Object-Centric Alignments

Lukas Liss(✉), Jan Niklas Adams(✉), and Wil M. P. van der Aalst(✉)

RWTH Aachen University, Aachen, Germany
{liss,niklas.adams,wvdaalst}@pads.rwth-aachen.de

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 analyzes event data to provide insights into processes, using a variety of conceptual models. One standard pipeline for this includes data extraction, process model discovery, and conformance checking [14]. The insights of each step are bound by the expressiveness of the used models. This paper proposes to use more expressive models for conformance checking to correctly handle inter-object dependencies, for which traditional methods fail to give correct insights. We introduce object-centric alignments that can model deviations in interacting subprocesses with multiple objects. Traditional methods use representations that model a process using a single case notion meaning that all

J. P. A. Almeida et al. (Eds.): ER 2023, LNCS 14320, pp. 201–219, 2023.
https://doi.org/10.1007/978-3-031-47262-6_11

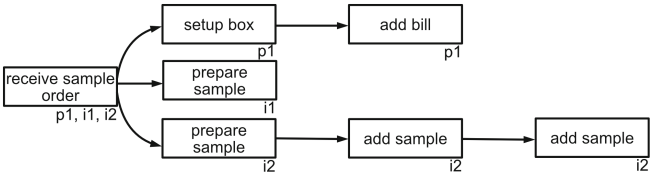


Fig. 1. A process execution of our running examples. Events are associated with objects of type package (prefix p) or item (prefix i). The process execution describes the partial order of events induced by the individual objects.

actions created for one object define a process execution. Real-world processes often do not fit that assumption. For example, a supply chain process involves multiple subprocesses operating on varied objects like raw materials, products, orders, and customers. One execution of the supply chain is not defined by a single object.

Recently, *object-centric process mining* [21] was introduced to generalize the notion of a process so that one can follow multiple objects through multiple, connected sub-processes. In object-centric process mining a process execution is a graph showing the partial order between sub-process events [7] So far, replay-based fitness has been proposed for object-centric conformance checking [5]. However, process owners are typically interested in aligning observed behavior to modeled behavior, to identify deviations, 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 is the process execution in Fig. 1 belonging to a packaging process with cross-object dependencies between a package and multiple items. This process is described using the object-centric Petri net in Fig. 2, which differs from traditional ones by introducing place types with typified tokens and variable arcs (highlighted in red). Tokens of a type can only occupy places of the same type, and variable arcs can consume multiple tokens at once. There are two types here: item (green) and package (blue). In this process, the paths of the package and items depend on each other. The first event involves all objects and decides whether it is a sample or product order which defines the following allowed behavior for the package and the items. 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 Fig. 1 in Fig. 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. 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*

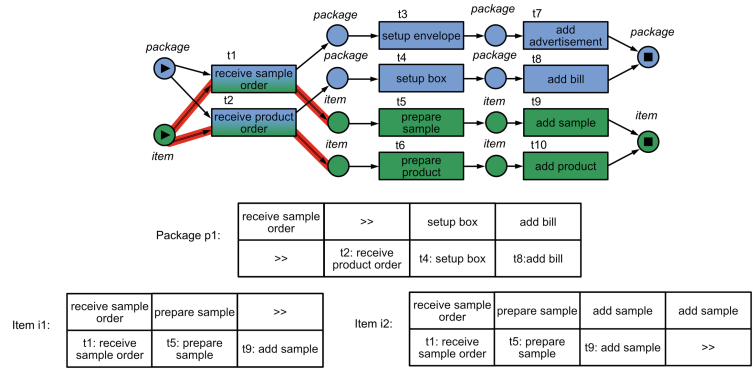


Fig. 2. Top: De-jure model as an object-centric Petri net. Variable arcs are marked red. Bottom: The process execution of Fig. 1 is flattened to the individual objects and aligned to the de-jure model's subnet of the object's types. (Color figure online)

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 computation 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 Sect. 2 and preliminaries in Sect. 3. Then, we define object-centric alignments in Sect. 4. Our algorithm to compute alignments consists of two parts: constructing the synchronous product net (Sect. 5) and finding an optimal alignment in the synchronous product net (Sect. 6). In this paper, we give a declarative description of our algorithm. The formal definitions for all the steps in Sect. 5 and Sect. 6 are presented in the extended pre-print [26]. We present a qualitative and quantitative evaluation in Sect. 7 and conclude the paper in Sect. 8.

2 Related Work

Process mining includes discovery, conformance checking, and enhancement of business processes [1]. Our approach belongs to conformance checking, where

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

behavior from the event log is compared to allowed behavior that is specified by a de-jure model [14]. For traditional processes, there exists a variety of conformance checking approaches [20], which mainly use token-based replay [29] approaches, or alignments [8]. 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 [17] or A^* [15] algorithm. Different ways to speed up the calculation have been researched [19, 31]. However, the alignment algorithm assumes the process to have a single case identifier and can therefore not be used directly with object-centric processes.

Multiple extensions to the traditional alignment algorithm use higher-order nets to consider additional dimensions together with the workflow dimension [12,13]. The data and resource-aware conformance checking approach from de Leoni et al. uses data Petri nets [25]. Felli et al. use data Petri nets together with satisfiability modulo theories to compute data-aware alignments [22]. Sommers et al. constructed a ν-Petri net to calculate resource-constrained alignments [30]. 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 [27]. Conceptual models like business artifacts [28] and GSM [23] can model the interactions between multiple process entities but can not be generated automatically from real-world event data. Thus, we use model notations from object-centric process mining [2] to model processes with a variety of objects from different types that interact with each other. Object-centric Petri nets can be discovered directly from event data from current information systems [4]. 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 as well as object-centric Petri nets [4] that can describe allowed behavior. 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 example process execution with one package and two items. Object-centric Petri nets describe object-centric behavior by using types like colored Petri nets [24]. $\mathcal{B}(A)$ is used to represent all multisets for a set A.

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 $l : T \rightarrow \mathbb{U}_{act} \cup \{\tau\}$ with invisible activity $\tau \notin \mathbb{U}_{act}$. $pt : P \rightarrow \mathbb{U}_{typ}$ 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.*

The object-centric Petri net in Fig. 2 models the running example using typed places and has variable arcs for *receive sample order* and *receive product order*. The variable arcs model that multiple items can be part of one package.

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 the consumed and produced objects per object type. For example, a binding for the object-centric Petri net in Fig. 2 can define that *t1* fires using objects *p1*, *i1*, and *i2*.

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* $B = \{(t, b) \in T \times (\mathbb{U}_{type} \not\rightarrow \mathcal{P}(\mathbb{U}_{obj})) | dom(b) = tpl(t) \wedge \forall_{ot \in tpl_{nv}(t)} \forall_{p \in pl_{nv}(t), pt(p) = ot} |b(ot)| = F(p, t)\}$. *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 multiset 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.*

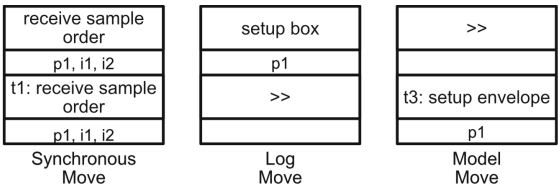


Fig. 3. One synchronous, one log, and one model move for the running example.

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 are a conceptual model to describe how observed (log) behavior and normative (model) behavior relate. It consists of moves that represent whether something occurs in the process execution, the de-jure model, or both of them.

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* $o_{log} = o_{mod} = \pi_{obj}(e) = \bigcup_{o \in range(b)} o$.

Figure 3 shows the three types of moves. Log and model moves model deviations, whereas synchronous moves model conforming behavior. For synchronous moves, activities and objects of 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, represented by skip symbol $\gg$. 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}. 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.

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 $MG_{\downarrow log} = (M_{\downarrow log}, C_{\downarrow log})$ and the reduction to moves with visible activity in the model part is $MG_{\downarrow mod} = (M_{\downarrow mod}, C_{\downarrow mod})$ 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.*
- *$M_{\downarrow mod} = \{m \in M | \pi_{ma}(m) \neq \gg\}$ 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 Fig. 4 both $MG_{\downarrow log}$ and $MG_{\downarrow model}$ are visualized for a 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. 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 $f : E_X \rightarrow M_{\downarrow log}$ such that $\forall_{e \in E_X} \pi_{act}(e) = \pi_{la}(f(e)) \wedge \pi_{obj}(e) = \pi_{lo}(f(e))$.

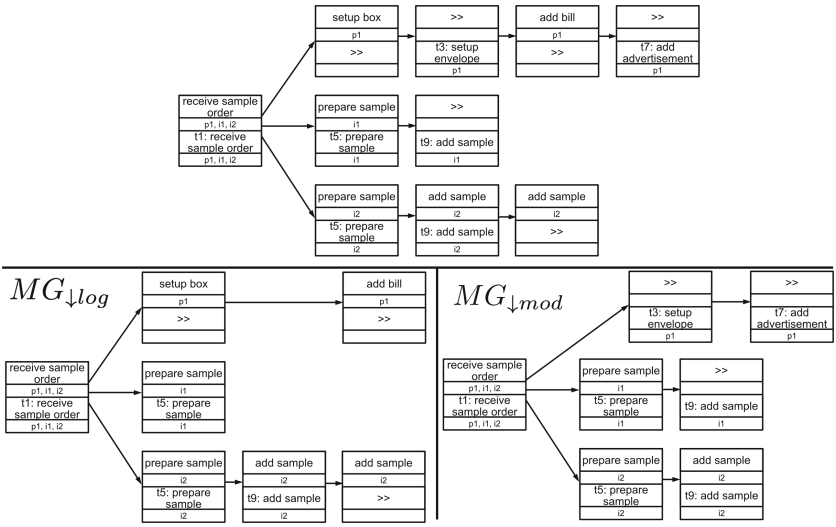


Fig. 4. Optimal object-centric alignment for the running example and reductions $MG_{\downarrow log}$ and $MG_{\downarrow model}$.

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* $f' : B \rightarrow M_{\downarrow mod}$ *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.*

Figure 4 shows an alignment for the running example. Its reduction to log and synchronous moves $MG_{\downarrow log}$, is isomorphic to the given process execution in Fig. 1. This ensures that the alignment contains the process execution behavior. The reduction to the model part relates to a binding sequence that is in the language of the de-jure model in Fig. 2. This ensures that the model part describes behavior that is accepted by the model. We want to find deviations between the process execution and the most similar allowed behavior. Thus, we want an alignment to have as few model or log moves as possible. By giving model and log moves higher costs than synchronous moves, we can prefer synchronous behavior.

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* $cost_{move} : moves(P_X, AN) \rightarrow \mathbb{R}$ *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 Fig. 4 is 6 because there are 3 model and 3 log moves that have one object each. The lower the cost the fewer deviations are in the alignment. 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 be multiple optimal alignments with the same cost for a given process execution and a de-jure Petri net. Also, practitioners can modify the cost function to weight deviations according to domain-specific knowledge. Figure 4 shows the optimal object-centric alignment for the running example. The inter-object dependencies that are defined in the object-centric Petri net in Fig. 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

This section presents a declarative description of the first part of our optimal alignments algorithm. Additional formal definitions for each presented step can be found in the extended pre-print [26]. This first part creates a synchronous product net that can generate all possible alignments. It consists of three parts, each relating to a move type. In the synchronous product net in Fig. 7 the log, model, and synchronous 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. We assume the model to use the same objects that are in the process execution.

5.1 Process Execution Net Construction

The process execution net is the part of the synchronous product net that ensures that the process execution is contained in the alignment. The construction relates to Petri net runs [16] and causal nets [3]. The process execution

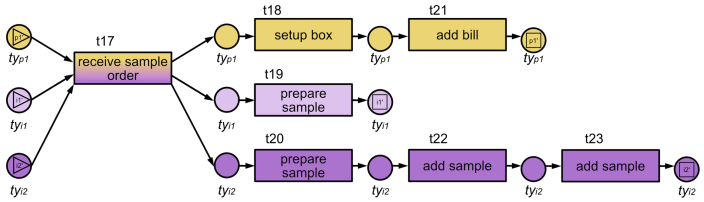


Fig. 5. Process Execution Net

contains *directly follows* relations on the level of objects. For example, the running example process execution in Fig. 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 and not by object. This differentiation is important because otherwise, deviations in one object 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. To seperate each object, we have to create a new individual type for each object in the process execution. Since objects can only have one type, we also create a new object for each object from the process execution to use them with the new types. 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. Thereby, the process execution net precisely models the process execution behavior.

5.2 Pre-processing of the Object-Centric Petri Net

The pre-processed de-jure modes will be the part of the synchronous product net that guarantees that allowed behavior is contained in the model part of the alignment. The pre-processing replaces variable arcs in the de-jure model. For transitions with variable arcs, we do not know beforehand how many objects they use. This information is needed to find synchronous transitions when creating the synchronous product net because two transitions can only be synchronous if they use the same objects which implies using the same number of objects.

As aforementioned, we assume that the set of objects for the model part of the alignment is defined by the process execution. Thus, these objects form the initial and final marking. For a predefined set of objects, 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 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

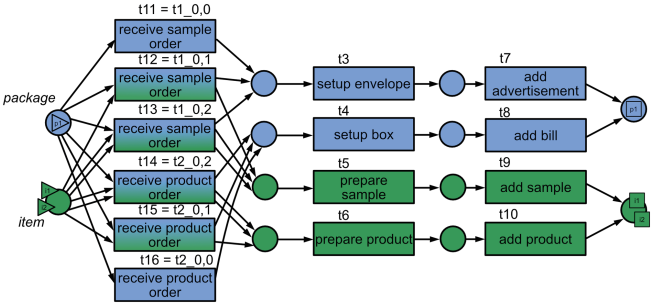


Fig. 6. Pre-processed accepting object-centric Petri net without variable arcs

Fig. 2 the result will be the Petri net without variable arcs in Fig. 6. We call the pre-processed accepting object-centric Petri net a de-jure net.

5.3 Creating the Synchronous Product Net

The synchronous product net models possible alignments. The two nets without variable arcs from the previous steps represent the model and log part of the synchronous product net. In an alignment, either process execution or de-jure parts advance individually, or they progress synchronously. Activities can occur simultaneously in both parts if they involve the same activity on identical object instances. We add a synchronous transition for each pair of transitions from the model and log part that have the same activity label and use the same number of objects per type. For comparing types between the model and log part, we first map the newly created types in the process execution net to their original types. To ensure that in the model and log part the same objects and not only the same types are used by a synchronous transition, we add ν-net requirements. The ν net requirement function assigns variables to the in-going arcs of the synchronous transitions. Arcs with the same variable have to consume the same objects. This refers to the original objects not the newly created ones in the process execution net. 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 transition to use the same object instance in the process execution net and the de-jure net. Figure 7 shows the resulting synchronous Petri net for the running example.

6 Alignments from Synchronous Product Net

This section describes the second part of our approach to finding object-centric alignments in a declarative way. Additional formal definitions and proofs can be found in the extended pre-print [26]. The input is a synchronous product net in which every binding relates to a move of an alignment. The activity and the set of objects of the move are given by the label of the transition and

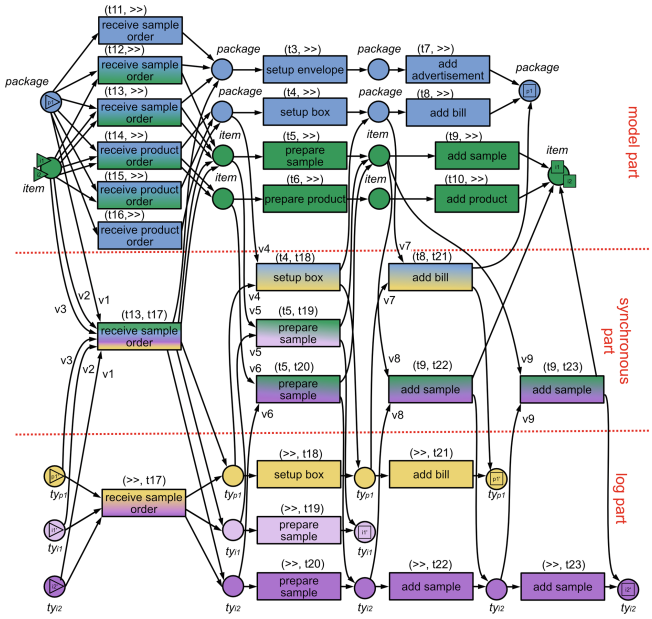


Fig. 7. Synchronous product net for the process execution net in Fig. 5 and accepting object-centric Petri net in Fig. 6

the objects that are defined in the binding. The type of the move depends on whether the transition in the binding is from the log, model, or synchronous part. All binding sequences from the initial marking to the final marking in the synchronous product net relate to an alignment because the log part ensures that the process execution is contained in the log part of the alignment and the model part ensures that the model part of the alignment describes allowed behavior. 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. The cost of the edges is given by the cost of 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.

As shown in the extended version, the search space is finite given that the event log and the de-jure model have a finite size [26]. Thus, the Dijkstra algorithm [15] can find the cheapest path. If there is no cheapest path, there is no path at all. So the de-jure model has no option to complete for the set of objects of the process execution meaning it does not contain any related allowed behavior. 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 pro-

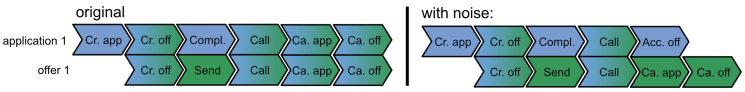


Fig. 8. A process execution from [18] and a variation of it that has some noise.

cess execution, the cheapest path relates to the optimal alignment. The resulting shortest path is a binding sequence from the initial to the final marking. It relates directly to an optimal object-centric alignment for the given process execution and the de-jure model. For the running example Fig. 4 shows the optimal object-centric alignment. 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 [18]. The evaluation is split into a qualitative part and a quantitative evaluation.

7.1 Qualitative

The qualitative evaluation aims to assess if object-centric alignment can give better insights into object-centric processes compared to traditional alignments. We used BPI2017 data, one of few public event logs convertible into an object-centric format. Its process includes two object types: application and offer. For this evaluation, we selected only the 4 most dominant variants that made up 19.6% of process executions, excluding the activities *Submit*, *Complete*, and *Accept* for clarity. 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]. We then introduced noise to the log by removing and replacing events. Figure 8 shows the original accepted process execution and the one with noise. For object *application 1*, *Cancel application* is removed and *Cancel offer* is replaced with *Accept offer*. This creates deviations from the de-jure model behavior. For instance, it is impossible to accept an application without accepting an offer, which is an inter-object dependency one wants to be respected. Also, *Cancel application* and *Cancel offer* events, now only recorded for an offer, represent unwanted behavior. We applied our object-centric alignment approach and the traditional one with flattening to the evaluation data. The optimal object-centric alignment and traditional alignment for each object can be found in Fig. 9.

The traditional alignment failed to detect that application 1 should not have been accepted. This is because the traditional alignment does not consider the inter-object dependency that there has to be a matching offer to accept. This isolated view results in the traditional alignment suggesting *Validate* and *Pending* activities are missing, reinforcing the false acceptance of *application 1*. However,

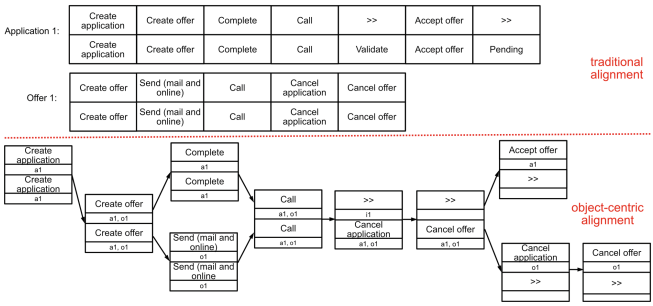


Fig. 9. Traditional and object-centric alignments for the variant with noise and the evaluation de-jure model.

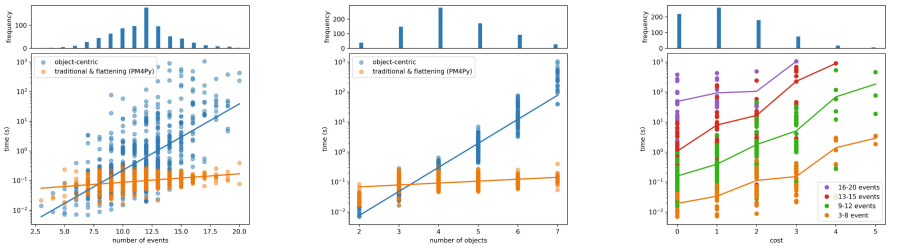
our approach, considering all inter-object dependencies, identifies *Accept Offer* as a log move. Additionally, the traditional alignment doesn't indicate any deviation for *offer 1*, creating a contradiction between alignments of *offer 1* and *application 1*. Such contradictions can not occur in an object-centric alignment because the whole process execution is considered at once. The more inter-object dependencies a process has, the bigger the benefit of object-centric alignments.

7.2 Quantitative

To evaluate the scalability of our approach, we performed a quantitative analysis of the run time, comparing our method with traditional alignment methods.

Evaluation Setup. As the data source we used BPI2017 event data [18]. 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 (See Footnote 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. For comparison with current methods, we also tracked the run time of flattening and then computing optimal alignments for the separate objects using PM4Py [11]. The raw results of that evaluation can be found at GitHub (See Footnote 1). The evaluation was performed on a 3.1 GHz Dual Core Intel Core i5 with 8 GB of RAM.

Results. We computed 755 alignments with costs ranging from 0 to a maximum of 5. Process executions had 3 to 20 events, mostly having 11 to 14 events. Object instances varied from 2 to 7. The calculation time ranged from 0.007 s to 1051.8 s. Results at the end of the attribute range are less resistant to outliers due to fewer data points. Moreover, the correlation coefficient between events and objects is 0.59, while the correlation between events and cost is −0.38. We



(a) number of events. (b) number of objects. (c) cost, by event count.

Fig. 10. Computation time on a logarithmic scale over input parameters. (Color figure online)

plotted the calculation time against the number of events (Fig. 10a), number of objects (Fig. 10b), and alignment cost (Fig. 10c), with a logarithmic scale for the time. To mitigate the effect of the negative correlation, we grouped data points in Fig. 10c by the number of events. The computation time for object-centric alignments grows exponentially across all three dimensions.

This result can be explained by the structure of the search space. More events increase the size of the process execution net and potentially the number of synchronous transitions which creates more potential markings resulting in a bigger search space. Similarly, more objects add parallel behavior, again increasing the search space. The run time of the Dijkstra algorithm is on average exponential in the size of the search space [17]. Therefore, the computation time grows exponentially for the number of events and objects. For an alignment with a higher cost, a bigger portion of the search space is explored, resulting in exponential growth for the cost of the alignment. For small process executions, object-centric alignments (blue graph) can be faster than traditional ones (orange graph) because of the overhead of flattening, but the traditional method has better scalability, as one can see in Fig. 10a and Fig. 10b.

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 them. 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 (See Footnote 1). There are some limitations of the provided model notation and implementation. The notation does not include additional perspectives to the workflow dimension, which makes it less expressive in that regard. But it also makes the notation and implementation applicable to any domain with normative and observed workflow information. If

there are multiple cheapest alignments, it depends on the implementation which one is returned. The presented algorithm does not detect missing or redundant objects although the alignment notation would support that. Finally, we evaluated our approach. The qualitative evaluation shows the benefits of object-centric alignments in detecting deviations because they respect inter-object dependencies. 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 limitations of the current approach, like finding missing or redundant objects. 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.

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

References

1. van der Aalst, W.M.P.: Process mining. Commun. ACM **55**(8), 76–83 (2012). https://doi.org/10.1145/2240236.2240257
2. Aalst, W.M.P.: Object-centric process mining: dealing with divergence and convergence in event data. In: Ölveczky, P.C., Salaün, G. (eds.) SEFM 2019. LNCS, vol. 11724, pp. 3–25. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30446-1_1
3. van der Aalst, W., Adriansyah, A., van Dongen, B.: Causal nets: a modeling language tailored towards process discovery. In: Katoen, J.-P., König, B. (eds.) CONCUR 2011. LNCS, vol. 6901, pp. 28–42. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23217-6_3
4. 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
5. Adams, J.N., van der Aalst, W.M.P.: Precision and fitness in object-centric process mining. In: 3rd International Conference on Process Mining, ICPM 2021, Eindhoven, The Netherlands, 31 October–4 November 2021, pp. 128–135. IEEE (2021). https://doi.org/10.1109/ICPM53251.2021.9576886
6. Adams, J.N., Park, G., van der Aalst, W.M.P.: ocpa: a python library for object-centric process analysis. Softw. 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.P.: Defining cases and variants for object-centric event data. In: 4th International Conference on Process Mining, ICPM 2022, Bolzano, Italy, 23–28 October 2022, 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

9. Adriansyah, A., Munoz-Gama, J., Carmona, J., van Dongen, B.F., van der Aalst, W.M.P.: Alignment based precision checking. In: La Rosa, M., Soffer, P. (eds.) BPM 2012. LNBIP, vol. 132, pp. 137–149. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36285-9_15
10. Adriansyah, A., Sidorova, N., van Dongen, B.F.: Cost-based fitness in conformance checking. In: 11th International Conference on Application of Concurrency to System Design, ACSD 2011, Newcastle Upon Tyne, UK, 20–24 June 2011, pp. 57–66. IEEE Computer Society (2011). https://doi.org/10.1109/ACSD.2011.19
11. Berti, A., van Zelst, S.J., van der Aalst, W.M.P.: Process mining for python (PM4Py): bridging the gap between process- and data science. CoRR abs/1905.06169 (2019)
12. Borrego, D., Barba, I.: Conformance checking and diagnosis for declarative business process models in data-aware scenarios. Expert Syst. Appl. **41**(11), 5340–5352 (2014). https://doi.org/10.1016/j.eswa.2014.03.010
13. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. **65**, 194–211 (2016). https://doi.org/10.1016/j.eswa.2016.08.040
14. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99414-7
15. Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of A*. J. ACM **32**(3), 505–536 (1985). https://doi.org/10.1145/3828.3830
16. Desel, J., Reisig, W.: Place/transition Petri nets. In: Reisig, W., Rozenberg, G. (eds.) ACPN 1996. LNCS, vol. 1491, pp. 122–173. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_15
17. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1**, 269–271 (1959). https://doi.org/10.1007/BF01386390
18. van Dongen, B.: BPI Challenge 2017 (2017). https://doi.org/10.4121/UUID:5F3067DF-F10B-45DA-B98B-86AE4C7A310B
19. Dongen, B.F.: Efficiently computing alignments. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 197–214. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_12
20. 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, S-BPM ONE 2019, Seville, Spain, 26–28 June 2019, pp. 4:1–4:10. ACM (2019). https://doi.org/10.1145/3329007.3329014
21. Fahland, D.: Process mining over multiple behavioral dimensions with event knowledge graphs. In: van der Aalst, W.M.P., Carmona, J. (eds.) Process Mining Handbook. LNBIP, vol. 448, pp. 274–319. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-08848-3_9
22. 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., Van Looy, A., Reichert, M. (eds.) BPM 2021. LNCS, vol. 12875, pp. 217–234. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85469-0_15
23. Hull, R., et al.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: Bravetti, M., Bultan, T. (eds.) WS-FM 2010. LNCS, vol. 6551, pp. 1–24. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19589-1_1

24. 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
25. de Leoni, M., van der Aalst, W.M.P., van Dongen, B.F.: Data- and resource-aware conformance checking of business processes. In: Abramowicz, W., Kriksciuniene, D., Sakalauskas, V. (eds.) BIS 2012. LNBIP, vol. 117, pp. 48–59. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30359-3_5
26. Liss, L., Adams, J.N., van der Aalst, W.M.P.: Object-centric alignments (2023). https://doi.org/10.48550/arXiv.2305.05113
27. Nesterov, R., Bernardinello, L., Lomazova, I.A., Pomello, L.: Discovering architecture-aware and sound process models of multi-agent systems: a compositional approach. Softw. Syst. Model. **22**(1), 351–375 (2023). https://doi.org/10.1007/s10270-022-01008-x
28. Nigam, A., Caswell, N.S.: Business artifacts: an approach to operational specification. IBM Syst. J. **42**(3), 428–445 (2003). https://doi.org/10.1147/sj.423.0428
29. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008). https://doi.org/10.1016/j.is.2007.07.001
30. Sommers, D., Sidorova, N., van Dongen, B.F.: Aligning event logs to resource-constrained ν-Petri nets. In: Bernardinello, L., Petrucci, L. (eds.) PETRI NETS 2022. LNCS, vol. 13288, pp. 325–345. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-06653-5_17
31. Song, W., Xia, X., Jacobsen, H., Zhang, P., Hu, H.: Efficient alignment between event logs and process models. IEEE Trans. Serv. Comput. **10**(1), 136–149 (2017). https://doi.org/10.1109/TSC.2016.2601094