Transforming Event Knowledge Graph to Object-Centric Event Logs: A Comparative Study for Multi-dimensional Process Analysis

Shahrzad Khayatbashi[1(✉)], Olaf Hartig[1], and Amin Jalali[2]

[1] Linköping University, Linköping, Sweden
{shahrzad.khayatbashi,olaf.hartig}@liu.se
[2] Stockholm University, Stockholm, Sweden
aj@dsv.su.se

Abstract. Process mining has significantly transformed business process management by introducing innovative data-based analysis techniques and empowering organizations to unveil hidden insights previously buried within their recorded data. The analysis is conducted on event logs structured by conceptual models. Traditional models were defined based on only a single case notion, e.g., order or item in the purchase process. This limitation hinders the application of process mining in practice for which new data models are developed, a.k.a, multi-dimensional Event Knowledge Graph (EKG) and Object-Centric Event Log (OCEL). While several tools have been developed for OCEL, there is a lack of process mining tooling around the EKG. In addition, there is a lack of comparison about the practical implication of choosing one approach over the other. To fill this gap, the contribution of this paper is threefold. First, it defines and implements an algorithm to transform event logs represented as EKG to OCEL. The implementation is then used to transform five real event logs based on which the approach is evaluated. Second, it compares the performance of analyzing event logs represented in these two models. Third, it reveals similarities and differences in analyzing processes based on event logs represented in these two models. The results highlight ten important findings, including different approaches in calculating directly-follows relations when analyzing filtered event logs in these models and issues that need to be considered in analyzing event lifecycle and inter-log relations using OCEL.

Keywords: Event Knowledge Graph · Object-Centric Event Log · Object-Centric Process Mining · Neo4j · Graph database

1 Introduction

Business process analysis is important in modern organizations because it enables comprehension, optimization, and enhancement of operational processes based on recorded data [38]. These processes are complex due to the complex nature of the business domain. To address this complexity, log file formats and standards have emerged as conceptual models that capture the essential information required to support the analysis [2,14,21,33]. These conceptual models drive the development of algorithms and facilitate the processing and analysis of recorded data.

Process mining is a research area that facilitates data-driven business process analysis based on recorded event logs [38]. Log files are crucial for analyzing business processes. Thus, extensive efforts have been made to define conceptual models, in the forms of log file formats and standards, that enable the analysis of recorded data using different software systems [21,23,34]. These formats and standards ensure compatibility and interoperability across various systems while providing a consistent and structured format for recording process-related information.

Traditionally, event log formats assume a single case notion as an obligatory element based on which the rest of the information could be correlated. For example, a purchase order event log could be extracted using either the order or the item notions, while the process contains both objects as potential cases. In reality, business processes deal with different perspectives, which may require several case notions. Hence, restricting log formats to a single case notion limits the applicability of process mining in practice.

To circumvent this limitation, researchers and practitioners flattened the recorded event log to perform process analysis, which introduces its limitations, including false behavior and false analysis results [16] (which result from so-called divergence and convergence problems [39]). For example, one order may contain many items. In the log extraction, if we consider the “item” as the case notion, events like “create order” must be repeated for each item. The mapping of events based on one case notion, like this example, is called flattening. One consequence is that we will get false statistics when retrieving the number of orders which are created. If the log is flattened around the “order” case notion, the relation between the “select item” and “approve item” in the process can be lost because all items can be stored around one order resulting in losing information about relations between items. The lack of these relations could introduce loops between the activities of these two events in discovering process models, which is considered false behavior. These issues compromise the accuracy of the analysis [39].

The Object-Centric Event Log (OCEL) [21] has been proposed to address the limitation of having only one case notion when extracting log files, and it is part of a new and emerging paradigm in process mining called Object-Centric Process Mining (OCPM) [39]. This paradigm aims to support analyzing business processes considering multiple case notions that require developing algorithms, techniques, and methods to support multi-dimensional process analysis. Although OCPM has started recently, due to the highly relevant problem that it targets, several algorithms, tools, and libraries have been developed to support such analysis, e.g., [3,4,6,11,26,34,35,39,40]. This development can also be observed in commercial tools like Celonis[1], showing the relevancy of the problem in practice.

Another recent alternative to recording event logs is knowledge graphs, which unleash their power within information systems, showcasing their ability to support various data sources, scalability, semantic reasoning, and adaptable schema evolution [24]. Thus, it is recently used to record and process event logs with multiple case notions, called multi-dimensional Event Knowledge Graph (EKG) [14]. However, the lack of process mining tools for analyzing EKG limits the practical application of this approach. Additionally, there is a lack of comparative analysis in terms of performance, strengths, weaknesses, limitations, and differences between the processing of data rep-

[1] https://www.celonis.com/.

resented using these two approaches (EKG and OCEL). Therefore, this paper aims to address the following research questions:

RQ1) How can an event knowledge graph be transformed into an object-centric event log?
RQ2) How does the performance of processing event knowledge graph compare to processing object-centric event log in process mining?
RQ3) What are the differences and similarities in applying process mining on an event knowledge graph compared to an object-centric event log?

To answer the first research question, we define an algorithm that transforms it into a set of OCELs. We implemented an algorithm as a part of a Python library, called `neo4pm`, that can be used to perform the transformation. In this paper, we use this implementation to transform five real EKGs into OCEL files, which are available publicly [28–32]. In addition, we compare similarities and differences in analyzing processes based on event logs represented in EKG and transformed OCELs, which helped us answer the third research question.

The structure of the paper is as follows. Section 2 gives an overview of related work. Section 3 provides preliminaries which are used in Sect. 4, where we define the algorithm formally. Section 5 reports the results and discusses the findings. Finally, Sect. 6 concludes the paper by giving future direction.

2 Related Work

In this section, we provide an overview of the research that offers tool support for processing event logs represented in multi-dimensional Event Knowledge Graph (EKG) and Object-centric Event Log (OCEL). Table 1 summarizes the process mining tools developed for OCEL and EKG. The table categorizes the level of support into eight use cases: transformation, exploration, monitoring, performance analysis, discovery, conformance checking, enhancement, and predictive process monitoring.

The tool support for EKG focuses on transforming traditional log files into the EKG data model [14]. A recent study has proposed a method for transforming OCEL to EKG [16]; however, the existing implementation does not yet support the transformation from the serialized standard OCEL files. Furthermore, there is a lack of support for EKG in other use cases. In contrast to EKG, the existing contributions to OCEL varies in different use cases. These categories are represented as columns in Table 1.

In the *transformation* use case, we have identified three sub-categories of transformation. Firstly, there are approaches focused on transforming traditional log to OCEL [37]. Secondly, there are methods for transforming data recorded in databases or Enterprise Resource Planning (ERP) systems to OCEL [10,42]. Lastly, there are techniques available for flattening OCEL to traditional log [11,21]. In the *exploration* use case, we have identified four sub-categories of exploration. This includes support for filtering events based on certain criteria [11], identifying concept drift in event data [8], supporting variant analysis on event logs [5,7], and splitting the log into several clusters based on similarity in underlying behaviour [26].

Table 1. Summary of studies providing tool support for OCEL or EKG

Approach	UC_1	UC_2	UC_3	UC_4	UC_5	UC_6	UC_7	UC_8
OCEL	[10,11,21,37,42]	[5,7,8,11,26]	[36]	[11,35,36]	[4,26,40]	[4,6,11]	[3,4]	[4,22]
EKG	[14,16]							

UC_1: Transformation, UC_2: Exploration, UC_3: Monitoring, UC_4: Performance Analysis, UC_5: Discovery
UC_6: Conformance Checking, UC_7: Enhancement, UC_8: predictive process monitoring

In the *monitoring* use case, Park and van der Aalst present a tool for monitoring object-centric constraints [36]. In the *performance analysis* use case, a tool called OC-PM is available for calculating the duration time of objects [11]. Additionally, performance metrics computation is supported by [36] and [35]. In the *discovery* use case, the discovery of object-centric Petri nets is supported by [40] and [4]. In addition, the discovery of Markov Directly-Follow Multigraphs is supported by [26] by extending the discovery of Markov Directly-Follow Graphs [27]. In the *conformance checking* use case, Berti and van der Aalst provide a tool for conformance checking [11]. Also, tool support is provided for calculating precision and fitness [4,6]. In the *enhancement* use case, tool support is provided for enhancing process models through feature extraction [3,4]. In the *predictive process monitoring* use case, Adams et al. [4] provide a tool for predictive monitoring, and Gherissi et al. [22] offer a tool for predicting the next event time, activity, and remaining sequence time.

3 Preliminaries

This section introduces the notions of the Event Knowledge Graph (EKG) and the Object-Centric Event Log (OCEL), which serve as the foundation for defining the transformation algorithm in Sect. 4. We explain the EKG definition using a running example, which will also be utilized to demonstrate the approach and algorithm in the subsequent sections of this paper.

Figure 1 illustrates a running example that is used to explain the components of EKG. The figure represents recorded information in an EKG for a fictitious business process involving a customer order (o1) with two items (i1 and i2). Orders and items are depicted as ovals (annotated with : Entity), while events are represented as rectangles (annotated with : Event). Each event has an activity name and a timestamp (e.g., Submit Order and 15 : 00 for e1, respectively). Some events have the performing resource (e.g., Elin for e3). The figure illustrates the chronological sequence of events: Submit Order, two instances of Check Availability (one for each item), and Pick Items. The following definitions will define the elements within this graph based on which we can define the transformation algorithm.

Definition 1 (Universes). We define the following universes to be used throughout the paper, some of which are adopted from [39]:

- $\mathbb{U}_{lbl}$ is an infinite set of strings representing labels,
- $\mathbb{U}_{att}$ is an infinite set of strings representing attribute names,
- $\mathbb{U}_{val}$ is an infinite set of strings representing attribute values containing the following disjoint subsets:

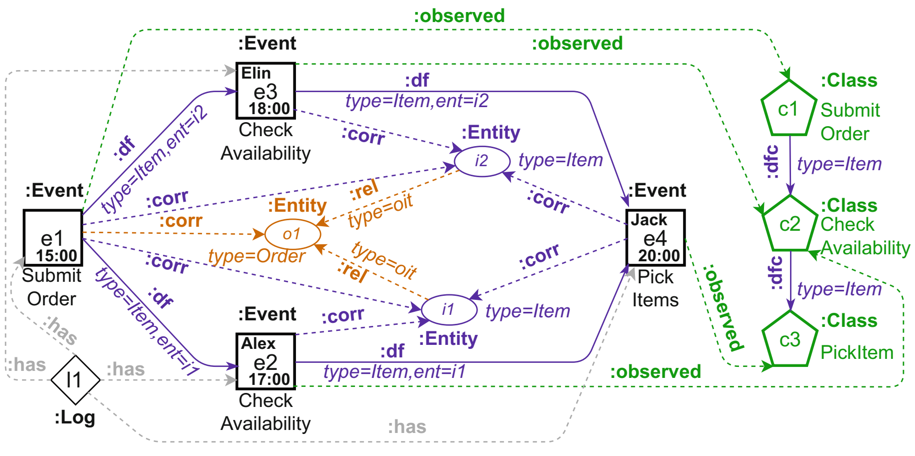

Fig. 1. Running example showing event log represented in an EKG

- $\mathbb{U}_{eid} \subset \mathbb{U}_{val}$ represents the universe of event identifiers,
- $\mathbb{U}_{time} \subset \mathbb{U}_{val}$ represents the universe of timestamps,
- $\mathbb{U}_{act} \subset \mathbb{U}_{val}$ represents the universe of activity names,
- $\mathbb{U}_{ot} \subset \mathbb{U}_{val}$ represents the universe of object types,
- $\mathbb{U}_{oid} \subset \mathbb{U}_{val}$ represents the universe of object identifiers,

- $type : \mathbb{U}_{oid} \rightarrow \mathbb{U}_{ot}$ is a function that assigns exactly one object type to each object identifier,
- $\mathbb{U}_{omap} = \{omap : \mathbb{U}_{ot} \rightarrow \mathcal{P}(\mathbb{U}_{oid}) \mid \forall_{ot \in dom(omap)} \forall_{oid \in omap(ot)}\ type(oid) = ot\}$ is the universe of all object mappings indicating which object identifiers are included per object type[2],
- $\mathbb{U}_{vmap} = \{vmap : \mathbb{U}_{att} \nrightarrow \mathbb{U}_{val}\}$ is the universe of value assignments,[3] and
- $\mathbb{U}_{event} = \mathbb{U}_{eid} \times \mathbb{U}_{act} \times \mathbb{U}_{time} \times \mathbb{U}_{omap} \times \mathbb{U}_{vmap}$ is the universe of events.

Definition 2 (Labeled Property Graph (LPG)). An LPG (adopted from [9,16]) is a tuple $G = (N, R, \gamma, \lambda, \rho)$, where:

- N and R are finite sets of nodes and relations, respectively,
- $\gamma : R \rightarrow N \times N$ is a total function assigning a pair of nodes (representing the source and target, respectively) to a relation,
- $\lambda : (R \cup N) \rightarrow \mathbb{U}_{lbl}$ is a total function assigning a label to a node or a relation,
- $\rho : (N \cup R) \times \mathbb{U}_{att} \nrightarrow \mathbb{U}_{val}$ is a partial function assigning a value to an attribute of a node or a relation.

Given an LPG $G = (N, R, \gamma, \lambda, \rho)$, we call $E = N \cup R$ the set of elements in the graph containing both nodes and relations. Considering a Label $l \in \mathbb{U}_{lbl}$, we write E^l to

[2] $\mathcal{P}(\mathbb{U}_{oid})$ is the powerset of the universe of object identifiers, i.e., objects types are mapped onto sets of object identifiers.

[3] $\mathbb{U}_{att} \nrightarrow \mathbb{U}_{val}$ is the set of all partial functions mapping a subset of attribute names onto the corresponding values.

denote the subset of E consisting of all the elements with Label l. Formally, we show this as $E^l = \{e \in E \mid \lambda(e) = l\}$. We use the same notation for the subsets N and R of E (e.g., N^l). Moreover, for every element $e \in E$ and every attribute name $a \in \mathbb{U}_{att}$, if $(e, a) \in dom(\rho)$, we write $e.a$ to refer to the value $v \in \mathbb{U}_{val}$ for which it holds that $\rho(e, a) = v$; if $(e, a) \notin dom(\rho)$, then $e.a$ denotes a special value $\perp$ that is not in $\mathbb{U}_{val}$.

Example 1. In Fig. 1, we can see ten nodes. One is annotated with e1, where we refer to it by n and its activity name by act in this example. Thus, we can say $\rho(n, act) =$ SubmitOrder representing that the activity name of this event is SubmitOrder. We can also write $n.act =$ SubmitOrder. As this node is labeled with Event, we can say $\lambda(n) =$ Event or $n \in N^{\mathsf{Event}}$. This node has a relation to another event annotated with e2. We refer to this event by n' and to its relation to n' by r. We can say $\gamma(r) = (n, n')$. This relation is labeled with df, so $\lambda(r) =$ df or $r \in R^{\mathsf{df}}$.

After defining LPG, we now introduce a special kind of LPG that uses a specific schema named Event Knowledge Graph. We define the schema as $\mathcal{S} = \{(\mathsf{has}, (\mathsf{Log}, \mathsf{Event})), (\mathsf{observed}, (\mathsf{Event}, \mathsf{Class})), (\mathsf{rel}, (\mathsf{Entity}, \mathsf{Entity})), (\mathsf{df}, (\mathsf{Event}, \mathsf{Event})), (\mathsf{dfc}, (\mathsf{Class}, \mathsf{Class}))\}$. This schema specifies the possible label of the source and the target node in each relation based on the relation's label. Each member of the set is a tuple, where the first element indicates a possible relation's label, and the second element indicates the label of source and target nodes, respectively. In the Event Knowledge Graph definition, we restrict the universe of labels as $\mathbb{U}_{lbl} = \bigcup_{(l,(s,t)) \in \mathcal{S}} \{l\} \cup \{s\} \cup \{t\}$, meaning that $\mathbb{U}_{lbl} = \{$Event, Entity, Class, Log, observed, has, rel, df, dfc, corr$\}$. Note that an EKG can have multiple nodes labeled as Log, meaning that it can record events related to multiple logs in one graph.

Definition 3 (Event Knowledge Graph (EKG)). An EKG is an LPG $G = (N, R, \gamma, \lambda, \rho)$, that has the following properties[4].

a) $\forall_{e \in N^{\mathsf{Event}}} (e.id \in \mathbb{U}_{eid} \wedge e.act \in \mathbb{U}_{act} \wedge e.time \in \mathbb{U}_{time})$ indicating that each node with the label Event has attributes called id, act, and $time$ with the value of an event identifier, an activity name, and a timestamp, respectively,
b) $\forall_{e \in N^{\mathsf{Entity}}} (e.id \in \mathbb{U}_{oid} \wedge e.type \in \mathbb{U}_{ot})$ indicating that each node with the label Entity has an attribute called id and $type$ with the value of an object identifier and object type, respectively,
c) The relations between nodes can be specified as $\forall_{(l,(s,t)) \in \mathcal{S},\ r \in R \text{ with } \gamma(r) = (e, e')}\ (e \in N^s \wedge e' \in N^t) \Leftrightarrow r \in R^l$ indicating that a relation can be labeled as specified in schema if and only if the source and target nodes are labeled accordingly,
d) $\forall_{r \in R^{\mathsf{rel}}}\ r.type \in \mathbb{U}_{ot} \cup \{\mathsf{Reified}\}$ indicating that each relation with the label rel has attributes called $type$ with the value of an object type or a special value called Reified. The Reified type is used to model the relation between derived entities to other entities.

We keep the definition of EKG to a minimum in this paper without elaborating on detailed properties that are not needed for the transformation algorithms. For example, we omit details on properties that should be held by df and dfc relations. More details can be found in [14,16].

[4] The definition is aligned with definitions in [14,16].

Example 2. Our running example graph fulfills the properties stated in Definition 3 (a-b). As required by Definition 3 (a), each event in our graph has an event identifier (e.g., e1), an activity name (e.g., SubmitOrder for e1), a timestamp (e.g., 15 : 00 for e1). Also, all entities have an identifier as well as a type as required by Definition 3 (b), e.g., the mustard-colored oval has an identifier with the value of o1 and type of Order.

Our running example graph fulfills the properties stated in Definition 3 (c-d). As required by Definition 3 (c), every relation that its source and target are labeled with Log and Event respectively are labeled with has, e.g., the relation between l1 and e1. The same applies to other relations such as observed, rel, df, and dfc, where their source and target nodes are labeled as indicated in the defined set. As required by Definition 3 (d), every relation which is labeled by rel has an attribute named type, e.g., the relation between i1 and o1 which has a type with the value of oit.

The following two definitions are adopted from [39] describing an OCEL, the target format to which we will transform the described EKG.

Definition 4 (Event Projection (adopted from [39])). An *event* e is a tuple $(eid, act, time, omap, vmap)$ where $eid \in \mathbb{U}_{eid}$, $act \in \mathbb{U}_{act}$, $time \in \mathbb{U}_{time}$, $omap$ is an object mapping, and $vmap$ is a value assignment. For each such event $e = (eid, act, time, omap, vmap)$, we write $\pi_{eid}(e)$ to denote eid, $\pi_{act}(e)$ denotes act, $\pi_{time}(e)$ to denote $time$, $\pi_{omap}(e)$ to denote $omap$, and $\pi_{vmap}(e)$ denotes $vmap$.

Definition 5 (Object-Centric Event Log (OCEL) [39]). An event log L is a pair $(E, \preceq_E)$ with $E \subseteq \mathbb{U}_{event}$ and $\preceq_E \subseteq E \times E$ such that:

- $\preceq_E$ defines a partial order (reflexive, antisymmetric, and transitive),
- $\forall_{e_1,e_2 \in E}\ \pi_{eid}(e_1) = \pi_{eid}(e_2) \Rightarrow e_1 = e_2$, and
- $\forall_{e_1,e_2 \in E}\ e_1 \preceq_E e_2 \Rightarrow \pi_{time}(e_1) \leqslant \pi_{time}(e_2)$.

4 Approach

This section introduces a transformation algorithm that enables transforming an Event Knowledge Graph (EKG) into a set of Object Centric Event Logs (OCELs), addressing RQ1. In this algorithm's definition, the following Design Choices (DC) have been made:

DC1. EKG with Multiple Logs**:** The algorithm converts an EKG with multiple logs (i.e., an EKG with multiple nodes with the label Log) into a set of OCEL files. This choice aligns with the OCEL standard, allowing one global log element per file [21]. An alternative option would be to include all of events in one log file and mark events related to a log file using a $vmap$. However, this alternative deviates from the standard, as the $vmap$ value does not represent logs according to the standard. Our approach can easily support the second design choice by merging the generated OCELs into one with a new $vmap$ indicating the log file.

DC2. Event Lifecycles**:** Unlike XES, OCEL does not explicitly define event lifecycles which specifies events representing different states of an operational task in a business process. As a result, we chose to omit to transform event classes (representing lifecycles in EKG) to OCEL. Event classes in EKG can be related to multiple lifecycle states, and the explicit definition of the event lifecycle in a log file can enable the development of lifecycle-aware algorithms, similar to algorithms developed for XES. If OCEL is extended to support lifecycles in the future, our transformation algorithm can easily include the transformation logic. As an alternative design choice, it is possible to transform the lifecycle as event attributes or related objects, yet this still will not help in the definition of lifecycle-aware algorithms as this information needs to be explicitly supported by standards so that algorithms can take them into account.

***DC3. Relations Between Entities*:** The algorithm also omits to transform EKGs' reified entities. OCEL does not support these relations, leaving them out of the transformation process.

By making these design choices, the algorithm ensures compliance with the current version of the OCEL standard while accommodating potential future extensions for lifecycle support and other entity transformations. Algorithm 1 describe the transformation logic, where the input is an EKG, and the output is a set of OCELs.

Algorithm 1. Converting EKG to OCELs

```
1: **Input:** A event knowledge graph $G = (N, R, \lambda, \gamma, \rho)$
2: **Output:** A set of OCELs $\mathcal{O}$
3: **Begin**
4: $\mathcal{O} \leftarrow \emptyset$
5: $\not{\mathcal{R}} \leftarrow N^{Entity} \setminus \{n \in N^{Entity} \mid \exists_{n' \in N^{Entity}} (n, n') \in R^{Rel} \wedge \exists_{r \in R^{Rel}} \gamma(r) = (n, n') \wedge r.type = \text{Reified}\}$
6: **for each** $l \in N^{Log}$ **do**
7:     $E \leftarrow \emptyset$
8:     **for each** $e \in N^{Event}$ **do**
9:         $omap \leftarrow \emptyset$
10:        $vmap \leftarrow \emptyset$
11:        **if** $\exists_{r \in R^{Has}} \gamma(r) = (l, e)$ **then**
12:            $\mathcal{E}_{\not{\mathcal{R}}} \leftarrow \{n \in \not{\mathcal{R}} \mid \forall_{r \in R^{Corr}} \gamma(r) = (e, n)\}$
13:            $\mathcal{OT} \leftarrow \bigcup_{n \in \mathcal{E}_{\not{\mathcal{R}}}} n.type$
14:            **for each** $ot \in \mathcal{OT}$ **do**
15:                $omap(ot) \leftarrow \bigcup_{n \in \mathcal{E}_{\not{\mathcal{R}}} \wedge (n.type = ot)} n.id$
16:            **end for**
17:            **for each** $att \in \mathbb{U}_{att} \setminus \{id, act, time\}$ **do**
18:                **if** $e.att \neq \perp$ **then**
19:                    $vmap(att) \leftarrow e.att$
20:                **end if**
21:            **end for**
22:            $E \leftarrow E \cup \{(e.id,\ e.act,\ e.time,\ omap,\ vmap)\}$
23:        **end if**
24:    **end for**
25:    $\preceq_E \leftarrow \{(e, e') \mid e, e' \in E \wedge e \neq e' \wedge \pi_{time}(e) \leqslant \pi_{time}(e')\}$
26:    $\mathcal{O} \leftarrow \mathcal{O} \cup \{(E, \preceq_E)\}$
27: **end for**
```

Here, we elaborate on this algorithm. Line 5 assigns the set of non-reified entities to $\mathcal{R}$. In our running example, $\mathcal{R} = \{i1, i2, o1\}$. We exclude reified entities in EKG as OCEL does not capture relations among entities. Thus, we only need the set of non-reified entities. Then, the algorithm starts iterating around each log node. It defines a set for capturing all events of the log, i.e., E (line 7). Then, for each event, it defines two empty functions (lines 9 and 10) that will be configured accordingly: if the log has a has relation to the event, the algorithm i) retrieves all non-reified entities to which the event has a $corr$ relation and assigns them to $\mathcal{E}_{\mathcal{R}}$ (line 12), and ii) retrieves the type of all retrieved entities and assigns them to $\mathcal{OT}$ (line 13). If we look at our running example, this algorithm sets the mentioned variables for $\mathsf{e1}$ accordingly: $\mathcal{E}_{\mathcal{R}} = \{i1, i2, o1\}$, $\mathcal{OT} = \{\mathsf{Order}, \mathsf{Item}\}$.

Then, the algorithm sets $omap$ and $vmap$ through two loops. The first loop configures the $omap$ function by relating each retrieved object type to a set of related object identifiers (line 15). This means that, $omap(\mathsf{Order}) = \{o1\}$ and $omap(\mathsf{Item}) = \{i1, i2\}$ for $\mathsf{e1}$. The second loop configures the $vmap$ function by assigning all event's attributes (except for id, act, and $time$) to $vmap$ (line 19). For event $\mathsf{e1}$, $vmap$ will be empty as the event has no other attributes. However, if we consider $\mathsf{e3}$, $vmap(\mathsf{Resource}) = \mathsf{Elin}$.

Finally, the algorithm updates the variable capturing all events within the processing log, i.e., E (line 22). For our example when processing $\mathsf{e1}$, $E = (\mathsf{e1}, \mathsf{SubmitOrder}, \mathsf{15{:}00}, \{omap(\mathsf{Order}) = \{o1\}, omap(\mathsf{Item}) = \{i1, i2\}\}, \{\})$. Iterating all these steps will produce an OCEL, and line 26 retrieves the set of OCELs transformed from the EKG.

5 Evaluation

This section presents the evaluation results of comparing transformed OCEL with EKG. Through this evaluation, we analyze the differences and similarities between these two approaches. A comparative performance analysis is also conducted between EKG and OCEL, further investigating the disparities and similarities between these approaches.

5.1 Data Processing

The transformation algorithm was implemented as part of an open-source Python library[5], called `neo4pm`[6]. For evaluation, EKG was transformed to OCEL using our implemented algorithm, and the transformed logs are available publicly at [28–32]. Due to the large size of the log files, the transformation was performed on a server. Subsequently, EKG and OCEL were evaluated and compared on a laptop, replicating the environment typically used by analysts.

Data Transformation: To evaluate our approach, we transformed five open-access real-world EKG: BPIC14 [17], BPIC15 [18], BPIC16 [19], BPIC17 [20], and BPIC19 [15]. As a result, we obtained nine OCELs (one OCEL file for each EKG, except for BPIC15, which produced five OCEL files).

[5] The library can be installed using `!pip install neo4pm`.

[6] The source code is available at https://github.com/neo4pm/neo4pm.

Evaluation Setup: For the evaluation setup, we used a laptop with the following specifications: two 6-core Intel Core i9 CPUs running at 2.90 GHz, 32 GB of RAM, a 1 TB HDD, and a 64-bit Windows 11 Enterprise operating system. Docker (v.4.17.1) was installed on the laptop to host the running evaluations. Neo4j (community edition 3.5) and PM4Py (v.2.7.3) were utilized for the evaluations [12, 13].

5.2 Information Preserving Evaluation

Table 2 illustrates the information-preserving evaluation results, comparing the number of different elements in the EKG and the transformed OCEL. This table captures the count of Logs, Events, non-reified Entities (objects in OCEL), Classes (activity names in OCEL), Observed relations (showing the activity lifecycles), corr relations, and direct-follow relations (df), shown as columns in the table. The rows represent the evaluation result for different BPICs. BPIC15 consists of multiple logs, so the numbers are given in detail for each log for OCEL, and they are aggregated to be compared with EKG. In the subsequent discussion, we will explore the differences observed in these elements.

As can be seen in the table, information preservation is evident for all BPICs except BPIC15 and BPIC17, which exhibit some differences compared to the others. BPIC15 involves process data associated with multiple log files, leading to the transformation of EKG into multiple OCEL log files (as followed based on DC1.). EKG for BPIC17, on the other hand, captures information regarding the lifecycle of each event. Further elaboration on these differences will be provided below.

Table 2. Information preserving evaluation result

		# Log	# Event	# Entity*	# Class	# observed	# corr*	# df
BPIC 14	OCEL	1	690,622	228,885	330	690,622	2,732,213	2,503,328
	EKG	1	690,622	228,885	330	690,622	2,732,213	2,503,328
BPIC 15	$OCEL_1$	1	52,217	1,269	289	52,217	208,868	207,599
	$OCEL_2$	1	443,54	859	304	44,354	177,416	176,557
	$OCEL_3$	1	59,681	1,465	277	59,681	238,724	237,259
	$OCEL_4$	1	47,293	1,084	272	47,293	189,172	188,088
	$OCEL_5$	1	59,083	1,202	285	59,083	236,332	235,130
	OCEL	Sum:	262,628	**5,879**	**1,427**	262,628	1,050,512	**1,044,633**
	EKG	5	262,628	**5,862**	**356**	262,628	1,050,512	**1,044,650**
BPIC 16	OCEL	1	7,360,146	748,913	620	7,360,146	36,430,880	35,681,967
	EKG	1	7,360,146	748,913	620	7,360,146	36,430,880	35,681,967
BPIC 17	OCEL	1	1,202,267	106,162	**26**	**1,202,267**	2,404,534	2,298,372
	EKG	1	1,202,267	106,162	**92**	**2,404,534**	2,404,534	2,298,372
BPIC 19	OCEL	1	1,595,923	330,685	42	1,595,923	5,984,602	5,653,917
	EKG	1	1,595,923	330,685	42	1,595,923	5,984,602	5,653,917

#: Number of, $*$: Non-Reified, $OCEL_n$: n^{th} sublog

Differences in BPIC 15: Three differences can be observed when comparing the EKG with the generated OCELs, i.e., the difference in the total number of Entities (referred to as Objects in OCEL), Classes, and directly-follows relations.

The difference in the total number of Entities and Classes is the result of splitting the data to multiple OCELs for BPIC15, as shown in the Table 2, which is due to the limitation of OCEL to capture multiple logs. Consequently, some entities are repeated across different log files, leading to double counting when aggregating the numbers. The same applies to the count of classes. However, these differences do not affect the analysis, as each OCEL represents a subset of the log.

An additional disparity lies in the number of directly-follows relations. These relations significantly impact process discovery and conformance-checking algorithms, warranting a detailed analysis to ascertain the reasons behind the difference. We identified 860 missing directly-follows relations after transforming the EKG BPIC15 into OCEL. Notably, this number does not align with the difference reported in the table. The reason is that directly-follows relations need to be calculated at runtime for a given OCEL. This is different from EKG which materializes these relations. Hence, additional directly-follows relations may be inferred in OCEL that were not present in the source EKG. To illustrate this case, Fig. 2 presents a sub-graph extracted from the EKG for BPIC15, which allows us to delve deeper into the aforementioned issue.

In Fig. 2, we can observe two types of directly-follows (DF) relations: intra-log and inter-log directly-follows relations. The two red DF flows represent intra-log relations, indicating that these relations exist among events within a single log, i.e., events related to BPIC15_1. Additionally, there is one intra-log directly-follows relation involving events related to BPIC15_3, denoted by a thin mustard-colored (DF) relation. The figure's two thick mustard-colored DF relations represent inter-log directly-follows relations. These relations occur when the source and target events are associated with different logs in the graph.

Figure 3 showcases the directly-follows relations discovered using PM4Py python library [12] with the transformed OCEL specifically for BPIC15_1. Several similarities and differences can be observed in comparison to Fig. 2. i) The two intra-log directly-follows relations for BPIC15_1 are preserved in the transformed OCEL. ii) However, the two inter-log directly-follows relations are lost, indicating that they are not captured in the transformed OCEL. iii) An additional intra-log directly-follows relation is introduced between the register submission date request and enter senddate acknowledgement events for the Case_R object type. Please note that we omit to discuss the intra-log directly-follows relation for BPIC15_3 in this context, as it is present in the other log file.

The absence of the two inter-log relations in the transformed OCEL is indeed expected, as OCEL does not support multi-log event storage. Based on this observation, we can conclude that:

– **Finding 1.** Analyzing a process using multiple OCEL logs (as followed based on DC1.) can result in missing the inter-log relations. On the one hand, an Event Knowledge Graph (EKG) supports analyzing multiple logs simultaneously, meaning it will

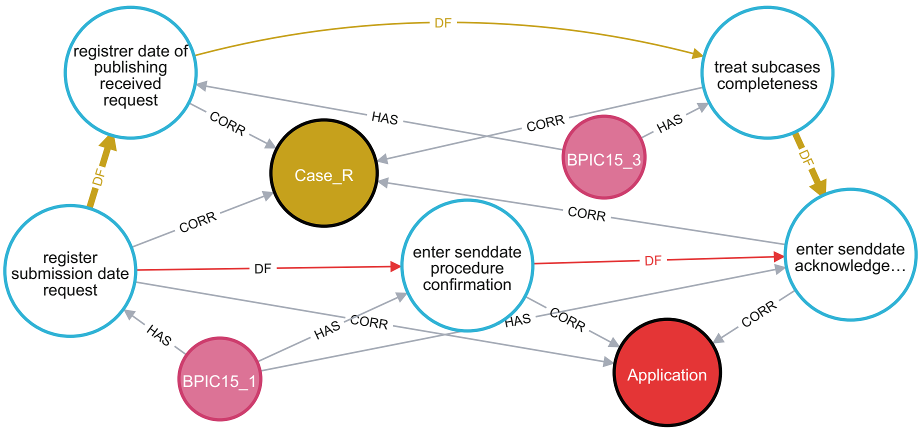
registrer date of publishing received request
treat subcases completeness
Case_R
BPIC15_3
register submission date request
enter senddate procedure confirmation
enter senddate acknowledge…
BPIC15_1
Application
DF
HAS
CORR

Fig. 2. Intra- and inter- log directly-follows relations (shown by thin and thick flows respectively) for a part of BPIC15_1 & for BPIC15_3 in the Event Knowledge Graph

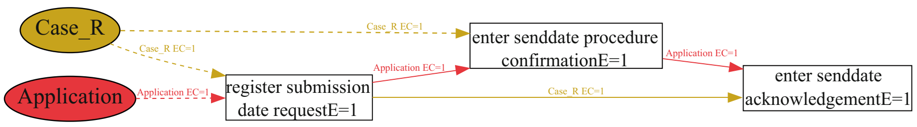
Case_R
Application
register submission date requestE=1
enter senddate procedure confirmationE=1
enter senddate acknowledgementE=1
Case_R EC=1
Application EC=1

Fig. 3. Inter-log directly-follows relations for a part of BPIC15_1 equivalent to Fig. 2

not miss these relations; on the other hand, merging multiple logs into one OCEL and keeping the log information as event attributes can be considered as a technique to handle this shortcoming.

As previously mentioned, some directly-follows relations in the transformed log were not present in the original EKG. For instance, the relation between register submission date request and enter senddate acknowledgement for the Case_R object type was not captured in the EKG. The reason behind this discrepancy lies in the runtime computation of directly-follows relations in Object-Centric Process Mining. In the EKG, two other events were occurring between these two events, resulting in the absence of a direct relation. However, when we project events related to a specific event log, events from other logs are removed, leading to different computations of directly-follows relations among events.

The addition of directly-follows relations can also be observed when filtering event logs based on certain event attributes. An important difference arises when filtering out specific events, such as the enter senddate procedure confirmation event in the EKG (as depicted in Fig. 2). In this case, there would be no directly-follows (DF) relation between the register submission date request and enter senddate acknowledgement events for the Application entity. However, applying the same filter in OCEL would result in a new directly-follows (DF) relation between these two events.

This difference arises because directly-follows relations in OCEL are calculated at runtime based on existing timestamps.

It is important to note that we do not conclude which approach is correct or incorrect. However, this discrepancy is a significant difference that analysts should be aware of to avoid drawing incorrect conclusions.

- **Finding 2.** Inter-log directly-follows relations are not preserved when transforming an EKG to multiple OCELs (as followed based on DC1.). If those relations matter in the analysis, an analyst may follow the alternative design choice stated in DC1.
- **Finding 3.** Analyzing processes with multiple logs using OCEL can include additional directly-follows relations due to the absence of inter-log directly-follows relations. The alternative design choice can be followed to overcome this challenge as stated in DC1.
- **Finding 4.** Filtering OCEL event logs based on specific events can introduce extra directly-follows relations due to the lack of filtered events, similar to the case of filtering traditional logs. These relations are not added when analyzing event knowledge graphs, as all directly-follows relations are pre-calculated.

Differences in BPIC 17: In the EKG, each event is associated with two classes. For instance, event 9 with the activity name O_Created is linked to two classes in the EKG, both of which have the same name as the activity. One class has the type Activity with the same name, while the other class has the type Activity+Lifecycle with the lifecycle value of COMPLETE. However, when transforming to OCEL, the information regarding the lifecycle is not taken into transformation since the OCEL standard does not include lifecycle specifications.

- **Finding 5.** The OCEL standard does not include support for the event lifecycle, but it is supported in EKG. One option to overcome this limitation is to map this information as event's values or related objects as explained in alternative choice for DC2.

5.3 Performance Evaluation

Table 3 shows the performance comparison result of processing event data in EKG and OCEL. The column labeled "Loading Time" in the table represents the time required to prepare the log file for analysis. For OCEL, it indicates the time taken to load the log file into memory. For the EKG, it refers to the time required to load the dump file into Neo4j.

- **Finding 6.** Analyzing OCEL using PM4Py requires the log file to fit within the computer's memory. In contrast, EKG (stored in Neo4j) can handle large data sizes without such memory limitations because a part of graph content is loaded into memory as needed and processed on demand [1], as also demonstrated in [25]. This distinction is crucial when dealing with big data in process analysis as it can enable scaling process mining in practice.

Table 3. Performance comparison (in seconds)

		Loading	Query Execution Time						
		Time	#Log	#Event	#Entity*	#Class	#observed	#corr*	#df
BPIC 14	OCEL	24.86	0.00	0.00	0.00	0.16	0.20	0.00	**124.12**
	EKG	53.59	0.02	0.02	2.37	0.02	0.02	2.79	0.03
BPIC 15	OCEL	9.52	0.00	0.00	0.00	0.05	0.04	0.00	**37.59**
	EKG	25.97	0.02	0.02	0.41	0.01	0.01	1.52	0.03
BPIC 16	OCEL	349.95	0.00	0.00	0.00	2.19	2.44	0.00	**1845.80**
	EKG	166.02	0.02	0.02	7.32	0.01	0.01	**67.71**	0.03
BPIC 17	OCEL	38.96	0.00	0.00	0.00	0.28	0.33	0.00	**181.44**
	EKG	45.03	0.02	0.02	3.24	0.01	0.01	3.53	0.03
BPIC 19	OCEL	57.96	0.00	0.00	0.00	0.22	0.24	0.00	**277.14**
	EKG	62.48	0.02	0.02	2.87	0.02	0.02	7.23	0.03

∗: Non-Reified

– **Finding 7.** Loading logs into EKG is a one-time process, similar to loading data into databases. Once the data is loaded, multiple analyses can be performed without reloading the data. However, with OCEL and PM4Py, the analyst needs to consider the loading time for every new analysis. Keeping large datasets in memory for extended periods may not be efficient, requiring careful consideration for each analysis conducted with OCEL and PM4Py when dealing with big data.

The columns labeled #Log, #Event, and #Entity* represent the query execution times for retrieving the number of logs, events, and non-derived entities in OCEL and EKG, respectively. The queries on OCEL are extremely fast, with execution times rounded to zero. On the other hand, the query execution time for EKG is also reasonable. In the worst case, it takes approximately 7 seconds for BPIC16, which is a substantial EKG. Similar observations can be made for #Class, and #observed. However, there is one exception for BPIC16 in the case of #corr. Retrieving the number of #corr elements takes around one minute due to the size of the EKG, and the additional filtering of #corr relations for non-reified entities significantly increases the query execution time.

Considering the execution query times, a significant difference is observed in calculating the number of directly-follows relations in the log file. These relations play a crucial role as fundamental information for many process mining algorithms. EKG outperforms OCEL in this aspect. This is because all directly-follows relations are materialized in EKG, whereas in OCEL, these relations are computed at runtime during processing. Based on this observation, we can conclude that:

– **Finding 8.** Discovering directly-follows relations on the entire log file is more efficient (performance-wise) in the EKG than OCEL. This is because the relations are materialized in EKG, whereas in OCEL, they are computed at runtime. The precalculation of directly-follows relations in the EKG enhances the efficiency and performance of process mining analyses.

Table 4. Execution time by filtering (in seconds)

		Filters on:	Entity Type		Entity	
		timestamp		timestamp		timestamp
BPIC 14	OCEL	0.73	0.10	0.13	0.10	0.13
	EKG	**12.19** (0.16)	0.46 (0.45)	0.17 (0.07)	0.15 (0.07)	0.12 (0.08)
BPIC 15	OCEL	0.27	0.10	0.08	0.06	0.10
	EKG	**3.97** (0.18)	0.07 (0.09)	0.09 (0.07)	0.12 (0.07)	0.09 (0.09)
BPIC 16	OCEL	8.59	1.85	1.75	1.77	1.78
	EKG	**109.03** (0.16)	1.99 (1.28)	0.45 (0.07)	0.11 (0.07)	0.10 (0.08)
BPIC 17	OCEL	1.49	0.20	0.23	0.21	0.22
	EKG	**15.89** (0.18)	0.53 (0.51)	0.20 (0.08)	0.12 (0.09)	0.09 (0.11)
BPIC 19	OCEL	1.26	0.15	0.16	0.14	0.18
	EKG	**31.26** (0.16)	0.77 (0.79)	0.26 (0.08)	0.13 (0.07)	0.10 (0.10)

The numbers in parentheses are execution time after creating an index on the timestamp.

Applying process mining without appropriate filters can lead to unhelpful and complex process models, often called "spaghetti" models, which is considered a fundamental weakness in most early process mining algorithms [41]. Hence, filtering event logs and focusing on a subset of directly-follows relations is common practice. In our paper, we compare the performance of retrieving different subsets of directly-follows relations from EKG and OCEL on all listed BPICs. We employ common filtering operations such as i) dicing the log based on a timestamp, ii) slicing the log based on an entity type, iii) slicing and dicing the log based on a timestamp and an entity type, iv) slicing the log based on an entity, and v) slicing and dicing the log based on a timestamp and an entity. The performance of slicing and dicing based on timestamp can be improved in neo4j if an index is defined for the timestamp. However, this solution may not be applicable for all attribute types, e.g., if we slice or dice based on the similarity of a textual attribute. Thus, we will test both approaches here. For the timestamp, we follow a pessimistic approach by selecting a timestamp and an entity type that does not exist in the data, which mandates traversing the whole graph when it has no index. Table 4 shows the performance comparison result of retrieving directly-follows relations by applying the above filtering. The numbers in the parenthesis represent the total query time execution after creating an index on the timestamp.

From the third column, it is evident that the performance of retrieving directly-follows relations using PM4Py is significantly better compared to EKG when applying a filter solely based on the event's timestamp without the index. If the index can be defined, EKG has better performance. The main reason behind this difference is that applying such a filter in the EKG without the index necessitates traversing all nodes in the graph, resulting in a time-consuming operation. If the index can be used, EKG will not need to traverse the whole graph. On the other hand, PM4Py executes this operation by processing data in memory.

As observed from the remaining columns, the disparity mentioned above becomes less significant when filtering the log based on other log elements, such as entity type (referred to as object type in OCEL) and entities (referred to as objects). In summary, we can conclude with the following findings:

– **Finding 9.** Analyzing a process using an OCEL log is much more efficient than an EKG without a relevant index when filtering only by dicing the data. In case that index can be defined, EKG has better performance.
– **Finding 10.** There is no significant performance difference when analyzing a process using sliced data for an OCEL or EKG.

There are some limitations and threats to validity that shall be discussed as well. We shall emphasize that some findings can get affected by following alternative design choices as discussed in this section. Currently, we limit the comparison to taken design choices, but we will extend the comparison by considering alternative choices in the future. Also, we shall emphasize that our analysis is based on the current version of the OCEL standard. Our findings and other investigation can influence the extension of this standard in the future, which can relax or change some of the identified findings.

6 Concluding Remarks

This study conducted a comparative analysis of multi-dimensional process analysis using two contemporary conceptual models, namely Object-Centric Event Log (OCEL) and Event Knowledge Graph (EKG). A novel algorithm was introduced to transform EKG into the set of OCEL, which was implemented in Python as part of an open-source library. Five real log files represented in EKG were transformed into OCEL using this algorithm, and the resulting log files were utilized for the comparative analysis.

A total of ten findings emerged from this study, with several noteworthy ones highlighted here. The research shows that transforming EKG containing multiple log files into separate OCELs can cause a loss of inter-log relations between events. Moreover, the study demonstrated differences in analyzing directly-follows relations, attributing them to the materialization of these relations in the EKG while requiring runtime calculations for OCEL. Additionally, it was found that analyzing a process using an OCEL log exhibited higher efficiency compared to an EKG without any index when only dicing the data. Also, it shows how the possibility of applying an index can shift the advantage toward EKG.

As a future direction, it will be interesting to investigate how the OCEL standard can be extended to address some of the reported limitations. It is also interesting to evaluate the difference between these two approaches in calculating directly-follows relations in real use cases where we can have access to stakeholders to evaluate those relations with the help of process experts.

Acknowledgements. Khayatbashi's and Hartig's contributions to this work were funded by Vetenskapsrådet (the Swedish Research Council, project reg. no. 2019-05655).

References

1. The neo4j operations manual v5: Performance: Disks, ram and other tips. https://neo4j.com/docs/operations-manual/current/performance/disks-ram-and-other-tips. Accessed 05 Aug 2023
2. IEEE Task Force on Process Mining. XES Standard Definition (2013). http://www.xes-standard.org
3. Adams, J.N., Park, G., Levich, S., Schuster, D., van der Aalst, W.M.P.: A framework for extracting and encoding features from object-centric event data. In: Troya, J., Medjahed, B., Piattini, M., Yao, L., Fernandez, P., Ruiz-Cortes, A. (eds.) ICSOC 2022. LNCS, vol. 13740, pp. 36–53. Springer, Cham (2022)
4. 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)
5. 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: 2022 4th International Conference on Process Mining (ICPM), pp. 128–135. IEEE (2022)
6. Adams, J.N., van der Aalst, W.M.P.: Precision and fitness in object-centric process mining. In: 2021 3rd International Conference on Process Mining (ICPM), pp. 128–135. IEEE (2021)
7. Adams, J.N., van der Aalst, W.M.P.: Oc π: object-centric process insights. In: Bernardinello, L., Petrucci, L. (eds.) PETRI NETS 2022. LNCS, vol. 13288, pp. 139–150. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-06653-5_8
8. Adams, J.N., van Zelst, S.J., Rose, T., van der Aalst, W.M.P.: Explainable concept drift in process mining. Inf. Syst. **114**, 102177 (2023)
9. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., Vrgoč, D.: Foundations of modern query languages for graph databases. ACM Comput. Surv. (CSUR) **50**(5), 1–40 (2017)
10. Berti, A., Park, G., Rafiei, M., van der Aalst, W.M.P.: An event data extraction approach from SAP ERP for process mining. In: ICPM Workshops, vol. 433, pp. 255–267 (2021)
11. Berti, A., van der Aalst, W.M.P.: OC-PM: analyzing object-centric event logs and process models. Int. J. Softw. Tools Technol. Transfer **25**(1), 1–17 (2023)
12. Berti, A., van Zelst, S., Schuster, D.: PM4Py: a process mining library for Python. Softw. Impacts **17**, 100556 (2023)
13. 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. arXiv preprint arXiv:1905.06169 (2019)
14. Esser, S., Fahland, D.: Multi-dimensional event data in graph databases. J. Data Semant. **10**(1–2), 109–141 (2021)
15. Fahland, D.: Event Graph of BPI Challenge 2019 (2021). https://data.4tu.nl/articles/dataset/Event_Graph_of_BPI_Challenge_2019/14169614/1
16. 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. LNCS, vol. 448, pp. 274–319. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-08848-3_9
17. Fahland , D., Esser, S.: Event graph of BPI challenge 2014 (2021). https://data.4tu.nl/articles/dataset/Event_Graph_of_BPI_Challenge_2014/14169494/1
18. Fahland, D., Esser, S.: Event Graph of BPI Challenge 2015 (2021). https://data.4tu.nl/articles/dataset/Event_Graph_of_BPI_Challenge_2015/14169569/1
19. Fahland, D., Esser, S.: Event Graph of BPI Challenge 2016 (2021). https://data.4tu.nl/articles/dataset/Event_Graph_of_BPI_Challenge_2016/14164220
20. Fahland, D., Esser, S.: Event Graph of BPI Challenge 2017 (2021). https://data.4tu.nl/articles/dataset/Event_Graph_of_BPI_Challenge_2017/14169584/1

21. Ghahfarokhi, A.F., Park, G., Berti, A., van der Aalst, W.M.P.: OCEL: a standard for object-centric event logs. In: Bellatreche, L., et al. (eds.) ADBIS 2021. CCIS, vol. 1450, pp. 169–175. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85082-1_16
22. Gherissi, W., El Haddad, J., Grigori, D.: Object-centric predictive process monitoring. In: Troya, J., et al. (eds.) ICSOC 2022. LNCS, vol. 13821, pp. 27–39. Springer, Cham (2023)
23. Gunther, C.W., Verbeek, H.: Xes-standard definition (2014)
24. Hogan, A., et al.: Knowledge graphs. ACM Comput. Surv. (CSUR) **54**(4), 1–37 (2021)
25. Jalali, A.: Graph-based process mining. In: Leemans, S., Leopold, H. (eds.) ICPM 2020. LNBIP, vol. 406, pp. 273–285. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72693-5_21
26. Jalali, A.: Object type clustering using Markov directly-follow multigraph in object-centric process mining. IEEE Access **10**, 126569–126579 (2022)
27. Jalali, A.: dfgcompare: a library to support process variant analysis through Markov models. BMC Med. Inf. Decis. Making **21**(1), 1–13 (2021)
28. Khayatbashi, S., Hartig, O., Jalali, A.: BPI Challenge 2014 (OCEL) (2023). https://doi.org/10.4121/7d097cec-7304-4b85-9e78-a3ca1cc44c40
29. Khayatbashi, S., Hartig, O., Jalali, A.: BPI Challenge 2015 (OCEL) (2023). https://doi.org/10.4121/110d2fcf-b5e1-494a-a588-896a0a21e60a
30. Khayatbashi, S., Hartig, O., Jalali, A.: BPI Challenge 2016 (OCEL) (2023). https://doi.org/10.4121/95613fb2-29a5-49dc-b196-0948cf96cd7c
31. Khayatbashi, S., Hartig, O., Jalali, A.: BPI Challenge 2017 (OCEL) (2023). https://doi.org/10.4121/6889ca3f-97cf-459a-b630-3b0b0d8664b5
32. Khayatbashi, S., Hartig, O., Jalali, A.: BPI Challenge 2019 (OCEL) (2023). https://doi.org/10.4121/46a7e15b-10c7-4ab2-988d-ee67d8ea515a
33. 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
34. Li, G., de Murillas, E.G.L., de Carvalho, R.M., van der Aalst, W.M.P.: Extracting object-centric event logs to support process mining on databases. In: Mendling, J., Mouratidis, H. (eds.) CAiSE 2018. LNBIP, vol. 317, pp. 182–199. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92901-9_16
35. 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
36. 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 (2023). https://doi.org/10.1007/978-3-031-27815-0_35
37. Rebmann, A., Rehse, J.R., van der Aa, H.: Uncovering object-centric data in classical event logs for the automated transformation from XES to OCEL. In: Di Ciccio, C., Dijkman, R., del Río Ortega, A., Rinderle-Ma, S. (eds.) BPM 2022. LNCS, vol. 13420, pp. 379–396. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-16103-2_25
38. van der Aalst, W.M.P.: Process Mining: Data Science in Action, vol. 2. Springer, Heidelberg (2016)
39. 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
40. van der Aalst, W.M.P., Berti, A.: Discovering object-centric petri nets. Fundamenta informaticae **175**(1–4), 1–40 (2020)

41. van der Aalst, W.M.P., Gunther, C.W.: Finding structure in unstructured processes: The case for process mining. In: Seventh International Conference on Application of Concurrency to System Design (ACSD 2007), pp. 3–12. IEEE (2007)
42. Xiong, J., Xiao, G., Kalayci, T.E., Montali, M., Gu, Z., Calvanese, D.: A virtual knowledge graph based approach for object-centric event logs extraction. In: Montali, M., Senderovich, A., Weidlich, M. (eds.) ICPM 2022. LNBIP, vol. 468, pp. 466–478. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-27815-0_34