. Using the synchronous product net in Figure 8 the search for an optimal alignment results in the alignment in Figure 5. Thereby respecting the inter-object dependencies between *package* and *item*. Also, object

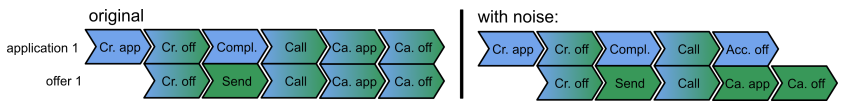


Fig. 9. A process execution from [17] and a variation of it that has some noise.

instances are correctly separated by the ν-net requirements, so that deviations of different object instances can not compensate another. That can be seen in Figure 5 where the missing *add sample* activity for *i1* and the additional *add sample* activity for *i2* are both identified as deviations.

There can be multiple binding sequences with the same cost. It is guaranteed that one of the cheapest is found, but if there are multiple binding sequences with the same cost, it depends on the implementation which one is found.

7 Evaluation

We conducted an evaluation with real-world data from the BPI2017 challenge [17]. The evaluation is split into a qualitative part and a quantitative evaluation of the run time.

7.1 Qualitative

The purpose of the qualitative evaluation is to evaluate whether the proposed approach does indeed give better insights into complex processes than existing approaches. In the following, we will compare the presented object-centric alignment approach to traditional alignments [8] on a flattened event log [4].

We used data from BPI2017 [17] for our real-world evaluation. It is one of the few publicly available event logs that can be turned into an object-centric event log. The two object types of that process are *application* and *offer*. All variants have exactly one application and at least one offer. An application can be canceled or an offer is accepted.

For this evaluation, we selected only the 4 most dominant variants that made up 19.6% of process executions. For simplicity and clarity, we removed the activities *Submit*, *Complete*, and *Accept* because they only appear in the beginning before the processes for *application* and *offer* interact. We discovered the de-jure Petri net from those 4 variants using the python library ocpa [6] which implements the discovery approach from van der Aalst and Berti [4]. The implementation of our approach also uses ocpa [6]. In the next step, we introduced noise in the log by removing and replacing events in the given process execution.

The original accepted process execution and the process execution with noise can be seen in Figure 9. In this example, the activity *Cancel application* is removed from the application trace, as well as the activity *Cancel offer*. Instead of *Cancel offer* the activity *Accept offer* was recorded for *application 1*. In the trace of *offer 1*, nothing was changed. An error like that could be introduced

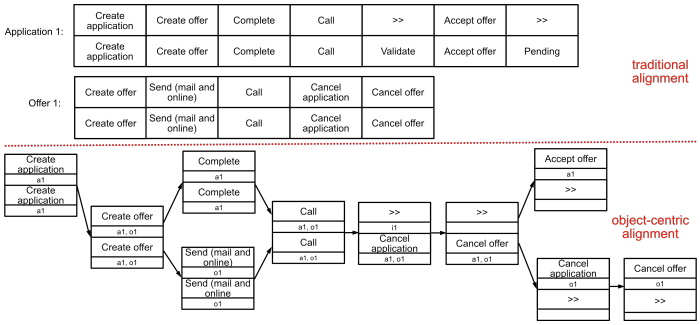


Fig. 10. Traditional and object-centric alignments for the variant with noise and the evaluation de-jure model.

to a real process by human mistakes. The change created multiple deviations from the wanted behavior described by the de-jure model. For example, it is not possible to accept an application without accepting an offer. This is an inter-object dependency we would want alignments to detect. Also, the events *Cancel application* and *Cancel offer* are now only recorded for an *offer* but are not connected with an *application*. Therefore, this represents unwanted behavior as well.

We applied the traditional alignment approach [8] after flattening [4] the object-centric process execution and the object-centric Petri net. This results in the two alignments. We also applied our object-centric alignment approach to the described evaluation data. Both the traditional and object-centric alignment can be seen in Figure 10.

Comparing those two, we can see big differences in the diagnostics. The biggest difference is that the traditional alignment did not detect that *application 1* should not have been accepted. The reason for that is that with the traditional approach, the alignment for *application 1* is computed without any knowledge about any offers. Therefore, it has to assume that activity *Accept Offer* was a valid activity, where instead there was no matching offer to accept. This missing information leads to the traditional alignment suggesting that activities *Validate* and *Pending* are missing in the log, further supporting the understanding that this application should have been accepted. Our approach considers all dependencies between objects and therefore identifies *Accept offer* as a log move. Moreover, the traditional alignment does not indicate any control flow deviation for *offer 1*. This creates a contradiction between the alignment of *offer 1* and *application 1* because an application can not be accepted and canceled. Contradictions like that can not occur in an object-centric alignment because the whole process execution is considered at once.

7.2 Quantitative

To evaluate the scalability of our approach, we performed a quantitative analysis of the run time.

Evaluation Setup As the data source we used BPI2017 event data [17]. Only the most frequent 50% of activities were used. All other activities were filtered out. Afterward, the log consisted of 755 variants. We used a Petri net designed so that the given log contains some dis-aligned process executions. The used Petri net is available on GitHub[1]. It has 6 visible transitions and 4 silent transitions. There are 10 places in the net. We aligned all the 755 variants with the model and tracked their properties and the resulting alignment calculation time. The raw results of that evaluation can be found at GitHub[1]. The evaluation was performed on a 3.1 GHz Dual Core Intel Core i5 with 8GB of RAM.

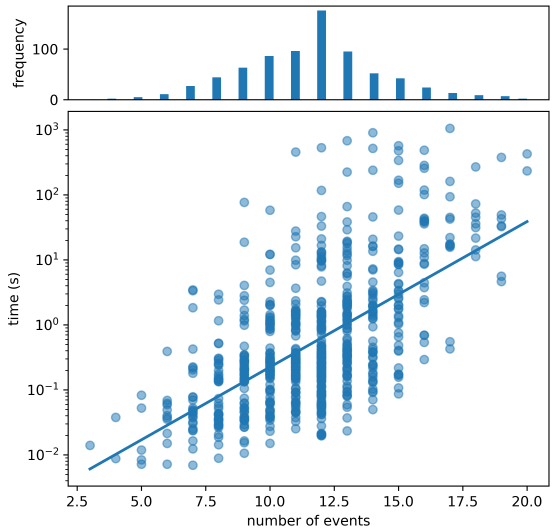


Fig. 11. Execution time on a logarithmic scale over the number of events.

Results Within the 755 calculated alignments, the cost varied from 0 to a maximum of 5. The number of events spanned 3 events up to 20 events with the majority of process execution having 11 to 14 events. The object instances involved started with 2 and went up to 7 instances. The distribution of these attributes can be seen in Figure 11, Figure 12, and Figure 13. The shortest calculation time was 0.007 seconds and the longest was 1051.8 seconds. One can

see that near the limits of the value range for all attributes, there are fewer data points, which makes the results for these values less resistant to outliers. We computed the correlation coefficient for each pair of attributes. The number of events and the number of objects show a positive correlation of 0.59 whereas the number of events and the cost show a negative correlation of -0.38. Especially the latter one is surprising because one would expect more deviations and therefore a higher cost for process variants with more events.

We plotted the calculation time over the three dimensions *number of events* (Figure 11), *number of objects* (Figure 12), and cost (Figure 13). To reduce the effect of the negative correlation, we grouped the data points in Figure 13 by their number of events. Note that the time scale for the time is logarithmic in all three plots and, therefore, the linear increase represents exponential growth. For all three dimensions, the calculation time grows exponentially.

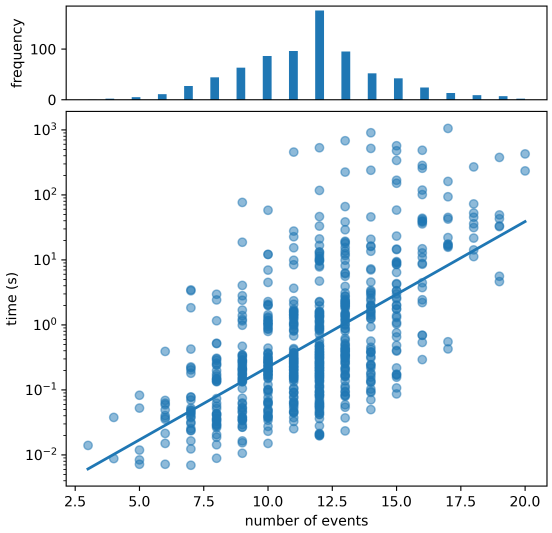


Fig. 12. Execution time on a logarithmic scale over the number of objects.

This result can be explained by the structure of the search space. More events increase the size of the process execution net and can also increase the number of synchronous transitions in the synchronous product net. Thus there are more possible markings which leads to a bigger search space. More objects also increase the size of the process execution net and add a lot of parallel behavior, thereby also increasing the search space. The run time of the Dijkstra algorithm is on average exponential in the size of the search space [16]. Therefore, the computation time grows exponentially for the number of events and objects.

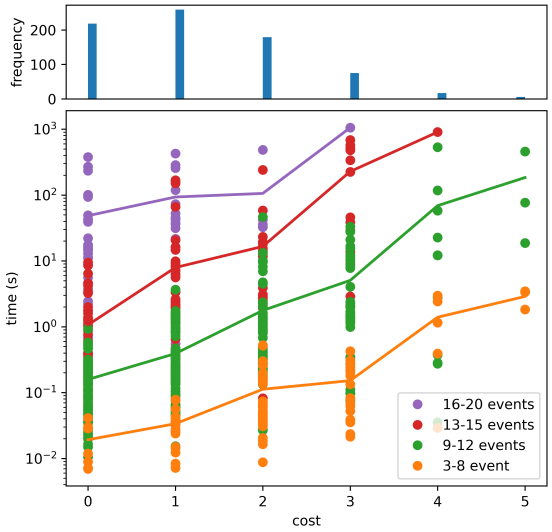


Fig. 13. Execution time on a logarithmic scale over alignment cost, grouped by the number of events.

For an alignment with a higher cost, a bigger portion of the search space is explored because Dijkstra explores all vertices that are reachable by a path that is cheaper than the cheapest path to the final marking. As a result, the run-time grows exponentially for the cost of the alignment.

8 Conclusion

This paper presented four contributions for conformance checking in object-centric process mining. First, we defined object-centric alignments generalizing traditional alignments to graphs of moves. Second, we provided an algorithm to calculate optimal alignments on an object-centric log and Petri net. The two-step approach creates a synchronous product net and searches for the optimal binding sequence from initial marking to final marking. Third, we implemented this algorithm using the open-source library OCPA [6] and made it publicly available on GitHub[1]. Finally, we performed an evaluation of the presented approach. The qualitative evaluation shows the advantages of object-centric alignments for deviation diagnostics. Complex inter-object dependencies are lost when flattening the event data, leading to contradictions in the alignment. Using object-centric alignments, we preserve these dependencies and avoid contradictions. Our quantitative evaluation indicates an exponential computation time in the number of object instances and cost of the alignment. This suggests, that an alignment of a whole object-centric event log to a moderately fitting model might be too

time-consuming. In those scenarios, one might use object-centric alignments to get specific diagnostics for individual process executions or variants.

There are two directions for future work based on object-centric alignments. On the one hand, one can investigate lifting restrictions of the current approach. For example, the assumption of a fixed object set could be dropped, allowing for the approach to introduce completely new objects that might be missing. On the other hand, one can work towards decreasing the complexity and run time: Heuristics, using the A^* algorithm, and defining relaxations of the problem are all promising directions to decrease the computation time.

References

1. van der Aalst, W.M.: Process mining. Communications of the ACM **55**(8), 76–83 (2012). https://doi.org/10.1145/2240236.2240257
2. van der Aalst, W.M.: Object-Centric Process Mining: Dealing with Divergence and Convergence in Event Data. In: Software Engineering and Formal Methods. pp. 3–25. Springer (2019). https://doi.org/10.1007/978-3-030-30446-1_1
3. van der Aalst, W.M., Adriansyah, A., van Dongen, B.: Causal Nets: A Modeling Language Tailored towards Process Discovery. In: CONCUR 2011 – Concurrency Theory, vol. 6901, pp. 28–42. Springer (2011). https://doi.org/10.1007/978-3-642-23217-6_3
4. van der Aalst, W.M., Berti, A.: Discovering object-centric petri nets. Fundam. Informaticae **175**(1-4), 1–40 (2020). https://doi.org/10.3233/FI-2020-1946
5. Adams, J.N., van der Aalst, W.M.: Precision and Fitness in Object-Centric Process Mining. In: 2021 3rd International Conference on Process Mining (ICPM). pp. 128–135. IEEE (2021). https://doi.org/10.1109/ICPM53251.2021.9576886
6. Adams, J.N., Park, G., van der Aalst, W.M.: ocpa: A Python library for object-centric process analysis. Software Impacts **14**, 100438 (2022). https://doi.org/10.1016/j.simpa.2022.100438
7. Adams, J.N., Schuster, D., Schmitz, S., Schuh, G., van der Aalst, W.M.: Defining Cases and Variants for Object-Centric Event Data. In: 2022 4th International Conference on Process Mining (ICPM). pp. 128–135. IEEE (2022). https://doi.org/10.1109/ICPM57379.2022.9980730
8. Adriansyah, A.: Aligning observed and modeled behavior (2014). https://doi.org/10.6100/IR770080, publisher: Technische Universiteit Eindhoven
9. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.: Alignment Based Precision Checking. In: Business Process Management Workshops. pp. 137–149. Springer (2013). https://doi.org/10.1007/978-3-642-36285-9_15
10. Adriansyah, A., Sidorova, N., van Dongen, B.: Cost-Based Fitness in Conformance Checking. In: 2011 Eleventh International Conference on Application of Concurrency to System Design. pp. 57–66 (2011). https://doi.org/10.1109/ACSD.2011.19
11. Borrego, D., Barba, I.: Conformance checking and diagnosis for declarative business process models in data-aware scenarios. Expert Systems with Applications **41**(11), 5340–5352 (2014). https://doi.org/10.1016/j.eswa.2014.03.010
12. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Systems with Applications **65**(C), 194–211 (2016). https://doi.org/10.1016/j.eswa.2016.08.040

13. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking: Relating Processes and Models. Springer (2018). https://doi.org/10.1007/978-3-319-99414-7
14. Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of A*. Journal of the ACM **32**(3), 505–536 (1985). https://doi.org/10.1145/3828.3830
15. Desel, J., Reisig, W.: Place/transition Petri Nets. In: Lectures on Petri Nets I: Basic Models, pp. 122–173. Springer (1998). https://doi.org/10.1007/3-540-65306-6_15
16. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**(1), 269–271 (1959). https://doi.org/10.1007/BF01386390
17. van Dongen, B.: BPI Challenge 2017 (2017). https://doi.org/10.4121/UUID:5F3067DF-F10B-45DA-B98B-86AE4C7A310B
18. van Dongen, B.F.: Efficiently computing alignments - using the extended marking equation. In: Business Process Management - 16th International Conference, BPM 2018. pp. 197–214. Springer (2018). https://doi.org/10.1007/978-3-319-98648-7_12
19. Dunzer, S., Stierle, M., Matzner, M., Baier, S.: Conformance checking: a state-of-the-art literature review. In: Proceedings of the 11th International Conference on Subject-Oriented Business Process Management. pp. 1–10. S-BPM ONE '19, Association for Computing Machinery (2019). https://doi.org/10.1145/3329007.3329014
20. Fahland, D.: Process mining over multiple behavioral dimensions with event knowledge graphs. In: Process Mining Handbook, pp. 274–319. Springer (2022). https://doi.org/10.1007/978-3-031-08848-3_9
21. Felli, P., Gianola, A., Montali, M., Rivkin, A., Winkler, S.: Cocomot: Conformance checking of multi-perspective processes via SMT. In: Polyvyanyy, A., Wynn, M.T., Looy, A.V., Reichert, M. (eds.) Business Process Management - 19th International Conference, BPM 2021, Rome, Italy, September 06-10, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12875, pp. 217–234. Springer (2021). https://doi.org/10.1007/978-3-030-85469-0_15
22. Jensen, K., Kristensen, L.M.: Colored petri nets: a graphical language for formal modeling and validation of concurrent systems. Commun. ACM **58**(6), 61–70 (2015). https://doi.org/10.1145/2663340
23. de Leoni, M., van der Aalst, W.M., van Dongen, B.F.: Data- and resource-aware conformance checking of business processes. In: Business Information Systems - 15th International Conference, BIS 2012. pp. 48–59. Springer (2012). https://doi.org/10.1007/978-3-642-30359-3_5
24. Nesterov, R., Bernardinello, L., Lomazova, I., Pomello, L.: Discovering architecture-aware and sound process models of multi-agent systems: a compositional approach. Software and Systems Modeling (May 2022). https://doi.org/10.1007/s10270-022-01008-x
25. Rozinat, A., van der Aalst, W.M.: Conformance checking of processes based on monitoring real behavior. Information Systems **33**(1), 64–95 (2008). https://doi.org/10.1016/j.is.2007.07.001
26. Sommers, D., Sidorova, N., van Dongen, B.: Aligning Event Logs to Resource-Constrained ν-Petri Nets. In: Application and Theory of Petri Nets and Concurrency, pp. 325–345. Springer (2022). https://doi.org/10.1007/978-3-031-06653-5_17
27. Song, W., Xia, X., Jacobsen, H.A., Zhang, P., Hu, H.: Efficient Alignment Between Event Logs and Process Models. IEEE Transactions on Services Computing **10**(1), 136–149 (2017). https://doi.org/10.1109/TSC.2016.2601094