Conformance Checking and Performance Analysis Using Object-Centric Directly-Follows Graphs

Gyunam Park(✉), Jan Niklas Adams, and Wil M. P. van der Aalst

Process and Data Science, RWTH Aachen University, Aachen, Germany
{gnpark,niklas.adams,wvdaalst}@pads.rwth-aachen.de

Abstract. Traditional process mining often simplifies the multi-object reality of business processes, linking each event to a single object called "the case". This oversimplification may lead to inaccurate analyses, caused by missing interactions between object types. In contrast, object-centric process mining permits events to be tied to multiple objects, capturing complex interactions and providing a more accurate representation of business processes. This paper introduces an approach for supporting object-centric process mining utilizing Object-Centric Directly-Follows Graphs (OC-DFGs). Despite their advantages, e.g., simplicity, OC-DFGs have been relatively untapped for essential process mining tasks, such as conformance checking and performance analysis. In order to address this, our research presents a comprehensive approach for OC-DFG-based conformance checking and performance analysis. We fully implement the proposed approach as a web application and demonstrate the use of OC-DFGs for these tasks within a case study of a real-life loan application process.

Keywords: Object-Centric Process Mining · Object-Centric Directly-Follows Graphs · Conformance Checking · Performance Analysis

1 Introduction

Conventional process mining necessitates every event to associate with a singular object type (i.e., a single case notion), and each change of case perspective requires new data extraction and configuration [12]. This requirement can inadvertently duplicate events (convergence) and displace causal information between events (divergence) [1]. Furthermore, the single-object focus can overlook interactions with other object types in the process, leading to potentially inaccurate and misleading analysis. These limitations are due to the oversimplified representation of processes that involve multiple objects, essentially flattening a 3-dimensional reality into 2-dimensional event logs [2].

Similarly to traditional process mining, process models play a pivotal role in object-centric process mining. The primary phase involves the discovery of process models from object-centric event logs. Additionally, conformance checking,

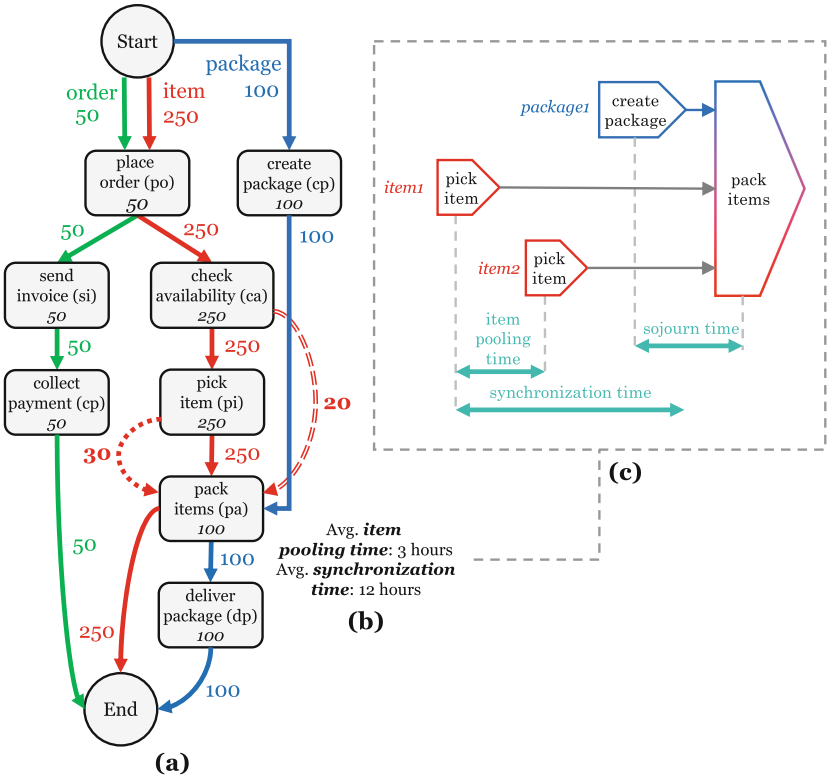

Fig. 1. A motivating example: (a) an example of OC-DFGs, (b) object-centric performance metrics for activity *pack items*, (c) object-centric performance metrics for a *pack items* event.

which compares observed behavior (event log) with modeled behavior (process model), aids in identifying discrepancies between actual process execution and the process model. Moreover, performance indicators can be visualized and analyzed within the process model's context, offering clear insights into process delays or inefficiencies.

A variety of modeling formalisms is employed to represent processes in object-centric process mining, including Proclets [14], Object-Centric Behavioral Constraint (OCBC) models [17], and Object-Centric Petri Nets (OCPNs) [4]. Techniques for conformance checking [5,7,15,20] and performance analysis [21,22] have been developed using the formal semantics of these advanced modeling formalisms.

Despite the prevalent adoption of directly-follows-based process maps in traditional process mining tools, attributed to their simplicity and interactive filtering capabilities, their object-centric equivalents, i.e., Object-Centric DFGs (OC-DFGs) [1], have not been widely adopted in practice due to the lack of capabilities for conformance checking and performance analysis. To address this gap, our work presents novel methods for conformance checking and performance

In more detail, this work presents a conformance checking technique using an OC-DFG and object-centric event logs. To that end, we translate each directly-follows graph constituting an OC-DFG to a workflow net and compute *alignments* of the corresponding projected event log on the workflow net. Next, we visualize the alignments using the OC-DFG. For example, Fig. 1(a) shows an example of an OC-DFG that visualizes alignments. The process model describes an order-to-cash process with three object types: *order*, *item*, and *package*. *Log moves* of 30 items, i.e., the actual behavior between *pick item* and *pack items* that is missing in the model, are described as a dotted line. *Model moves* of 20 items, i.e., 20 items that skip *pick item*, are described as a double line.

Next, we introduce a performance analysis technique by projecting the alignments on the process model and computing object-centric performance metrics such as synchronization time and pooling time. For example, a synchronization time for execution of *pack items* with *package1*, *item1*, and *item2*, described in Fig. 1(b), refers to the time taken from the picking of the last item to the creation of packages, as shown in Fig. 1(c). A pooling time of items in the execution refers to the time taken from the first picking to the last picking, as depicted in Fig. 1(c).

The proposed approach is fully implemented as a web application with a dedicated user interface. Furthermore, we evaluate the approach by applying it in a real-life loan application process using the implementation.

The remainder of this paper is organized as follows. We discuss the related work in Sect. 2. Next, we introduce preliminaries on event data and OC-DFGs in Sect. 3. Afterward, we explain an approach to conformance checking using OC-DFGs and alignments in Sect. 4 and performance analysis in Sect. 5. Then, Sect. 6 introduces the implementation of the proposed approach, presents a use case study on a real-life loan application process using the implementation, and discusses the utility and limitations of the proposed approach. Finally, Sect. 7 concludes the paper.

2 Related Work

This section introduces related work on object-centric process mining and process mining using directly-follows graphs.

2.1 Object-Centric Process Mining

This research aligns with recent developments in object-centric process mining [1]. Traditional process mining methods assume each event is associated with a single case, viewing the event log as a collection of isolated event sequences. Object-centric process mining deviates from this assumption, permitting an event to associate with multiple cases, leading to shared events between sequences, i.e., a graph of events.

A range of process modeling formalisms has been proposed to address the complexities of multiple case notions. Proclets [14], the first modeling technique designed to describe interacting workflow processes, was later extended by

artifact-centric modeling [11]. DB nets [19], a modeling technique built on colored Petri nets, were also introduced. Li et al. [17] proposed Object-Centric Behavioral Constraint (OCBC) models, merging data models with process models. In addition, Object-Centric Petri Nets (OCPNs), a restricted variant of colored Petri nets, were suggested, where places are typed, tokens refer to objects, and transitions correspond to activities [4]. Moreover, typed Jackson nets, a subclass of typed Petri nets, were recently suggested along with a unique reconstructability property [9]. Alongside these advanced modeling languages, OC-DFGs have been presented as a simpler solution to address the inherent complexity in advanced modeling languages [1].

Numerous techniques have been proposed to facilitate object-centric process mining using these advanced modeling formalisms. Initially, process discovery techniques were suggested to automatically discover process models in Proclets [18], OCBC [17], and OCPNs [4] from event data. Furthermore, conformance checking techniques were developed to assess the discrepancies between modeled behavior and actual behavior in event data using Proclets [15], OCBC [5], and OCPNs [4,7]. Moreover, techniques for performance analysis have been introduced. For instance, an approach to object-centric performance analysis based on OCPNs was proposed in [21].

While several process mining approaches utilizing OC-DFGs exist, such as the baseline approach for discovering OC-DFGs from object-centric event logs [1] and the technique by Berti and van der Aalst for discovering Multiple View Point (MVP) models from databases [10], there remains a gap in the area of conformance checking and performance analysis using OC-DFGs. To our knowledge, there is no existing work that tackles these two aspects with OC-DFGs. This paper, therefore, strives to fill this gap by developing a technique for conformance checking and performance analysis using OC-DFGs.

2.2 Process Mining Using Directly-Follows Graphs

Directly-Follows Graphs (DFGs) are advantageous in traditional process mining due to their simplicity, allowance for vagueness, and scalability [6]. First, these models are easier to understand for end-users, as they circumvent the complexity often associated with formal models like Petri nets and BPMN. Secondly, they accommodate vagueness, allowing them to represent the majority of process behaviors without having to account for every outlier or deviation. Finally, directly-follows graphs are highly scalable. They can handle logs with millions of events efficiently, which is a crucial requirement for commercial process mining tools.

Many techniques have been developed to enhance process mining using DFGs. First, various discovery techniques aim to lessen the complexity of DFGs. For instance, Heuristic Miner [23] excludes infrequent behavioral relations based on the frequency of occurrences. Leemans [16] present a technique for conformance checking and performance analysis with DFGs. They propose a DFG-to-Petri-net conversion to align the model with a log. This alignment serves as

a foundation for analyzing discrepancies between the log and the model. Subsequently, they determine performance metrics by projecting alignments onto DFGs. In our work, we employ the method proposed by Leemans [16] to calculate alignments for each DFG that forms a part of an OC-DFG. The alignment is then projected onto the DFG to compute object-centric performance metrics.

3 Preliminaries

In this section, we introduce object-centric event logs, object-centric directly-follows graphs, and alignments.

3.1 Object-Centric Event Logs (OCELs)

An object-centric event log overcomes the limitation of traditional event logs by relating events to multiple interacting objects [1]. To define object-centric event logs, we first introduce several universes.

Definition 1 (Universes). *$\mathbb{U}_{ei}$ is the universe of event identifiers, $\mathbb{U}_{oi}$ is the universe of object identifiers, $\mathbb{U}_{act}$ is the universe of activity names, $\mathbb{U}_{time}$ is the universe of timestamps, $\mathbb{U}_{ot}$ is the universe of object types, $\mathbb{U}_{attr}$ is the universe of attributes, $\mathbb{U}_{val}$ is the universe of values, and $\mathbb{U}_{map} = \mathbb{U}_{attr} \nrightarrow \mathbb{U}_{val}$ is the universe of attribute-value mappings. For any $f \in \mathbb{U}_{map}$ and $x \notin dom(f)$, $f(x) = \perp$.*

Using the universes, we define an object-centric event log.

Definition 2 (Object-Centric Event Log). *An object-centric event log is a tuple $L = (E, O, \mu, R)$, where $E \subseteq \mathbb{U}_{ei}$ is a set of events, $O \subseteq \mathbb{U}_{oi}$ is a set of objects, $\mu \in (E \to \mathbb{U}_{map}) \cup (O \to \mathbb{U}_{map})$ is a mapping, and $R \subseteq E \times O$ is a relation, s.t. for any $e \in E$, $\mu(e)(act) \in \mathbb{U}_{act}$ and $\mu(e)(time) \in \mathbb{U}_{time}$, and for any $o \in O$, $\mu(o)(type) \in \mathbb{U}_{ot}$. $\prec_L \subset E \times E$ is a total order such that for any pair of events $e_1, e_2 \in E : e_1 \prec_L e_2$ implies $e_1 \neq e_2$ and $\mu(e_1)(time) \leq \mu(e_2)(time)$.*

For the sake of brevity, we denote $\mu(e)(x)$ as $\mu_x(e)$ and $\mu(o)(x)$ as $\mu_x(o)$. Table 1 describes a fragment of a simple event log $L_1 = (E_1, O_1, \mu_1, R_1)$ with $E_1 = \{e_1, e_2, \dots\}$, $O_1 = \{o_1, , i_1, i_2, \dots\}$, $R_1 = \{(e_1, o_1), (e_1, i_1), \dots\}$, $\mu_{act}(e_1) = po$, $\mu_{time}(e_1) = 25\text{-}11\text{-}2023{:}09.35$, $\mu_{type}(o_1) = order$, and $\mu_{type}(i_1) = item$.

The event sequence and trace of an object are defined as follows.

Table 1. A fragment of an event log

event id	activity	timestamp	order	item
e_1	*place order (po)*	25-11-2023:09.35	$\{o_1\}$	$\{i_1, i_2, i_3\}$
e_2	*check availability (ca)*	25-11-2023:13.35	$\emptyset$	$\{i_1\}$
e_3	*check availability (ca)*	25-11-2023:18.35	$\emptyset$	$\{i_2\}$
e_4	*send invoice (si)*	26-11-2023:15.35	$\{o_1\}$	$\emptyset$
...	...	...	...	...

Definition 3 (Event Sequences and Traces). *Let* $L = (E, O, \mu, R)$ *be an object-centric event log. For object* $o \in O$, $seq(o) = \langle e_1, e_2, \ldots, e_n \rangle$ *s.t.* $\{e_1, e_2, \ldots, e_n\} = \{e \in E \mid (e, o) \in R\}$ *and* $e_i \prec_L e_j$ *for any* $1 \leq i < j \leq n$ *is the sequence of all events where the object is involved in. For object* $o \in O$, $trace(o) = \langle a_1, a_2, \ldots, a_n \rangle$ *s.t.* $seq(o) = \langle e_1, e_2, \ldots, e_n \rangle$ *and* $a_i = \mu_{act}(e_i)$ *for any* $1 \leq i \leq n$ *is the trace of the object.*

Given L_1, $seq(o_1) = \langle e_1, e_4 \rangle$ and $trace(o_1) = \langle po, si \rangle$.

3.2 Object-Centric Directly-Follows Graphs (OC-DFGs)

An OC-DFG can be conceptualized as a layered arrangement of Directly-Follows Graphs (DFGs), where each DFG corresponds to a distinct object type within a business process. In this layered structure, the arcs, as well as the start and end activities, are always associated with a specific object type. The interlinking within this multilayered construct is achieved through shared activities which, acting as interconnecting nodes, bridge different object types.

Definition 4 (OC-DFGs). *An OC-DFG* $M = (OT, A, A_{start}, A_{end}, R)$ *consists of*

- $OT \subseteq \mathbb{U}_{ot}$ *is a set of object types,*
- $A \subseteq \mathbb{U}_{act}$ *is a set of activities,*
- $A_{start} = \{start_{ot} \mid ot \in OT\}$ *is a set of start activities and* $A_{end} = \{end_{ot} \mid ot \in OT\}$ *is a set of end activities such that* $(A_{start} \cup A_{end}) \cap A = \emptyset$, *and*
- $R \subseteq \{(ot, a_1, a_2) \in OT \times (A \cup A_{start}) \times (A \cup A_{end}) \mid (a_1 \in A_{start} \implies a_1 = start_{ot}) \wedge (a_2 \in A_{end} \implies a_2 = end_{ot})\}$ *is a set of edges labeled with the corresponding object type.*

Figure 1 shows an OC-DFG with *order*, *item*, and *package* as object types. An OC-DFG can be discovered from an object-centric event log.

Definition 5 (Discovering OC-DFGs). *Let* $L = (E, O, \mu, R)$ *be an object-centric event log.* $disc(L) = (OT, A, A_{start}, A_{end}, R)$ *is the corresponding OC-DFG where*

- $OT = \{\mu_{type}(o) \mid o \in O\}$ *is the set of object types,*
- $A = \{\mu_{act}(e) \mid e \in E\}$ *is the set of activities,*
- $R = \{(ot, a_i, a_{i+1}) \mid o \in O \wedge ot = \pi_{type}(o) \wedge trace(o) = \langle a_1, a_2, \ldots, a_n \rangle \wedge a_0 = start_{ot} \wedge a_{n+1} = end_{ot} \wedge 0 \leq i \leq n\}$ *is the set of edges with the corresponding object type.*

4 Conformance Checking

Conformance checking techniques compare an event log with a process model, which can be constructed either manually or by process discovery techniques. The commonly used technique for conformance checking involves calculating

alignments on Petri nets. In this section, we describe how alignments can be applied to OC-DFGs and visualized accordingly.

Figure 2 shows an overview of the approach. First, an OC-DFG is translated into multiple workflow nets, each of which represents a unique object type. The OCEL is then converted into a set of event logs projected on different object types. Subsequently, alignments are calculated for each workflow net and event log pair. Finally, these individual alignments are visualized within the OC-DFG.

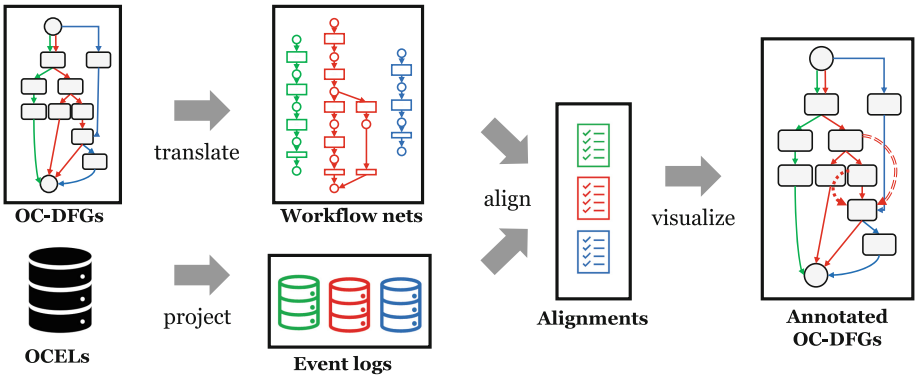

Fig. 2. An overview of conformance checking using OC-DFGs

4.1 Translating an OC-DFG to Workflow Nets

First, we translate an OC-DFG, i.e., $M = (OT, A, A_{start}, A_{end}, R)$, into a set of DFGs projected on object types OT. For $ot \in OT$, a projected DFG ($DFG_{ot} = (A_{ot}, start_{ot}, end_{ot}, R_{ot})$) consists of

- a set of activity nodes, i.e., $A_{ot} = \{a \in A \mid \exists_{a' \in A}\ (ot, a, a') \in R \vee (ot, a', a) \in R\}$,
- a start node, i.e., $start_{ot} \in A_{start}$,
- an end node, i.e., $end_{ot} \in A_{end}$, and
- a set of edges, i.e., $R_{ot} = \{(s,t) \in (A_{ot} \cup \{start_{ot}\}) \times (A_{ot} \cup \{end_{ot}\}) \mid (ot, s, t) \in R\}$.

Figure 3(a) shows an example of projecting M_1 to DFG_{item}.

Next, we translate each DFG to a workflow net. A translated workflow net ($PN_{ot} = (P, T, F, l)$) from a DFG ($DFG_{ot} = (A_{ot}, start_{ot}, end_{ot}, R_{ot})$) consists of

- a set of places that correspond to the nodes, i.e., $P = A_{ot} \cup \{start_{ot}, end_{ot}\}$,
- a set of transitions correspond to the edges, i.e., $T = R_{ot}$,
- a set of arcs that connect the transitions (i.e., the edges) to the corresponding places (i.e., the source and target nodes), i.e., $F = \bigcup_{(s,t) \in R_{ot}} \{(s, (s,t)), ((s,t), t)\}$, and
- a labeling function l such that, for $(s,t) \in T$, $l(s,t) = t$ if $t \neq end_{ot}$. $l(s,t) = \tau$ otherwise.

Figure 3(b) shows translating DFG_{item} to PN_{item}.

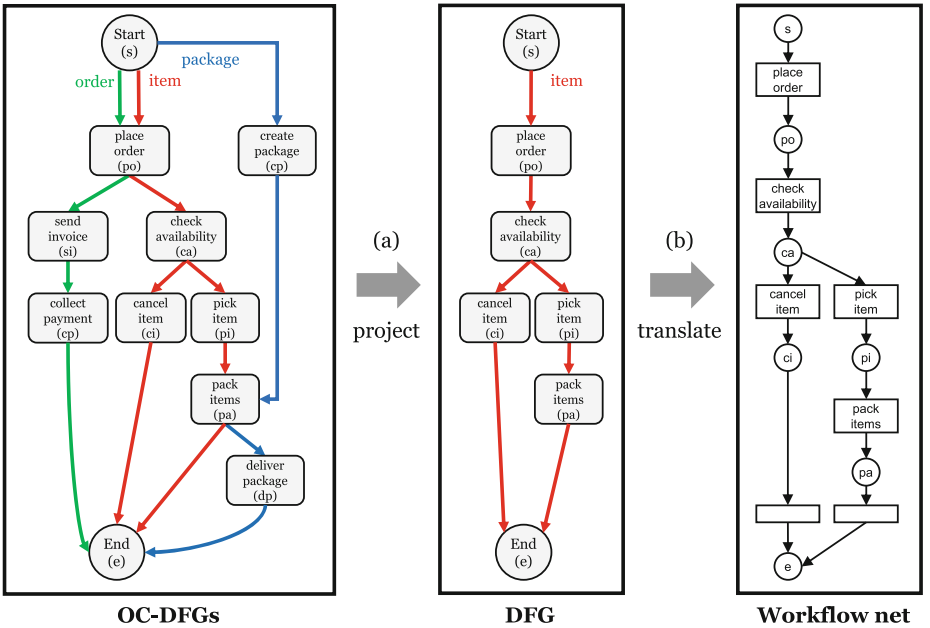

Fig. 3. (a) A OC-DFG is projected on *item* object type, resulting in a DFG representing the process model of items. (b) A projected DFG is translated to a workflow net.

4.2 Projecting an OCEL to Event Logs

We convert an object-centric event log $L = (E, O, \mu, R)$ into projected event logs, i.e., $\{L_{ot_1}, L_{ot_2}, \dots\}$, each corresponding to an object type of the OC-DFG. For any object type ot, we create a projected event log $L_{ot} = [seq(o) \mid o \in O \wedge \mu_{type}(o) = ot]$. Given the object-centric event log L described in Table 2, we derive $L_{item} = [\langle e_{11}, e_{12}, e_{13}, e_{21}\rangle, \dots]$ and $L_{package} = [\langle e_{20}, e_{21}, e_{22}\rangle, \dots]$.

4.3 Aligning Event Logs with Workflow Nets

Optimal alignments are then computed for each object type ot, utilizing the corresponding projected event log L_{ot} and the translated workflow net PN_{ot}. Several techniques have been developed to calculate optimal alignments. The most recognized approach employs the A^* algorithm to identify the shortest path in the reachability graph of a synchronous product net [3], using the marking equation of workflow nets. Figure 4 showcases optimal alignments of the package and item object types, respectively. For example, *item2* has a log move at e_{17} (*pick item*), e_{18} (*inspect item*), and e_{19} (*report item*), whereas *item3* has a model move at *pick item*.

package1's alignment

log	e_{20}	e_{21}	e_{22}
model	cp	pa	dp

item1's alignment

e_{11}	e_{12}	e_{13}	e_{21}
po	ca	pi	pa

item2's alignment

e_{11}	e_{14}	e_{15}	e_{17}	e_{18}	e_{19}	e_{21}
po	ca	pi	≫	≫	≫	pa

item3's alignment

e_{11}	e_{16}	≫	e_{21}
po	ca	pi	pa

...

Fig. 4. Optimal alignments for the package and item object types

4.4 Visualizing Alignments

Alignments result in two types of deviations: log moves and model moves. Log moves suggest that an event appeared in the log that the model did not permit. These are visualized using a dotted edge connecting model nodes, indicating additional events executed between the source and target activities in the model. There are five distinct situations in which log moves may occur, and each is visualized in a different way. Assuming that Fig. 5(a) presents a DFG for a specific object type, the situations can be explained as follows:

1. A log move at the beginning of an alignment, e.g., $\langle (e_1, \gg), (e_2, a), (e_3, b) \rangle$, is visualized as a dotted edge connecting the start node to the activity of the first synchronous move (cf. Fig. 5(b)).
2. A log move in the middle of an alignment, e.g., $\langle (e_1, a), (e_2, \gg), (e_3, b) \rangle$, is visualized as a dotted edge connecting the activities of the preceding and following synchronous moves (cf. Fig. 5(c)). Additionally, we handle two special cases:
   (a) If $\mu_{act}(e_2) = a$, the log move is visualized as a self-loop to the activity of the preceding synchronous move (cf. Fig. 5(d)).
   (b) If $\mu_{act}(e_2) = b$, the log move is visualized as a self-loop to the activity of the following synchronous move (cf. Fig. 5(e)).

Table 2. A fragment of an event log.

event id	activity	timestamp	order	item	package
...	...	...	...	...	...
e_{11}	*place order (po)*	27-11-2023:09.35	$\{o_1\}$	$\{i_1, i_2, i_3\}$	$\emptyset$
e_{12}	*check availability (ca)*	27-11-2023:13.35	$\emptyset$	$\{i1\}$	$\emptyset$
e_{13}	*pick item (pi)*	28-11-2023:11.35	$\emptyset$	$\{i1\}$	$\emptyset$
e_{14}	*check availability (ca)*	28-11-2023:12.35	$\emptyset$	$\{i2\}$	$\emptyset$
e_{15}	*pick item (pi)*	28-11-2023:18.35	$\emptyset$	$\{i2\}$	$\emptyset$
e_{16}	*check availability (ca)*	29-11-2023:09.35	$\emptyset$	$\{i3\}$	$\emptyset$
e_{17}	*pick item (pi)*	29-11-2023:10.35	$\emptyset$	$\{i2\}$	$\emptyset$
e_{18}	*inspect item (ii)*	29-11-2023:15.35	$\emptyset$	$\{i2\}$	$\emptyset$
e_{19}	*report item (ri)*	29-11-2023:17.35	$\emptyset$	$\{i2\}$	$\emptyset$
e_{20}	*create package (cp)*	30-11-2023:11.35	$\emptyset$	$\emptyset$	$\{p1\}$
e_{21}	*pack items (pa)*	30-11-2023:12.35	$\emptyset$	$\{i1, i2, i3\}$	$\{p1\}$
e_{22}	*deliver package (dp)*	02-12-2023:17.35	$\emptyset$	$\emptyset$	$\{p1\}$
...	...	...	...	...	...

3. A log move at the end of an alignment, e.g., $\langle (e_1, a), (e_2, b), (e_3, \gg) \rangle$, is visualized as a dotted edge connecting the activity of the last synchronous move to the end node (cf. Fig. 5(f)).

Conversely, model moves, indicated by a double-line edge bypassing a model node, suggest that an event that the model dictates should have occurred was missing from an event log trace.

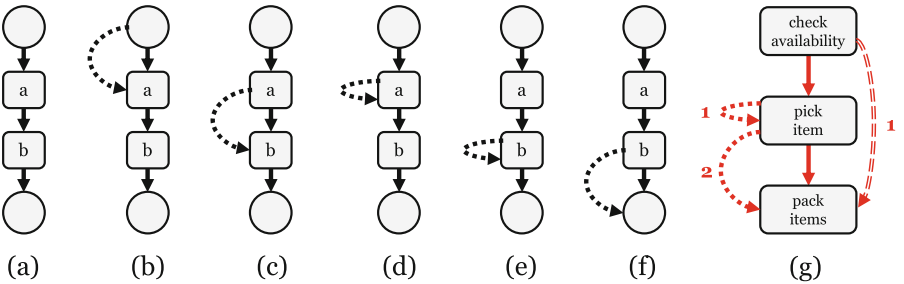

Fig. 5. (a) A DFG, (b–f) Five different types of log moves, and (g) Log moves (dotted lines) and model moves (double line) visualized on an OC-DFG.

Figure 5(g) visualizes the alignments described in Fig. 4. For example, the dotted edge connecting *pick item* node shows the log move of *pick item* (with 1 indicating *item2*'s log move at e_{17}, i.e., *pick item*), while the dotted edge connecting *pick item* to *pack items* indicates the log move between *pick item* and *pack items* (with 2 denoting *item2*'s log moves at e_{18}, i.e., *inspect item*, and e_{19}, i.e., *report item*, respectively). Moreover, the double-line edge in Fig. 5(g) demonstrates the model move at *pick item* (with 1 indicating *item3*'s model move at *pick item*).

5 Performance Analysis

Using the alignments computed in the previous section, we calculate various object-centric performance metrics in relation to the process model. Firstly, the alignments are projected onto the process model. An *event relationship* is then established by linking each event for which performance metrics need to be computed with the preceding events. Lastly, object-centric performance measures for an event are computed using its associated event relationship.

5.1 Projecting Alignments

First, we project alignments on the process model. The synchronous moves of an alignment are linked to the corresponding activities in the process model. For instance, synchronous moves of *item1*, such as (e_{11}, po), (e_{12}, ca), (e_{13}, pi), and (e_{21}, pa), are linked to respective activities *place order*, *check availability*, *pick item*, and *pack items*, as depicted in Fig. 6(a).

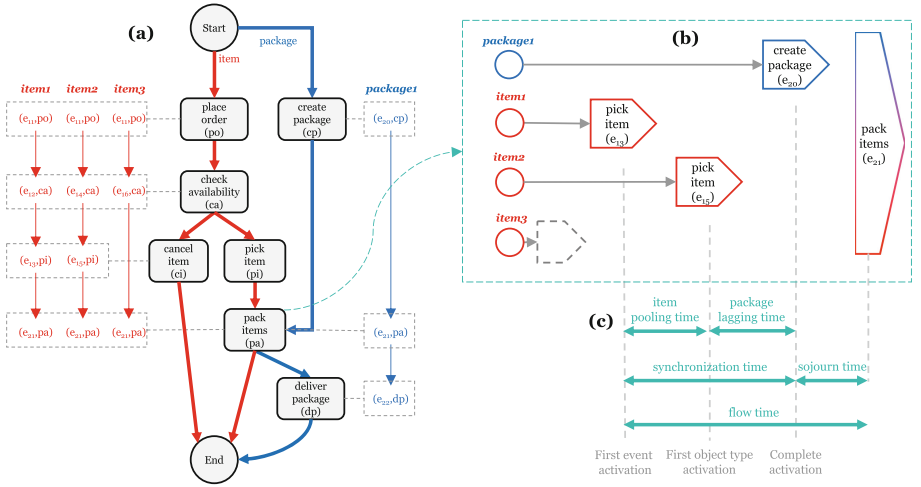

Fig. 6. (a) A OC-DFG projected by alignments. (b) A event relationship of e_{21} (*pack items*). (c) Object-centric performance metrics of e_{21} (*pack items*).

5.2 Establishing Event Relationship

Then, we define an event relationship for each event by associating the event with its preceding events. Figure 6(b) demonstrates an event relationship of e_{21}, where *item1*'s e_{13}, *item2*'s e_{15}, and *package1*'s e_{20} are the preceding events. This is done by first identifying the objects involved in the event by collecting the synchronous moves for the event. For instance, the synchronous moves for e_{21} include *item1*'s (e_{21}, pa), *item2*'s (e_{21}, pa), *item3*'s (e_{21}, pa), and *package1*'s (e_{21}, pa). From this, it is deduced that *item1*, *item2*, *item3*, and *package1* are involved in e_{21}.

Next, for each object, we identify the event that corresponds to the preceding activity of the target activity in the model, e.g., the preceding activity of *pack items* is *pick item* and *create package*. For example, *item1*'s e_{13} and *item2*'s e_{15} correspond to the activity *pick item*, while *package1*'s e_{20} corresponds to activity *create package*. Note that *item3* does not have an event that corresponds to the preceding activity of *pack items*.

5.3 Object-Centric Performance Measures

With the event relationship of an event, we can calculate various object-centric performance measures. To achieve this, we introduce the concepts of *first event activation*, *first object type activation*, and *complete activation*. The first event activation refers to the first event in the event relationship. For instance, e_{13} is the first event activation for the event relationship described in Fig. 6(b).

Next, the first object type activation refers to the last event of the object type that occurs the earliest among other object types. For example, e_{15} is the

first object type activation as it is the last event of the *item* object type, which occurs earlier than the last event of the *package* object type. Finally, the complete activation refers to the last event among the preceding events, i.e., e_{20}.

Using these concepts, we explain each object-centric performance measure as follows. Figure 6(c) shows an example of object-centric performance measures related to e_{21} (*pack items*).

- *Sojourn time* is the time difference between the target event and the complete activation.
- *Synchronization time* is the time difference between the complete activation and the first event activation.
- *Flow time* is the time difference between the target event and the first event activation.
- *Lagging time for an object type* is the time difference between the last event of the object type and the first object type activation.
- *Pooling time for an object type* is the interval between the last event of the object type and the first event of the object type.

6 Application and Discussion

This section presents a tool that implements the proposed conformance checking and performance analysis. Next, we conduct a use case study on a real-life process using the tool. Finally, we discuss the utility and limitations of the proposed method.

6.1 Implementation: *Explori*

The approach described in this paper has been implemented as an open-source web application named *Explori*, based on the Python library OCPA [8]. The source code and manual are available at https://github.com/gyunamister/Explori. The tool consists of four functional components: *event log management*, *process discovery*, *conformance checking*, and *performance analysis*.

Event Log Management. This component aims to support users to manage and interact with Object-Centric Event Logs (OCELs). It allows users to view available OCELs, delete existing OCELs, upload new OCELs, and select an OCEL to start a new analysis. Users can upload an OCEL in either CSV or JSONOCEL formats. After the file is uploaded, it will be displayed in the list of available OCELs. The deletion of an OCEL will remove all associated information, such as the cached analysis results.

Process Discovery. This component aims to discover an OC-DFG from a selected OCEL. Figure 7(a) is the OC-DFG that is discovered from an uploaded OCEL. Each node in the OC-DFG represents an activity that occurred in the

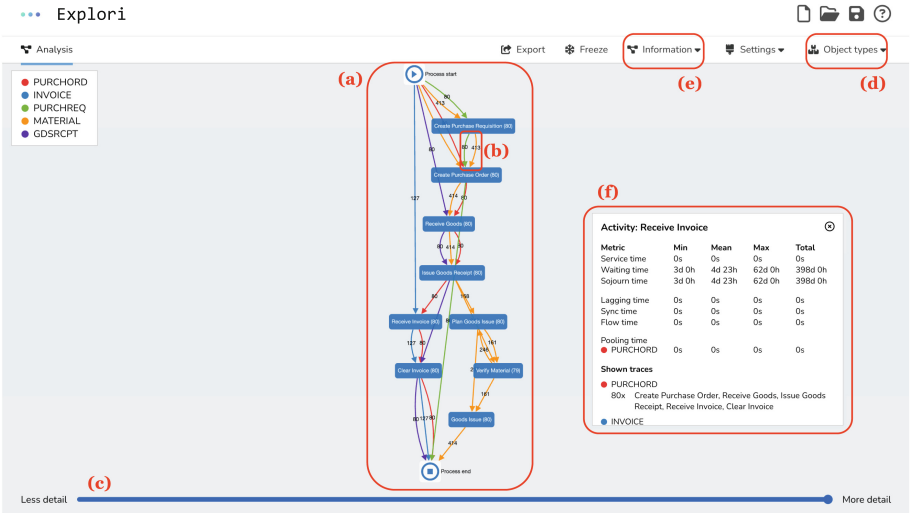

Fig. 7. A screenshot of *Explori* (Color figure online)

event log, while the edges indicate the directly-follows relation between the source activity and target activity. Multiple edges between two nodes signify the involvement of multiple object types in both activities, with each edge color representing a specific object type. For instance, Fig. 7(b) indicates the involvement of *MATERIAL* (represented as the yellow-colored edge) and *PURCHREQ* (represented as the green-colored edge) in *Create Purchase Requisition* and *Create Purchase Order*. The graph can be panned, zoomed, and nodes can be repositioned for better visualization.

OCELs often contain a mix of typical process behavior and outlier cases, which can complicate the model. To filter the graph and display only the most frequent behavior, users can adjust *threshold slider* at the bottom of the page (cf. Fig. 7(c)). The threshold represents the proportion of objects shown in the model. Moreover, users can also filter the graph by selecting specific object types of interest through the "Object Types" button in the top right corner shown in Fig. 7(d).

Conformance Checking. This component aims to compute alignments. The information dropdown in the navigation bar (cf. Fig. 7(e)) allows users to display alignment information within the process model. Log moves are visualized as dotted edges, while model moves are visualized as double-line edges.

Performance Analysis. This component aims to compute various performance measures. Users can click on nodes and edges to display the information box that contains object-centric performance metrics for the selected element, as shown in Fig. 7(f). The performance metrics include the flow, sojourn, synchronization, pooling, and lagging time.

6.2 Real-Life Use Case: Loan Application Process

Using the implementation, we analyze a real-life loan application process of a Dutch Financial Institute [13]. The process encompasses two types of objects: *applications* and *offers*, where an application may include multiple offers. Figure 8(a) depicts a process model of 1,317 applications and 4,457 offers that describe the cancellation of the applications under various scenarios. The process initiates when a customer submits an application, which, upon acceptance, leads to the bank generating and communicating loan offers to the customer, followed by the eventual cancellation of both the application and any associated offers.

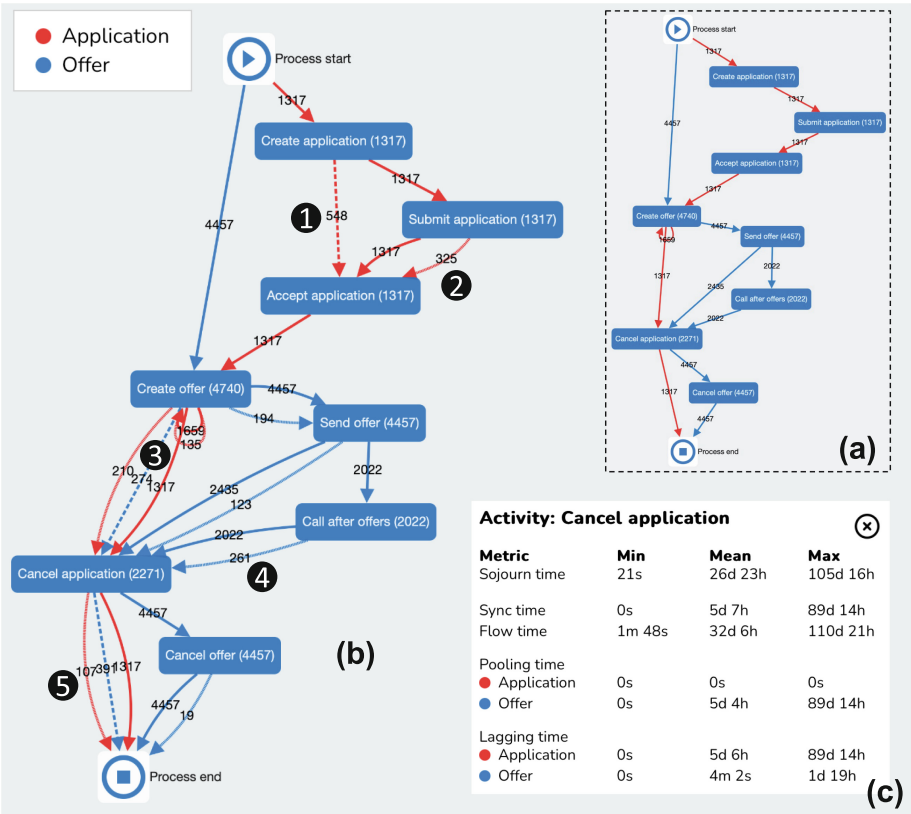

Fig. 8. (a) A OC-DFG representing the loan application process, (b) conformance checking visualization, and (c) performance analysis of *cancel application* activity

Conformance Checking. A conformance check was performed using an event log comprising 31,010 events related to the 2,302 applications and 5,128 offers. Figure 8(b) visualizes the conformance checking outcomes. Model moves at ① indicate the omission of the *submit application* activity for 548 applications,

attributed to in-person customer visits negating the need for this activity. Log moves at ② highlight 325 applications where an additional *handle leads* activity was necessary prior to acceptance. Model moves at ③ show 274 offers being canceled immediately after creation, bypassing the *send offer* and *call after offers* activities. Log moves at ④ and ⑤ illustrate additional activities required for 261 offers (i.e., *return offer*) and 107 applications (i.e., *call incomplete files*), respectively, following cancellation.

Performance Analysis. The cancellation process for applications engages a variable number of offers along with their corresponding applications. As per the process model, three activities precede the action of canceling applications. Within the lifecycle of applications, the preceding activity is consistently *create offer*, whereas the preceding activities for offers encompass *create offer*, *send offer* and *call after offers* activities.

The average sojourn time, i.e., the duration from the initiation of the preceding activities until the cancellation of an application, is approximately 26 days and 23 h. Thus, it suggests that, on average, about 26 days elapse to cancel applications following the execution of *create offer*, *send offer*, and *call after offers*. The synchronization time, i.e., the duration between completing the first and last of these activities before cancelling an application, averages at 5 days and 7 h, suggesting a coordination time taken for completing all necessary prerequisites.

Furthermore, the average pooling time for offers, defined as the period from when the first to the last offer is prepared for cancellation, is 5 days and 6 h. The average lagging time of applications at 5 days and 6 h, contrasts with a notably shorter lagging time of offers, merely 4 min. This indicates that applications are typically cancelled only after consolidating all related offers.

6.3 Discussion

The use case study conducted offers valuable insights into the practical application of OC-DFGs for conformance checking and performance analysis in a real-life loan application process. The application of our approach has demonstrated its utility in simplifying complex process models and making them more accessible for analysis.

However, the proposed approach has limitations that stem from the "decompose and recombine" paradigm, particularly in the recombination phase, where a coherent global alignment is needed. Currently, the approach lacks a systematic way to ensure that shared transitions across local alignments are treated consistently. This can lead to incoherencies in the global view of the process when different object views suggest different local alignments of the same transition. For instance, consider the transition *cancel application*, which occurs in the life cycle of both offers and applications. In the local alignment for the *offer*, we might decide to model skip the *cancel application*. However, this same transition, when viewed from the *application* perspective, could be critical and thus included in the alignment. This discrepancy leads to an incoherent global view

where the offer is canceled, but the application is not, resulting in a conflicting representation of the process.

Such inconsistencies are particularly problematic in performance analysis. For example, if the *cancel application* is skipped in one object view but not in another, it would be unclear how to calculate metrics such as synchronization time related to *cancel application* activity within the broader context of the application process.

Additionally, the simplification of synchronization semantics in the translation from DFGs to workflow nets is a limitation. This overlooks the dynamics of concurrent processes and may lead to a misrepresentation of performance metrics that are subject to the sequencing of interdependent activities. For instance, in our case study involving the loan application process, activities such as *send* and *call* may occur in parallel. However, if we were to represent this scenario in a simplified DFG, the concurrency will be depicted as a series of sequential steps.

This sequential representation could significantly skew performance analysis. Suppose *call* typically takes longer than *send*. In that case, the time between the start of *send* and its completion is not just the execution time of that single activity but also includes the waiting time until *call* is completed.

To address these challenges, a systematic approach should be developed for the recombination phase, which can reconcile the local alignments into a coherent global alignment, thereby aligning shared transitions consistently across all object views. Moreover, future work should explore simple, but richer process modeling formalisms that can capture the concurrent nature of business processes. This could involve incorporating elements from other process modeling languages that include explicit constructs for concurrency and synchronization.

7 Conclusion

In this paper, we address the gap in the literature concerning OC-DFGs for conformance checking and performance analysis in object-centric process mining. OC-DFGs, while noted for their simplicity and practical applications, have previously lacked comprehensive support for these essential tasks. We developed and presented an approach that utilizes OC-DFGs for both conformance checking and performance analysis. By translating each directly-follows graph in an OC-DFG to a workflow net, we compute alignments for improved conformance checking. Additionally, we utilize these alignments for performance analysis, computing object-centric performance metrics. The proposed approach is fully implemented as a web application and is further validated through a case study on a real-life loan application process.

References

1. van der 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
2. van der Aalst, W.M.P.: Object-centric process mining: the next frontier in business performance. White paper, Celonis (2023)
3. van der Aalst, W.M.P., Adriansyah, A., van Dongen, B.F.: Replaying history on process models for conformance checking and performance analysis. WIREs Data Min. Knowl. Discov. **2**(2), 182–192 (2012)
4. van der Aalst, W.M.P., Berti, A.: Discovering object-centric Petri nets. Fundam. Informaticae **175**(1–4), 1–40 (2020)
5. van der Aalst, W.M.P., Li, G., Montali, M.: Object-centric behavioral constraints. CoRR abs/1703.05740 (2017)
6. van der Aalst, W.M.P., De Masellis, R., Di Francescomarino, C., Ghidini, C.: Learning hybrid process models from events - process discovery without faking confidence. In: Carmona, J., Engels, G., Kumar, A. (eds.) BPM 2017. LNCS, vol. 10445, pp. 59–76. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65000-5_4
7. Adams, J.N., van der Aalst, W.M.P.: Precision and fitness in object-centric process mining. In: ICPM 2021, pp. 128–135 (2021)
8. 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)
9. Barenholz, D., Montali, M., Polyvyanyy, A., Reijers, H.A., Rivkin, A., van der Werf, J.M.E.M.: There and back again - on the reconstructability and rediscoverability of typed Jackson nets. In: Gomes, L., Lorenz, R. (eds.) PETRI NETS 2023. LNCS, vol. 13929, pp. 37–58. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-33620-1_3
10. Berti, A., van der Aalst, W.: Extracting multiple viewpoint models from relational databases. In: Ceravolo, P., van Keulen, M., Gómez-López, M.T. (eds.) SIMPDA 2018-2019. LNBIP, vol. 379, pp. 24–51. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-46633-6_2
11. Cohn, D., Hull, R.: Business artifacts: a data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull. **32**(3), 3–9 (2009)
12. Diba, K., Batoulis, K., Weidlich, M., Weske, M.: Extraction, correlation, and abstraction of event data for process mining. WIREs Data Min. Knowl. Discov. **10**(3), e1346 (2020)
13. van Dongen, B.: BPI challenge 2017 (2017). https://doi.org/10.4121/UUID:5F3067DF-F10B-45DA-B98B-86AE4C7A310B. https://data.4tu.nl/articles/_/12696884/1
14. Fahland, D.: Describing behavior of processes with many-to-many interactions. In: Donatelli, S., Haar, S. (eds.) PETRI NETS 2019. LNCS, vol. 11522, pp. 3–24. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21571-2_1
15. Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.M.P.: Conformance checking of interacting processes with overlapping instances. In: Rinderle-Ma, S., Toumani, F., Wolf, K. (eds.) BPM 2011. LNCS, vol. 6896, pp. 345–361. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23059-2_26
16. Leemans, S.J.J., Poppe, E., Wynn, M.T.: Directly follows-based process mining: exploration & a case study. In: ICPM 2019, pp. 25–32. IEEE (2019)

17. Li, G., de Carvalho, R.M., van der Aalst, W.M.P.: Automatic discovery of object-centric behavioral constraint models. In: Abramowicz, W. (ed.) BIS 2017. LNBIP, vol. 288, pp. 43–58. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59336-4_4
18. Lu, X.: Artifact-centric log extraction and process discovery. Unpublished Master's thesis, Eindhoven University of Technology (2013)
19. Montali, M., Rivkin, A.: DB-Nets: on the marriage of colored Petri nets and relational databases. In: Koutny, M., Kleijn, J., Penczek, W. (eds.) Transactions on Petri Nets and Other Models of Concurrency XII. LNCS, vol. 10470, pp. 91–118. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-55862-1_5
20. Park, G., van der Aalst, W.M.P.: Monitoring constraints in business processes using object-centric constraint graphs. In: Montali, M., Senderovich, A., Weidlich, M. (eds.) ICPM 2022. LNBIP, vol. 468, pp. 479–492. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-27815-0_35
21. Park, G., Adams, J.N., van der Aalst, W.M.P.: OPerA: object-centric performance analysis. In: Ralyté, J., Chakravarthy, S., Mohania, M., Jeusfeld, M.A., Karlapalem, K. (eds.) ER 2022. LNCS, vol. 13607, pp. 281–292. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-17995-2_20
22. Park, G., Comuzzi, M., van der Aalst, W.M.P.: Analyzing process-aware information system updates using digital twins of organizations. In: Guizzardi, R., Ralyté, J., Franch, X. (eds.) RCIS 2022. LNBIP, vol. 446, pp. 159–176. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-05760-1_10
23. Weijters, A., van der Aalst, W., Alves De Medeiros, A.: Process mining with the HeuristicsMiner algorithm. BETA publicatie: working papers (2006)