Transforming Object-Centric Event Logs to Temporal Event Knowledge Graphs

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. Event logs play a fundamental role in enabling data-driven business process analysis. Traditionally, these logs track events related to a single object, known as the case, limiting the scope of analysis. Recent advancements, such as Object-Centric Event Log (OCEL) and Event Knowledge Graph (EKG), capture better how events relate to multiple objects. However, attributes of objects can change over time, which was not initially considered in OCEL or EKG. While OCEL 2.0 has addressed some of these limitations, there remains a research gap concerning how attribute changes should be accommodated in EKG and how OCEL 2.0 logs can be transformed into EKG. This paper fills this gap by introducing Temporal Event Knowledge Graph (tEKG) and defining an algorithm to convert an OCEL 2.0 log to a tEKG.

Keywords: Event Knowledge Graphs, Object-Centric Event Data, Object-Centric Process Mining

1 Introduction

Business processes involving participants, resources, and systems can be analyzed from different perspectives [9]. These perspectives include different objects based on which a process can be analyzed for further improvement. Traditional analysis focuses on a single object (a.k.a. case), making it challenging to answer questions considering multiple objects and perspectives simultaneously. Object-Centric Process Mining (OCPM) addresses this limitation, aiming to uncover insights by capturing interrelations between objects and events in event logs. Data that includes the relation between events to multiple objects is known as Object-Centric Event Data (OCED) [15], promising the discovery of more insights and addressing the limitations of traditional analysis methods.

Around 2020, two data models were introduced to record OCED: Object-Centric Event Log (OCEL) [6] and Event Knowledge Graph (EKG) [5]. OCEL 1.0 [6] proposed a conceptual model for event logs, enabling the recording of events related to multiple objects and facilitating the development of OCPM algorithms, e.g. [7, 8, 16]. EKG presented an alternative technique to record event logs in a Knowledge Graph [3, 4].

Transforming logs between these formats not only enables the application of techniques developed for each format but also facilitates comparing limitations, which can be used to extend these models for further analysis. For instance, a recent study on transforming EKG to OCEL 1.0 highlights the lack of support in capturing relations between objects in OCEL [10], a concern now addressed by OCEL 2.0 [2].

OCEL 2.0 extends its predecessor with support to record information on object relationships, to qualify relationships, and to capture the change of values for attributes of objects over time [2]. This extension allows capturing the temporal value of objects in practice. As an example, the price of an item in an online webshop can change over time. If these price changes aren't accurately tracked, it becomes difficult to analyze why an item suddenly becomes popular. This is because we lack the correct price data for when customers viewed the item at different times.

While EKG has also undergone improvements, it lacks support for such temporal aspects. Additionally, there exists a gap in transforming OCEL 2.0 to EKG, hindering a direct comparison and tool reuse between these two formats. To fill these gaps, this paper focuses on the following research questions:

RQ1: How can temporal aspects of object attributes be captured in an EKG?

RQ2: How can an OCEL 2.0 log be transformed into a temporal EKG?

To address RQ1 we extend the EKG model into a model of Temporal Event Knowledge Graph (tEKG). To address RQ2 we define an algorithm for transforming an OCEL 2.0 file into our proposed tEKG representation. We have implemented this algorithm and provide the source code of this implementation publicly.[3]

Structure of the paper: Section 2 gives a background using a running example, and Section 3 introduces tEKG informally. Section 4 defines tEKG formally, Section 5 defines the transformation algorithm, and Section 6 concludes the paper.

2 Background

This section introduces the relevant background on EKGs based on a running example.

The example revolves around a fictional education-related process where a student failed to pass a course and must retake it the following year. In the first year, the student reads instructions for an assignment and submitted it accordingly. Subsequently, the teacher decided to increase the points allocated for the assignment from 2 to 3, as it was discovered that the assignment was considerably more challenging than anticipated. Here, we provide a high-level overview of this process to convey the essential concepts necessary for understanding EKG and tEKG. To ensure simplicity, we do not include representations of students, teachers, or other entities typically involved in such a process. A key aspect, however, is that the number of points of the assignment can change over time and must be correctly recorded for the different years.

Fig. 1 illustrates a part of an EKG using nodes and edges to record data of our running example. In EKGs, nodes can be labeled as Log, Entity, Class, or Event. These nodes capture information about log files, objects, event types, and events, respectively. The label Entity is used for nodes representing objects in EKGs. In this paper, we use the terms "object" and "entity" interchangeably when referring to an object in OCEL and EKG, respectively. In the figure, only nodes labeled Entity and Event are displayed. For instance, `c1` and `a1`, depicted as ovals, represent the course and the assignment, respectively, in our example. An event, denoted by `e1` and shown as a diamond, captures

[3] `https://github.com/shahrzadkhayatbashi/BPM2024`

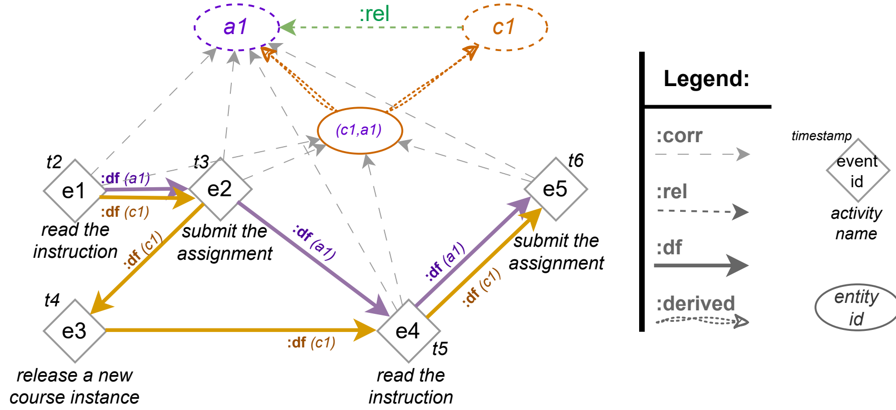


Fig. 1: A part of an Event Knowledge Graph

the event of the student reading the instruction at time t2. Each node can be annotated with key-value pairs called properties. For example, an assignment may have a specific number of points that students can receive upon submitting the assignment.

Edges in EKGs establish relationships among nodes, and these edges can be labeled to indicate the type of relationship between nodes. For example, relationships between entities are represented using edges labeled as rel. In our example, c1 is connected to a1 via such an edge, indicating that the course has an assignment. Edges between nodes labeled Event and nodes labeled Entity are labeled corr, and the label df is used for edges representing directly-follows relationships among nodes representing events.

Such directly-follows relationships between events are fundamental in process mining. In EKGs, two events, say e1 and e2, can be connected by a df edge if i) there is a shared entity to which both events have a corr edge, ii) e1 occurred before e2, and iii) there are no other events that fulfill the first condition and occurred between e1 and e2. Such a df edge is associated with two properties called EntityID and EntityType, representing the identifier and type of the corresponding entity to which the two connected events are related via a corr edge. As an example, in the figure, e1 is linked to e2 by an edge labeled df, with a1 as the value of EntityType.

In this graph, all events have a corr edge to c1 since they are events that occurred within this course. However, we have not depicted these edges in this figure to avoid overwhelming complexity. Instead, we have visualized the resulting df edges. It is apparent that the event flow related to the assignment differs from that of the course, primarily because releasing a new course instance is unrelated to the assignment.

There are scenarios where it becomes necessary to link an object to the relationship between objects, which cannot be achieved directly in EKGs because an edge cannot connect a node to another edge. In EKGs this limitation is addressed by adding helper nodes known as reified entities. An example of a reified entity in Fig. 1 is (c1, a1) which reifies the rel edge from c1 to a1. These two entities are connected to the reified entity by edges with the label derived. All events connected to the aforementioned entities will also be connected to the reified entity. For more detailed information on these concepts, and on EKGs in general, we refer readers to Fahland's work [5].

We emphasize that the The lacks the capability to capture changes in the values of attributes of entities. In our example, the value of the Points property of a1 was modified to 3 in the second year when the student retook the course. Consequently, the recorded information for the first year, where the assignment had 2 Points, would be lost due to the overwrite. This discrepancy can lead to erroneous analysis results. The tEKG model that we propose in this paper addresses this limitation.

3 Temporal Event Knowledge Graphs

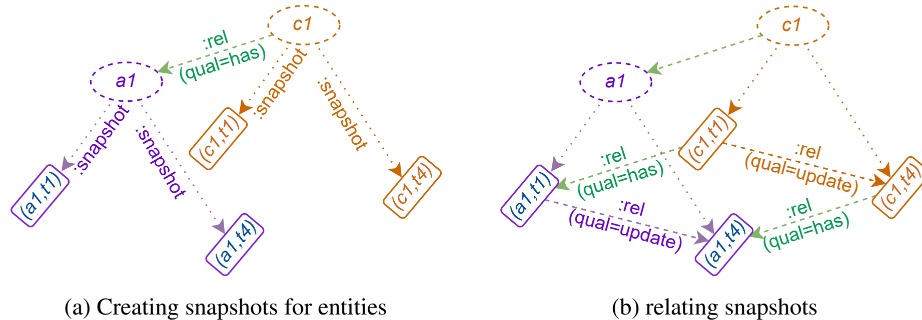
This section introduces our proposal informally and discusses our design choices.

Our initial design choice is to ensure backward compatibility with EKG. This choice aims to facilitate the reuse of existing solutions, such as inferring missing entity identifiers [14] or aggregating event knowledge graphs for task analysis [13]. Therefore, we define tEKG as an extension of the EKG model that supports all EKG features as well as additional features for handling temporal entities.

The values of attributes of entities can change over time, and there are various methods to track these changes in information systems. One approach involves recording transactions for attribute modifications, while another entails capturing snapshots of the state of an object at different points in time. The latter method is commonly employed in data warehousing to store facts in periodic snapshot fact tables [12], which prioritizes query performance for data analysis over transactional performance. Therefore, we have chosen a similar design choice to enhance query performance, which involves generating snapshots of entities when the values of their attributes change.

tEKGs contain multiple nodes per entity; one such node represents the entity in general, independent of the temporal dimension (i.e., exactly as in an EKG), and the other nodes represent snapshots of the entity at specific times. The identifier of each snapshot node is the combination of a timestamp and the identifier of the corresponding entity. Figure 2(a) illustrates such snapshot nodes; in particular, the course c1 and assignment a1 each have two snapshots at times t1 and t4, respectively. The relationship between each entity and its snapshots is established through edges with label snapshot.

Edges with the label rel can be used to capture relationships between snapshots, as illustrated in Figure 2(b). Such an edge has an attribute named qual with a value of



(a) Creating snapshots for entities (b) relating snapshots

Fig. 2: Creating snapshots to capture changes in object's attributes over time

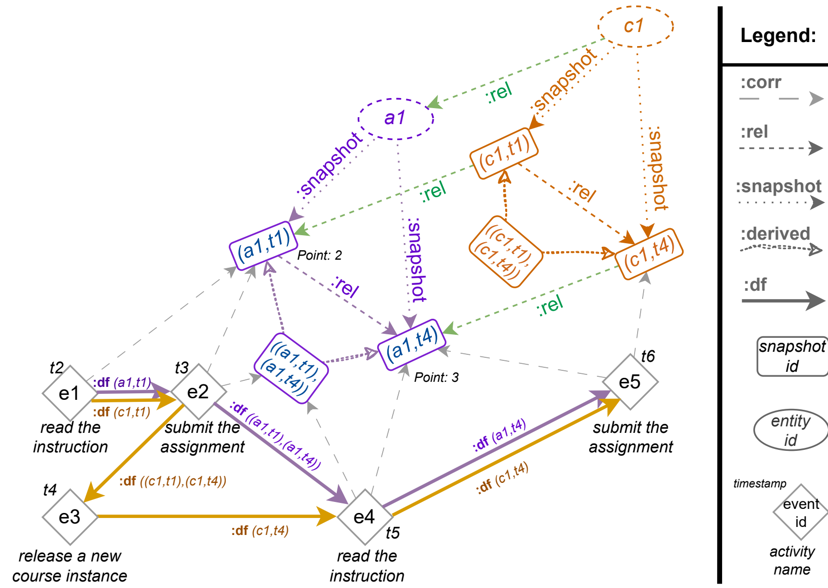


Fig. 3: An example of a Temporal Event Knowledge Graph

update if the connected snapshots are for the same entity. Essentially, such edges document the lifecycle of an entity within a tEKG. For instance, assignment a1 is initially created at time t1 and subsequently updated at time t4, as captured by the snapshot nodes (a1, t1) and (a1, t4), respectively. The edges labeled rel between entities will be copied to their snapshots, with the condition that each snapshot is connected only to snapshots that have existed in its lifetime.

We adapt the design choices made for EKGs to reify entities for snapshots, to connect events to snapshots, and to create directly-follows edges between events corresponding to the same snapshots. This results in additional edges in our graph compared to Figure 1, a portion of which is illustrated in Figure 3. For instance, we have included df edges for snapshots. To avoid over-complicating the illustration, we have omitted drawing previous edges. In particular, we depict corr edges only for the snapshots of a1, which caused the creation of df edges associated with the snapshots of a1.

The tEKG in Figure 3 is more feature rich than a pure EKG, allowing analysts to monitor temporal aspects of entities. For instance, event e2 is connected to the (a1, t1) snapshot and not (a1, t4), which highlights that the student reads the assignment when it had 2 points. Notice also that such connections between events and snapshots of entities at specific points in time are only implicitly present in OCEL 2.0. Making them explicit in a tEKG enables direct access to them for temporal analysis of event logs.

4 Formalization

This section defines our notion of a tEKG. We begin with a recap of definitions of OCEL and EKG that we build on and that are adapted from the related publications [1, 2, 5].

4.1 Preliminaries

Definition 1. $\mathbb{U}_{\Sigma}$ is a **universe** consisting of the following, pairwise disjoint sets [2]:

$\mathbb{U}_{eid}$ is the universe of event identifiers,
$\mathbb{U}_{oid}$ is the universe of object identifiers,
$\mathbb{U}_{etype}$ is the universe of event types,
$\mathbb{U}_{otype}$ is the universe of object types,
$\mathbb{U}_{att}$ is the universe of attribute names,
$\mathbb{U}_{val}$ is the universe of attribute values,
$\mathbb{U}_{time}$ is the universe of timestamps,
$\mathbb{U}_{qual}$ is the universe of qualifiers, and
$\mathbb{U}_{lbl}$ is the universe of labels.

Definition 2. An **Object-Centric Event Log (OCEL)** L is a tuple $(E, O, EA, OA, evtype, evid, time, objtype, objid, eatype, oatype, eaval, oaval, E2O, O2O)$ where [2]:

- E and O are disjoint sets of events and of objects, respectively,
- $EA \subseteq \mathbb{U}_{att}$ and $OA \subseteq \mathbb{U}_{att}$ are sets of attributes for events and objects, respectively,
- $evtype : E \to \mathbb{U}_{etype}$ is a function that assigns event types to events,
- $evid : E \to \mathbb{U}_{eid}$ is a function that assigns event id to events,
- $time : E \to \mathbb{U}_{time}$ is a function that assigns timestamps to events,
- $objtype : O \to \mathbb{U}_{otype}$ is a function that assigns object types to objects,
- $objid : O \to \mathbb{U}_{oid}$ is a function that assigns object id to objects,
- $eatype : EA \to \mathbb{U}_{etype}$ is a function that assigns event types to event attributes,
- $oatype : OA \to \mathbb{U}_{otype}$ is a function that assigns object types to object attributes,
- $eaval : (E \times EA) \nrightarrow \mathbb{U}_{val}$ is a partial function that assigns values to (some) event attributes such that $evtype(e) = eatype(ea)$ for all $(e, ea) \in dom(eaval)$,
- $oaval : (O \times OA \times \mathbb{U}_{time}) \nrightarrow \mathbb{U}_{val}$ assigns values to object attributes such that $objtype(o) = oatype(oa)$ for all $(o, oa, t) \in dom(oaval)$,
- $E2O \subseteq E \times \mathbb{U}_{qual} \times O$ are the qualified event-to-object relations, and
- $O2O \subseteq O \times \mathbb{U}_{qual} \times O$ are the qualified object-to-object relations.

While the $oaval$ function of OCEL assigns values to object attributes at particular points in time, the idea is that such a value remains valid until the next time point at which $oaval$ assigns a new value to the attribute. Yet, for all time points in between, the $oaval$ function is undefined for the corresponding attribute. To denote the *current value* that an object $o \in O$ has for an attribute $oa \in \mathbb{U}_{att}$ at some arbitrary point in time $t \in \mathbb{U}_{time}$, the OCEL specification writes $oaval^{t}_{oa}(o)$, which we formalize as follows.

- If there exists a timestamp $t' \in \mathbb{U}_{time}$ such that i) $t' \leqslant t$, ii) $(o, oa, t') \in dom(oaval)$, and iii) there is no $t'' \in \mathbb{U}_{time}$ such that $t' < t'' \leqslant t$ and $(o, oa, t'') \in dom(oaval)$, then $oaval^{t}_{oa}(o)$ is $oaval(o, oa, t')$.
- If no such t' exists, then $oaval^{t}_{oa}(o)$ is $\perp$, where $\perp$ is a special value not in $\mathbb{U}_{val}$.

Definition 3. A **Labeled Property Graphs (LPG)** G is a tuple $(N, R, \gamma, \lambda, \rho)$ (adopted from [1,5]), where:

- N and R are finite sets of nodes and of edges (relationships), respectively,
- $\gamma : R \to N \times N$ is a function assigning pairs of source and target nodes to edges,
- $\lambda : (N \cup R) \to \mathbb{U}_{lbl}$ is a function assigning a label to every node or every edge,
- $\rho : (N \cup R) \times \mathbb{U}_{att} \nrightarrow \mathbb{U}_{\Sigma} \cup (\mathbb{U}_{\Sigma} \times \mathbb{U}_{\Sigma}) \cup ((\mathbb{U}_{\Sigma} \times \mathbb{U}_{\Sigma}) \times (\mathbb{U}_{\Sigma} \times \mathbb{U}_{\Sigma}))$ is a partial function assigning (potentially composite) values to attributes of nodes and edges.

Given an LPG $G = (N, R, \gamma, \lambda, \rho)$ and a label $\ell \in \mathbb{U}_{lbl}$, we write N^{ℓ} to denote the subset of N consisting of all the nodes with label l; i.e., $N^{\ell} = \{n \in N \mid \lambda(n) = \ell\}$. Similarly, for edges: $R^{\ell} = \{r \in R \mid \lambda(r) = \ell\}$.

We now introduce Event Knowledge Graphs (EKGs) as a special kind of LPGs that use a specific schema $\mathcal{S}$, which we capture as a set of 3-tuples:

$$\mathcal{S} = \big\{(\mathsf{Log}, \mathsf{has}, \mathsf{Event}), (\mathsf{Event}, \mathsf{observed}, \mathsf{Class}), (\mathsf{Class}, \mathsf{dfc}, \mathsf{Class}), (\mathsf{Event}, \mathsf{df}, \mathsf{Event}), \\ (\mathsf{Event}, \mathsf{corr}, \mathsf{Entity}), (\mathsf{Entity}, \mathsf{rel}, \mathsf{Entity}), (\mathsf{Entity}, \mathsf{derived}, \mathsf{Entity})\big\}.$$

Each such 3-tuple represents one of the types of edges in EKGs, where the second element of the 3-tuple provides the label of these edges, and the first and the third element captures the labels of corresponding source and target nodes, respectively. Formally, we say that an LPG $G = (N, R, \gamma, \lambda, \rho)$ *conforms to* $\mathcal{S}$ if, for every edge $r \in R$ with $\gamma(r) = (n, n')$, there exists $(s, l, t) \in S$ such that $n \in N^s$, $n' \in N^t$, and $r \in R^l$.

The ρ function assigns values to an attribute of nodes and edges, so its range is defined to cover different scenarios that are informally explained in Figure 3 such as a singular value (e.g. a_1), a tuple (e.g. (a_1, t_1)), a tuple of tuples (e.g. $((a_1, t_1), (a_1, t_4))$).

Definition 4. An **Event Knowledge Graph (EKG)** is an LPG $(N, R, \gamma, \lambda, \rho)$ that conforms to the schema $\mathcal{S}$ and every node $n \in N$ has the following properties (as per [4,5]):

- If $n \in N^{\mathsf{Event}}$, then $\rho(n, \mathsf{id}) \in \mathbb{U}_{eid}$, $\rho(n, \mathsf{act}) \in \mathbb{U}_{etype}$, and $\rho(n, \mathsf{time}) \in \mathbb{U}_{time}$.
- If $n \in N^{\mathsf{Entity}}$, then $\rho(n, \mathsf{id}) \in \mathbb{U}_{oid} \cup (\mathbb{U}_{oid} \times \mathbb{U}_{oid})$ and $\rho(n, \mathsf{type}) \in \mathbb{U}_{otype}$.

By Definition 4, nodes with the label Event in an EKG have attributes id, act, and time, with the value of an event identifier, an event type and a timestamp, respectively. Similarly, nodes with the label Entity have attributes id and type, with the value of an object identifier and an object type, respectively. The id value can be a single identifier or a tuple thereof. Entities with an id value of the latter type are called *reified entities*.

In contrast to the original definition of EKGs [5], Definition 4 is more relaxed, as it does not enforce the existence of specific properties and edges. This flexibility allows our transformation algorithm to construct and add nodes, edges, and properties incrementally. The same approach is followed in the next definition.

4.2 Temporal Event Knowledge Graphs

To define tEKGs that capture temporal objects, we extend the aforementioned schema $\mathcal{S}$ by adding four more 3-tuples as follows:

$$\mathcal{S}' = \mathcal{S} \cup \{(\mathsf{Event}, \mathsf{corr}, \mathsf{Snapshot}), (\mathsf{Snapshot}, \mathsf{rel}, \mathsf{Snapshot}), \\ (\mathsf{Entity}, \mathsf{snapshot}, \mathsf{Snapshot}), (\mathsf{Snapshot}, \mathsf{derived}, \mathsf{Snapshot})\}.$$

Definition 5. A **temporal Event Knowledge Graph (tEKG)** is an LPG $(N, R, \gamma, \lambda, \rho)$ that conforms to the schema $\mathcal{S}'$ and every node $n \in N$ has the properties as in an EKG (see Definition 4) as well as the following property:

- If $n \in N^{\mathsf{Snapshot}}$, then $\rho(n, \mathsf{id}) \in (\mathbb{U}_{oid} \times \mathbb{U}_{time}) \cup ((\mathbb{U}_{oid} \times \mathbb{U}_{time}) \times (\mathbb{U}_{oid} \times \mathbb{U}_{time}))$ and $\rho(n, \mathsf{type}) \in \mathbb{U}_{otype}$.

By Definition 5, every node with the label Snapshot in a tEKG has attributes id and type, with the value of a snapshot identifier and an object type, respectively. The snapshot identifier is a tuple of an object identifier and a time, or a tuple of such tuples. In the latter case, the corresponding snapshot is called a *reified snapshot*.

5 Transformation

Given the notion of a tEKG, we now specify the transformation algorithm that converts logs from the OCEL 2.0 format into a corresponding tEKG representation. Algorithm 1 defines the main part of the transformation, which is complemented by a procedure for creating the directly-follows edges (Algorithm 2). Additionally, the algorithm uses a procedure called `AddNode` for adding a node to the tEKG, and a procedure called `AddEdge` for adding an edge; due to space constraints, the detailed pseudo code for the latter two procedures is available only in the extended version of this paper [11].

After initializing the tEKG to be populated (line 1 in Algorithm 1), the algorithm initializes a data structure—captured by function $\wp$—for tracking the nodes in the tEKG that have been created based on specific elements of the log (line 2). Next, a node for the log itself is added to the tEKG (lines 3–4). The next step involves adding a node with label Class to the tEKG for each event type in the log (lines 5–7). After that, for each event in the log, a node with label Event is added and connected to the previously-created nodes for both the log and the corresponding event type (lines 8–12).

The algorithm then iterates over all objects in the log (lines 13–23). For each object, it adds a node with label Entity (line 14). Next, it identifies all timestamps at which the value of an attribute of the object has changed (line 16). The algorithm then iterates over these timestamps, adding a node with label Snapshot to the tEKG and linking it to the corresponding entity (lines 17–20). Snapshots are added if the object has a value over time. If an object has no attribute and value in OCEL (i.e., it is not initiated or related to an event), the algorithm creates no snapshot for it. Finally, the algorithm creates edges between such snapshot nodes to represent the updates occurring over time (lines 21–23).

The next step involves adding a rel-labeled edge for every object-to-object relationship between any two objects o_1 and o_2 (line 25). After that, the algorithm iterates over all snapshot edges created for the node corresponding to o_1 in tEKG. It then collects all snapshots of the node corresponding to o_2 in tEKG that occurred before the snapshot of o_1, and puts them into a set named $\mathcal{R}_{st}$ (line 27). This set is used to filter the last valid snapshot for o_2 at the time of o_1, to which we can link the snapshot (line 29). In our running example (see Figure 3), the snapshot $(c1, t1)$ could be linked to $(a1, t1)$, which is a snapshot of the related object with a timestamp that is less than or equal to $(c1, t1)$.

The next step focuses on reified entities and reified snapshots. Here, the algorithm iterates over all rel-labeled edges in the tEKG and adds a node for each of them, as well as an edge from this node to the start and end node of each of these edges (lines 32–36).

Next, the algorithm iterates over all event-to-object relationships in the log (lines 37–46), performing the following operations for each of them: *First*, it adds an edge between the corresponding event and entity (line 38). *Second*, it iterates over all reified entities derived from the corresponding entity and adds an edge from the corresponding event to each of them to tEKG aligned with design choice made in [4] (line 40). *Third*,

Algorithm 1: Converting an OCEL 2.0 log L into a tEKG G.

Input: $L = (E, O, EA, OA, evtype, evid, time, objtype, objid, eatype, oatype, eaval, oaval, E2O, O2O)$

Output: $G = (N, R, \gamma, \lambda, \rho)$

1 Create G as an initially empty tEKG;

2 Let $\wp : E \cup O \cup \mathbb{U}_{etype} \cup (O \times \mathbb{U}_{time}) \cup (O \times O) \rightarrow N$ be an initially empty helper function that maps elements of L to nodes created for them;

// add a node for the log

3 $N \leftarrow N \cup \{log\}$, where log is a new node that is not in N (i.e., $log \notin N$);

4 Extend λ such that $\lambda(log) = \text{log}$;

// add a node for each event type

5 **foreach** $c \in \mathbb{U}_{etype}$ **do**

6 $\quad n \leftarrow$AddNode(c, Class, G, L);

7 $\quad$Extend $\wp$ such that $\wp(c) = n$;

// add a node for each event and connect it to both ...

8 **foreach** $e \in E$ **do**

9 $\quad n \leftarrow$AddNode(e, Event, G, L);

10 $\quad$Extend $\wp$ such that $\wp(e) = n$;

11 $\quad$AddEdge$(G, log, n, \text{has}, \varnothing)$; // ... the log node and the

12 $\quad$AddEdge$(G, n, \wp(evtype(e)), \text{observed}, \varnothing)$; // node of its class

// add a node for each object

13 **foreach** $o \in O$ **do**

14 $\quad n \leftarrow$AddNode(o, Entity, G, L);

15 $\quad$Extend $\wp$ such that $\wp(o) = n$;

16 $\quad \mathcal{O}_{st} \leftarrow \{t \in \mathbb{U}_{time} | (o, oa, t) \in dom(oaval)$ for some $oa \in OA\}$;

$\quad$// add a node for each object snapshot and connect ...

17 $\quad$**foreach** $t \in \mathcal{O}_{st}$ **do**

18 $\quad\quad n' \leftarrow$AddNode$((o, t), \text{Snapshot}, G, L)$;

19 $\quad\quad$Extend $\wp$ such that $\wp((o, t)) = n'$;

20 $\quad\quad$AddEdge$(G, n, n', \text{snapshot}, \varnothing)$; // ... it to the object node

$\quad$// connect the object snapshots in their temporal order

21 $\quad$**foreach** $t_1, t_2 \in \mathcal{O}_{st}$ **do**

22 $\quad\quad$**if** $t_1 < t_2$ and there is no $t_3 \in \mathcal{O}_{st}$ such that $t_1 < t_3 < t_2$ **then**

23 $\quad\quad\quad$AddEdge$(G, \wp((o, t_1)), \wp((o, t_2)), \text{rel}, \text{update})$;

// connect objects and their snapshots using qualifiers

24 **foreach** $(o_1, q, o_2) \in O2O$ **do**

25 $\quad$AddEdge$(G, \wp(o_1), \wp(o_2), \text{rel}, q)$;

26 $\quad$**foreach** $r \in R^{\text{snapshot}}$ with $\gamma(r) = (\wp(o_1), os_1)$ **do**

27 $\quad\quad \mathcal{R}_{st} \leftarrow \{r' \in R^{\text{snapshot}} \mid \rho(os_2, \text{time}) \leqslant \rho(os_1, \text{time})$ with $\gamma(r') = (\wp(o_2), os_2)\}$;

28 $\quad\quad$**foreach** $r' \in \mathcal{R}_{st}$ for which there is no $r'' \in \mathcal{R}_{st}$ with $\gamma(r'') = (\wp(o_2), os_2')$ such that $\rho(os_2, \text{time}) < \rho(os_2', \text{time})$ **do**

$\quad\quad\quad$// connect existing snapshots at a time ...

29 $\quad\quad\quad$AddEdge$(G, os_1, os_2, \text{rel}, q)$;

```
   // add nodes for reified entities and reified snapshots
32 foreach r ∈ R^rel with γ(r) = (℘(o1), ℘(o2)) do
33     n ←AddNode((o1, o2), label, G, L), where label = λ(℘(o1));
34     Extend ℘ such that ℘((o1, o2)) = n;
35     AddEdge(G, n, ℘(o1), derived, ∅);
36     AddEdge(G, n, ℘(o2), derived, ∅) ;
37 foreach (e, q, o) ∈ E2O do
       // connect event nodes to corresponding entity nodes
38     AddEdge(G, ℘(e), ℘(o), corr, q);
39     foreach r ∈ R^derived with γ(r) = (o', o) do
40         AddEdge(G, ℘(e), o', corr, q);
       // connect event nodes to corresponding snapshot nodes
41     R_st ← {r ∈ R^snapshot | ρ(os1, time) ≤ ρ(℘(e), time) with γ(r) = (℘(o), os1)};
42     foreach r ∈ R_st with γ(r) = (℘(o), os1) do
43         if there is no r' ∈ R_st with γ(r') = (℘(o), os2) such that
            ρ(os1, time) < ρ(os2, time) then
44             AddEdge(G, ℘(e), os1, corr, q);
45             foreach r'' ∈ R^derived with γ(r'') = (os3, os1) do
46                 AddEdge(G, ℘(e), os3, corr, q);
47 G ←AddDFs(G) ;                    // add directly-follows edges
48 return G;
```

it retrieves a set of snapshots for the corresponding object that existed at the time of the event (line 41). *Fourth*, it connects the corresponding event to the last valid snapshot (line 44), as well as connecting the event to all derived snapshots of the given snapshot aligning with the same design choice for reified entities made in [4] (line 46).

In the end, Algorithm 2 is called (line 47). This algorithm receives the current tEKG as input, adds relevant directly-follows edges to it, and returns the updated graph as output. The algorithm consists of three *phases*: adding all directly-follows edges, identifying edges that add new information, and removing the ones that do not.

More specifically, in the *first phase*, the algorithm iterates over any two corr edges that are targeting the same entity from two events. If there are no other events occurring in between that have a corr edge to the same entity, the algorithm adds an edge with label df between those two events (line 4). It also sets the value of the ent and type attributes of the added edge to the value of id and type of the entity (line 5).

The *second phase* identifies all df-labeled edges that provide new information (lines 6–10). To this end, the algorithm applies the same rule as defined by Fahland [5], stating that not all df edges created for derived entities provide additional information. Specifically, if there is a derived node o related to o' for which there exist df edges between two events, the df edge created for the derived entity o does not add new information.

In the *last phase*, the algorithm removes df edges that do not add new information with the condition that there shall not be any similar df edges both before and after them that are among the added information df.

Algorithm 2: Extending a given tEKG with directly-follows edges.

```
1  Function AddDFs:
     Input: G = (N, R, γ, λ, ρ)
     Output: G, extended with directly-follows edges
     // add directly-follows edges between event nodes
2    foreach r1, r2 ∈ R^corr with γ(r1) = (e1, o) and γ(r2) = (e2, o) such that e1 ≠ e2 do
3      if there is no r3 ∈ R^corr with γ(r3) = (e3, o) such that e1 ≠ e2 ≠ e3 and
        ρ(e1, time) < ρ(e3, time) < ρ(e2, time) then
4        r ← AddEdge(G, e1, e2, df, ∅);
5        Extend ρ such that ρ(r, type) = ρ(o, type) and ρ(r, ent) = ρ(o, id);
     // identify directly-follows edges providing new
        information
6    I ← ∅;
7    foreach label ∈ {Entity, Snapshot} do
8      foreach r ∈ R^df and o ∈ N^label such that ρ(r, ent) = ρ(o, id) do
9        if there is no r′ ∈ R^df and o′ ∈ N^label and r″ ∈ R^derived such that
          ρ(r′, ent) = ρ(o′, id) and γ(r) = γ(r′) and γ(r″) = (o, o′) then
10         I ← I ∪ {r};
     // remove directly-follows edges not providing new
        information
11   foreach r1, r2 ∈ R^df such that r1 ≠ r2 and γ(r1) = γ(r2), with γ(r1) = (e1, e2) do
12     if r1 ∉ I then
13       if there are no r3, r4 ∈ R^df with γ(r3) = (e′, e1) and γ(r4) = (e2, e″) such
          that ρ(r1, ent) = ρ(r2, ent) = ρ(r3, ent) and r3 ∈ I and r4 ∈ I then
14         R ← R\{r1};
15         dom(γ) ← dom(γ)\{r1};
16         dom(λ) ← dom(λ)\{r1};
17         foreach att ∈ U_att do
18           if (r1, att) ∈ dom(ρ) then
19             dom(ρ) ← dom(ρ)\{(r1, att)};
20   return G;
```

6 Concluding Remarks

This paper introduces and formalizes temporal Event Knowledge Graphs (tEKGs), which are designed to record object-centric event data and to facilitate the tracking of changes in entity attributes over time - like OCEL 2.0 specification but for logs to be stored and processed as graphs. For instance, consider the price of item, which can fluctuate; tEKGs allow for the analysis of events with respect to the accurate price at any given time, before or after any changes. This capability is crucial for conducting effective data-driven analyses in real-world scenarios. Moreover, the paper presents and implements an algorithm to transform Object-Centric Event Logs (OCEL) 2.0 into tEKGs.

As a future direction, we aim to provide a complete formal definition of temporal event knowledge graphs by eliciting requirements for object-centric event data based on different case studies. Investigating the practical applications of tEKGs could provide deeper insights into business processes and decision making.

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. R. Angles, M. Arenas, P. Barceló, A. Hogan, J. Reutter, and D. Vrgoč. Foundations of Modern Query Languages for Graph Databases. *ACM Computing Surveys*, 50(5), 2017.
2. A. Berti, I. Koren, J. N. Adams, G. Park, B. Knopp, N. Graves, M. Rafiei, L. Liß, L. T. G. Unterberg, Y. Zhang, C. Schwanen, M. Pegoraro, and W. van der Aalst. OCEL (Object-Centric Event Log) 2.0 specification. `https://www.ocel-standard.org/2.0/ocel20_specification.pdf`, 2023.
3. S. Esser and D. Fahland. Storing and Querying Multi-Dimensional Process Event Logs using Graph Databases. In *Business Process Management Workshops: BPM 2019 International Workshops, Vienna, Austria, September 1–6, 2019, Revised Selected Papers 17*, 2019.
4. S. Esser and D. Fahland. Multi-dimensional event data in graph databases. *Journal on Data Semantics*, 10(1-2):109–141, 2021.
5. D. Fahland. Process mining over multiple behavioral dimensions with event knowledge graphs. In *Process Mining Handbook*, pages 274–319. Springer, 2022.
6. A. F. Ghahfarokhi, G. Park, A. Berti, and W. M. van der Aalst. OCEL: A Standard for Object-Centric Event Logs. In *New Trends in Database and Information Systems: ADBIS 2021 Short Papers, Doctoral Consortium and Workshops*, 2021.
7. W. Gherissi, J. El Haddad, and D. Grigori. Object-centric predictive process monitoring. In *International Conference on Service-Oriented Computing*, pages 27–39. Springer, 2022.
8. A. Jalali. Object type clustering using markov directly-follow multigraph in object-centric process mining. *IEEE Access*, 10:126569–126579, 2022.
9. A. Jalali and P. Johannesson. Multi-perspective business process monitoring. In *Int. Workshop on Business Process Modeling, Development and Support*, 2013.
10. S. Khayatbashi, O. Hartig, and A. Jalali. Transforming event knowledge graph to object-centric event logs: A comparative study for multi-dimensional process analysis. In *International Conference on Conceptual Modeling*, pages 220–238. Springer, 2023.
11. S. Khayatbashi, O. Hartig, and A. Jalali. Transforming object-centric event logs to temporal event knowledge graphs (extended version), 2024. `https://arxiv.org/abs/2406.07596v1`.
12. R. Kimball and M. Ross. *The data warehouse toolkit: the complete guide to dimensional modeling*. John Wiley & Sons, 2011.
13. E. L. Klijn, F. Mannhardt, and D. Fahland. Aggregating event knowledge graphs for task analysis. In *International Conference on Process Mining*, pages 493–505. Springer, 2022.
14. A. Swevels, R. Dijkman, and D. Fahland. Inferring missing entity identifiers from context using event knowledge graphs. In *Int. Conference on Business Process Management*, 2023.
15. W. van der Aalst. Object-centric process mining: Unraveling the fabric of real processes. *Mathematics*, 11(12):2691, 2023.
16. W. van der Aalst and A. Berti. Discovering object-centric petri nets. *Fundamenta informaticae*, 175(1-4):1–40, 2020.