Implementing Object-Centric Event Data Models in Event Knowledge Graphs

Ava Swevels*[0000−0003−0992−208X], Dirk Fahland*[0000−0002−1993−9363], and Marco Montali†[0000−0002−8021−3430]

* Eindhoven University of Technology, Eindhoven, The Netherlands
{a.j.e.swevels, d.fahland}@tue.nl
† Free University of Bozen-Bolzano, Bolzano, Italy montali@inf.unibz.it

Abstract. Recent advances in object-centric process mining necessitated the standardization of object-centric event data (OCED). An IEEE taskforce has developed a "meta-model" for OCED, but there is no existing reference implementation or automated techniques to transform legacy data into OCED. This task requires domain-specific knowledge about the semantics of the legacy data in order to make explicit how events act on various inter-related data objects and their attributes. We propose a semantic header that defines how extracted legacy data maps to OCED concepts and the domain-specific reference ontology using PG-schema. We automatically translate the header into database queries to construct an event knowledge graph that is compliant with OCED and the domain ontology using a declarative extract-load-transform approach. The approach has been implemented and demonstrated on 7 real-life datasets, making it one of the first attempts to make OCED operational.

Keywords: OCED · Semantic Header · Event Knowledge Graph · PG-Schema · ELT Framework

1 Introduction

The process mining field has recently witnessed a surge in the analysis of real-life processes that (co-)evolve multiple, interrelated objects. Several works have highlighted the intrinsic limitations of conventional case-centric process mining techniques [1,11,12,6] when analyzing and representing such processes. This has fueled the new wave of so-called *object-centric process mining* (OCPM) [2], with some seminal techniques targeting the discovery of process models dealing with multiple objects and their interaction [16,3,7], and some process mining vendors (notably MyInvenio/IBM and Celonis) proposing solutions in this space.

Further research and adoption of OCPM relies on a commonly accepted data model for such processes that can *simultaneously* describe the *structural dimension* (objects, relationships, and their attributes) and the *temporal dimension* (how events create and change structure over time). Although the previously proposed OCEL format [13] is a lightweight representation of object-centric event data, it lacks concepts to express the nature and changes of relationships. To fill this gap, the IEEE task force on process mining has started an initiative to standardize *object-centric event data (OCED)*, which is still under discussion and so far only comes with a proposed meta-model[1], but misses a reference implementation and techniques to extract OCED event logs from legacy sources.

Sect. 2 recalls the OCED proposal and addresses two-subproblems related to the implementation of OCED along the process mining pipeline: *(i)* how to *represent and store* OCED in a way that enables process mining usage, and *(ii)* how to *transform* data from legacy systems into OCED formatted data. Both subproblems are intertwined with the more general problem of linking legacy data with a corresponding semantic description so that the transformed data conforms to the knowledge of the domain [19,20]. Specifically, the OCED proposal is defined on the level of a "metamodel" and has to be instantiated by refining generic event, object, and relation concepts to obtain the actual data model for a given data set.

We address these challenges by refining the model of *event knowledge graphs* (EKGs) [12], which naturally model events, objects, and their relations for process mining, to the OCED proposal. This allows us to represent and store OCED (and any domain-specific refinement of OCED) as a Property Graph (PG) [8] in a standard graph DB system. Our implementation consists of three steps where the first two steps define the "*semantic header*" of the data, which specifies data representation and transformation in line with data semantics, and the third step performs the transformation on the actual instance-level data:

(1) We formalize the OCED proposal as a PG-schema [5], a recent proposal for modeling graph-based data schemas, providing a common interface for process querying. The schema defines a *base ontology* for representing and transforming OCED, which includes a *semantic layer* (defining the OCED concepts) and a *record layer* (defining concepts for generic data records from a legacy system and how they are related to the semantic layer).

(2) We demonstrate how to use PG-schema's inheritance mechanism to specialize this base ontology into a domain-specific *reference ontology* which also includes a *semantic layer* (defining the domain's semantic objects, events, and relations), and a *record layer* (defining in which legacy records the domain-level concepts are stored). Furthermore, we extend these structural definitions with rules to transform data in the record layer into nodes and relationships of the semantic layer, similar to *ontology-based data access* [19].

(3) We provide a declarative *extract-load-transform* (ELT) framework, called *OCED-PG*. We load the legacy data records into the graph DB as a record layer. We then transform the data records into OCED by automatically translating the transformation rules of step (2) into queries over the record layer.

This paper presents two major contributions: the semantic header and OCED-PG, an ELT framework implemented in a Python library (PromG). All in all, this paper provides one of the first attempts to make OCED operational,

[1] cf. the *OCED Symposium* ICPM 2022, `https://icpmconference.org/2022/program/xes-symposium/`

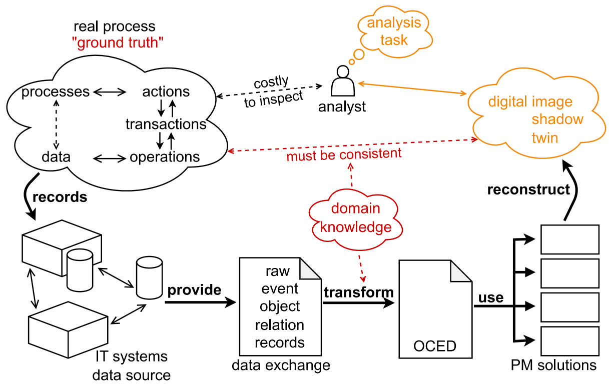


Fig. 1: Data exchange shall support consistent construction of a digital image that is consistent with reality.

and we do so relying on graph database languages and technologies, avoiding ad-hoc implementations allowing analysts to easily manipulate event data for process mining tasks.

The rest of this paper is structured as follows: Sect. 2 reviews the OCED proposal and discusses the challenges of its implementation. Sect. 3 presents a concrete realization of a semantic header for OCED using PG-schema. Sect. 4 gives an overview of OCED-PG. Finally, Sect. 5 evaluates the feasibility of our approach and outlines the conclusions and limitations of this work.

2 Goals and challenges in implementing OCED formats

The technical goal of realizing a standardized OCED format is to provide a unified interface for *data exchange* between source systems and process mining solutions. This goal serves an *analysis goal* as visualized in Fig. 1: an analyst solving a *process analysis task* requires a fundamental understanding of how the actual process operated on the underlying complex data (c.f. Sect. 1). As it is too costly to inspect the reality of the process, the process and data dynamics are recorded in *IT systems*, acting as a *data source* for *(re)constructing a digital image* of the real process. This image *must be consistent* with the process' “ground truth” to let the analyst draw valid conclusions.

Consistency for object-centric processes requires: **(C1)** only showing events, objects, and relationships that were observed in reality (i.e., avoiding convergence and divergence [12]), and **(C2)** representing them in terms of the domain's semantic concepts [19,20,21,9]. We now summarize how the OCED proposal aids consistency, and then we identify concrete challenges for implementing any (standardized) data format for OCED for process mining.

2.1 OCED Meta-Model

In order to ensure **(C1)**, a working group in the process mining community has been developing a more versatile event log standard [18] resulting in the pro-

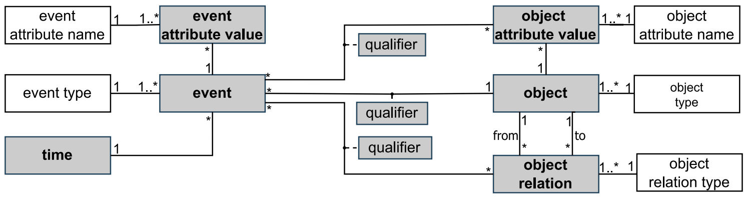


Fig. 2: Draft Object-Centric Event Data (OCED) Meta-Model circulated in the Process Mining community for feedback.

posal[1] shown in Fig. 2. The proposal tries to strike a careful balance between a simple standard and increased expressivity over sequential event logs. It does so by treating events, objects, relations, and their attributes as first-class citizens and by using attributes, relations, and qualified relations between events and objects to capture the structural dimension of the process in a graph-like form [4,14]. The proposal provides a base ontology for events, objects and relations but does not define its semantics as these are domain-specific, and the temporal dimension is only implicitly expressed through timestamps, enabling but not fulfilling **(C2)**.

2.2 Challenge: storage vs semantics

Data stored in source systems generally do not even meet criteria **(C1)** as their data model is optimized for usage rather than analysis: objects are spread across multiple tables, there are no events but only time-stamped records, relations are expressed in various ways, and activities rarely indicate which objects, relations, and attributes were involved [16,10]. Therefore, we argue that generating OCED should address **(C1)** and **(C2)** separately (see Fig. 1). First, extract the data from the source systems to generate a "raw record" ground truth of individual observed events, objects, and relations based on knowledge of the source system storage format. Then, use semantic information to transform the raw data into OCED for PM analysis. We focus on **(C2)** subsequently.

In line with prior literature [19,20,21,9], we stand for a pragmatic, goal-driven approach, arguing that such a semantics-aware transformation depends on the analysis task: the task determines the level of granularity, detail, and required representation of the data. This already holds for constructing classical event logs, where the analyst may combine multiple attributes to define a suitable case identifier or may refine activities. The range of options for OCED transformation is vast, so an OCED implementation must give the analyst the flexibility to determine how the (extracted) raw data maps to the domain knowledge. This gives the analyst the design space to build their analysis in line with the ground truth. Building on ideas from ontology-based data access [19] and virtual knowledge graphs [20], we propose to create a separate *semantic header* which describes how the raw data maps to **(C2a)** OCED's base ontology, and **(C2b)** to the domain data model of the process, requiring a refinement of OCED to represent specific domain concepts. For automation, the semantic header must be rich enough to enable the **(C2c)** automated transformation from the raw data to a “domain-specific” OCED representation. Notably, pairing raw records with a semantic header file holding the domain knowledge realizes also a *light-weight exchange format for OCED* placing the work intensive transformation to OCED to data import.

In the following, we tackle **(C2a)** to **(C2c)** through an entirely graph-based approach. We extend the model of event knowledge graphs [12] to satisfy **(C2a)** and **(C2b)**. Using a graph-based representation allows us to specify simple declarative transformation rules that we can automatically translate into graph queries for automated translation to satisfy **(C2c)**. Although not as general as full-fledged pipelines for mapping legacy relational data to case-centric [10,9] and object-centric [21] event logs, this results in a pragmatic approach that is fully grounded on graph-based representations, query languages, and underlying graph DB technologies.

3 A Semantic Header for Object-Centric Event Data

In this section, we define the concrete realization of a semantic header for object-centric event data, resorting to the PG-schema approach for property graphs [5]. The semantic header consists of a *(i) base ontology* including a *semantic layer* encoding OCED and a *record layer* encoding raw data records and linking them to the semantic layer, and *(ii)* a *reference ontology* extending the *base ontology* layers with domain-specific records, entities, events, and relations.

3.1 A Gentle Introduction to PG-Schema

A property graph database uses, as main storage mechanism, a property graph (PG), which is a directed multi-graph where nodes represent objects and edges represent relationships. The two main features of PGs is that nodes and relationships come with labels and properties. Labels are used to type elements. While nodes may be typed with multiple labels, each relationship has exactly one. Furthermore, nodes and relationships carry properties, represented as attribute-value pairs. PGs are typically schema-less, and only recent developments, especially PG-schema [5], have highlighted the need to define schemas for PGs.

PG-Schema defines property graph schemas via PG-Types and PG-Keys. PG-Types deal with typing rules for nodes/edges, defining allowed types through admissible combinations of labels and properties in nodes and edges, and also constraining the types of edges that can be connected between nodes of certain types. PG-Keys provide a range of integrity constraints over types, including keys and participation constraints . Our work is currently limited to only include PG-Types, consisting of node types, edge types and graph types. We describe these constructs showing how they can be used to define a schema for EKGs, following the structure presented in [12]; this will also prove useful later, as it will provide the basis for the PG-schema encoding of OCED. A PG-schema for EKGs is given through the declaration of the graph type EKGType, illustrated in Fig. 3. It consists of 3 node types characterized by labels Event, Entity and

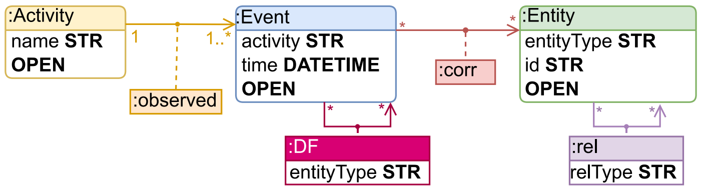


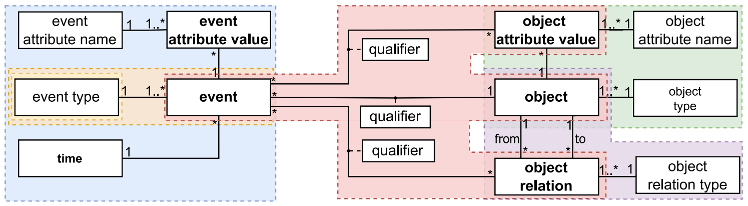
Fig. 3: PG-Schema for graph type EKGType

Activity, for events, activities, and entities respectively, and 4 relationships with labels corr ("event correlated to entity"), df ("event directly followed by event"), rel ("entity related to entity") and observed ("activity observed event").

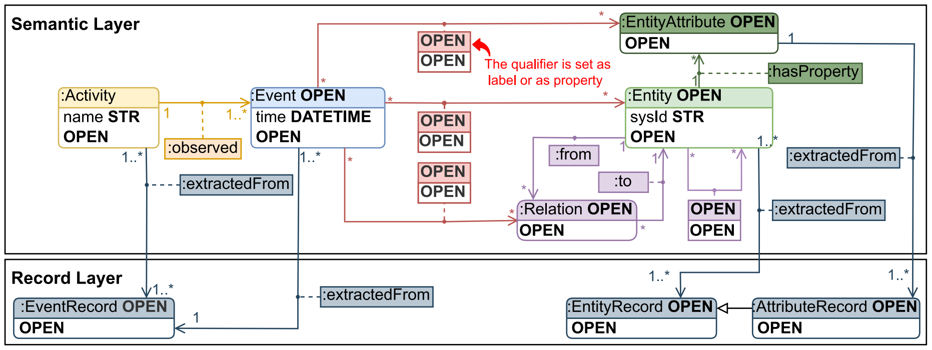
Node types are defined by their label and properties together with their data type. No other properties or labels can be attached to a node type, unless the keyword OPEN is declared among the properties or labels respectively. Furthermore, a property can be declared as OPTIONAL. Edge types are defined by their source and target node, label and properties. The properties of an edge type and the OPEN keyword are used similarly as with node types.

3.2 Base Ontology

Semantic layer: Encoding OCED as a PG-Schema We have formalized and specified the OCED proposal using PG-schema in the *semantic layer* of the *base ontology* (addressing **(C2a)**). The resulting graph type baseOntologyType is presented next to the OCED proposal in Fig. 4b (Semantic Layer) using the



(a) OCED Proposal



(b) Visual representation of PG-Schema of baseOntologyType

Fig. 4: Side-to-side comparison of OCED as meta-model and as PG-Schema

same color scheme for similar concepts. baseOntologyType stems from EKGType in Fig. 3, and substantially modifies it in the following five aspects. (1) Property entityType is removed from the Entity nodes. Now, we allow for multiple labels, as indicated by the keyword OPEN. Hence, instead of indicating the entity type as a property, it is specified by a dedicated label. In the same way, since there can be several types of events, the Event node now also allows for multiple labels. (2) The attributes of an entity that evolve over time are modeled as separate EntityAttribute nodes. (3) The relationship :rel can be modeled as a relationship, or reified as a :Relation node. In the latter case, edges :from and :to indicate the related entities and the direction of the relationship. The way in which entities are related depends on the domain context and should be specified by a label. Therefore, we permit additional labels through OPEN. (4) Since events act not only on entities, but also on attributes and relationships, a relationship is created from Event nodes to Entity nodes, EntityAttribute nodes and Relation nodes. Again, how events act on entities, attributes, and relationships depend on domain context. This qualifier can be set as a label or as a property (cf. OPEN). (5) The DF relation has been removed as this is not part of OCED.

Record Layer The *base ontology* also consists of a *record layer*, as shown in Fig. 4b (Record Layer). This layer is used to lift raw data records into the graph DB, and link them to the elements of the semantic layer - once the actual instances of the semantic layer are defined, it can hence be removed, or kept for provenance reasons. The record layer has 3 nodes with labels EventRecord, EntityRecord and AttributeRecord, and a relationship with label extractedFrom, indicating where a semantic layer node actually comes from. The nodes are OPEN, implying they can have additional attributes and labels.

3.3 Example of a Reference Ontology

From now on, we use an example related to a library loan process to illustrate how OCED-PG works. In this section, we use it to show how the base ontology can be refined into a reference ontology using PG-schema's inheritance mechanism as shown in Fig. 5 (addressing **(C2b)**). Specifically, we extend baseOntologyType (Fig. 4b) into libraryType to reflect the semantics of the process. A Library holds Books in its catalog. Events with observed Activities *borrow*, *extend*, *return* and *update membership*, are executed by a Member and act on Books or on the MembershipAttribute (which a Member has as property). The borrowed relation between Books and Members are created or removed by an Event. The record layer reflects the source of the semantic layer nodes. Hence, we consistently refine it as well, indicating that Events and Activities are extracted from EventRecords, Members from MemberRecords, and so on.

4 Framework: OCED-PG

We now explain how actual (event) data can be stored in a graph DB using the semantic header, following Sect. 3. To this end, we define OCED-PG which

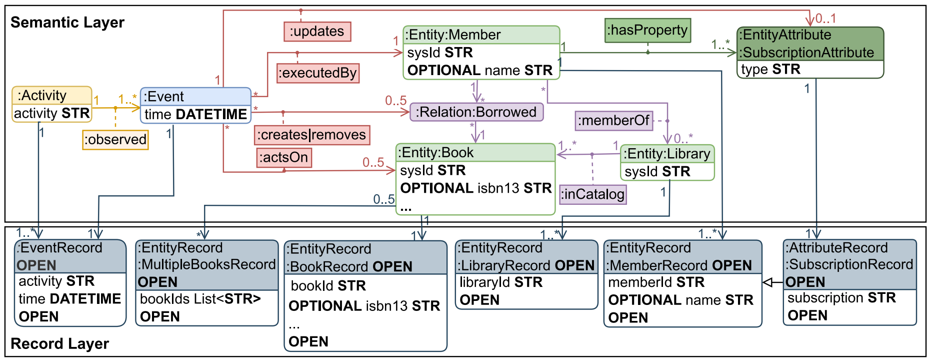


Fig. 5: PG-Schema for libraryType capturing the domain knowledge of a library loan process

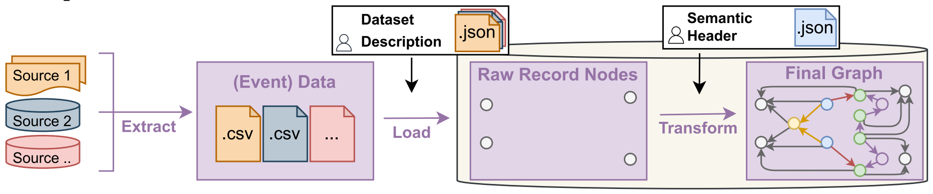


Fig. 6: Overview of OCED-PG, a declarative ELT framework

is a Extract, Load, and Transform (ELT) framework outlined in Figure 6. The first step is to *extract* the (event) data from different sources into CSV files (see Sect. 4.1). Then we *load* each record of the CSV files as a Record node in the graph database. We ensure that the properties in these nodes are properly names and typed using a data set description file (see Sect. 4.2). Lastly, we *transform* the Record nodes into a new layer in the graph DB that captures the semantic meaning of the records with automatically generated queries (see Sect. 4.3).

4.1 Extracting Data from Legacy Sources

The first step is to extract the original, legacy data, without attached semantics into CSV files. Such data can be taken from multiple sources, for example, SAP systems, production data, or APIs. Data sets may include event and/or entity records. Timestamped event records document changes to the process and associated data. Entity records describe entities with their properties.

For our technical example, we generated, through simulation, three data sets containing *(i)* timestamped records of members executing activities, *(ii)* book records, and *(iii)* member records respectively. An overview is given in Tab. 1.

4.2 Load Data

After extraction, we *load* the data set records into corresponding nodes of a graph DB. This is automated using *dataset description files* containing a description of each record attribute. Each attribute is described by its attribute name, the

Events (18616 records)			Library books (5000 records)			Members (1000 records)		
attribute	column	dtype	attribute	column	dtype	attribute	column	dtype
timestamp	time	TIME	libraryId	library	STR	memberId	id	STR
activity	activity	STR	bookId	id	STR	name	name	STR
memberId	member	STR	isbn13	isbn13	STR	subscription	type	STR
bookIds	book ids	List[STR]	...	...	...			
membership	type	STR						

Table 1: Overview of the simulated datasets for the library loan example

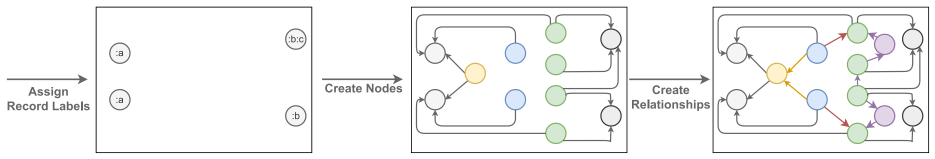


Fig. 7: Overview of the different transformation rules in the Transform step

corresponding column name, the datatype (see Tab. 1) and, optionally, a string format to parse timestamp attributes. Each data set is *loaded* as follows: *(i)* using a description file, the corresponding data set is loaded into memory; *(ii)* for each attribute, the corresponding column is renamed to comply with the syntactical rules of the graph DB; *(iii)* each record from the data set is loaded as a Record node into the graph DB; *(iv)* timestamp attributes are converted to date(time).

4.3 Transform

The last step of OCED-PG is to *transform* the raw Record nodes into an EKG using the semantic header (addressing **(C2c)**). The semantic header is concretely stored as a JSON document that contains the definition of the reference ontology (using the (OCED) constructs from the base ontology as supertypes). We extend these structural definitions with three types of transformation rules: *(i)* rules to assign labels to raw Record nodes; *(ii)* rules to create, from record layer nodes, so-called *semantic nodes*, i.e., instances of the types defined in the semantic and domain layers, *(iii)* rules to create relationships between semantic nodes. Rules of these three distinct types are applied in cascade, following the order shown in Fig. 7 (record nodes are in gray). We detail next how these rules are formed.

Assign Record Labels rules. Since Record nodes may contain different types of data, they are distinguished through specific labels. The semantic header includes, for each type of record, a label, the required and optional attributes, and optionally a condition using the pattern shown in Listing 1.1(a). The transformation rule is automatically translated into a query that identifies all Record nodes with the correct attributes and then assigns the appropriate label.

Semantic Node Creation rules. Semantic nodes are created based on their presence in at least one record layer node. The semantic header includes, for each semantic node, its labels, the required attributes, the optional attributes, and from which record node it is extracted, following the pattern in Listing. 1.1(b). The transformation rule is translated into a query that identifies the correct record nodes and then creates or merges a node with the correct labels and attributes. Merging a node means that it is created if it does not exist, otherwise

(a) Semantic Header Input for Record Nodes

```
(record:$record_labels
    WHERE $condition
    {$required_attributes,
    OPTIONAL $optional_attributes})
```

Generated Query

```
MATCH (record:Record)
WHERE $condition
AND $required_attributes_not_null
SET record:$record_labels
```

(b) Semantic Header Input for Nodes

```
Record: (record:$record_labels)

Semantic Node: ($node_name:$node_labels
            $required_attributes,
            OPTIONAL $optional_attributes})
```

Generated Query

```
MATCH (record:$record_labels)
CREATE/MERGE ($node_name:$node_labels
            {$set_required_attributes})
SET $set_optional_attributes
MERGE (record) <- [:extractedFrom]
     - ($node_name)
```

Listing 1.1: Transformation rule patterns and the generated queries to assign labels to record nodes (a) and to create semantic nodes (b).

it is retrieved. (E.g., Entity nodes with the same identifier are merged.) Laslty, a extractedFrom relation from the record to the semantic node is merged.

Semantic Relationship Creation rules. Relationships can be created in two different ways. First, two different semantic nodes are related if they have been extracted from the same Record node. Fig. 8 visually shows that the actsOn relationship can be created between Event and Book nodes that are extracted from the same :EventRecord:BookRecord node. Second, nodes can be related based on a subgraph of the semantic layer. For example, domain knowledge is used to determine the coreference constraints indicating that a Member can only borrow a Book from the catalog of a Library if they are a member of that Library (Fig. 8).

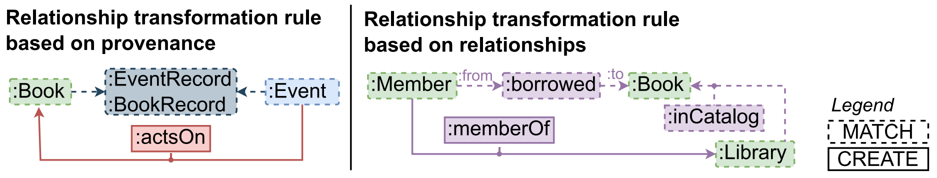


Fig. 8: Example of actsOn and memberOf transformation rules.

5 Conclusion and demonstration

This paper explored the development of a reference implementation for OCED. We proposed a three-layer approach to create a semantic-aware representation and storage system for OCED. We developed OCED-PG, a declarative ELT framework, that maps the raw data to a corresponding EKG, using the semantic header as a basis. The method is robust and extendable due to the flexibility of PG schema and property graphs allowing for alterations or expansions to OCED.

OCED-PG is implemented in a Python library called PromG. It is designed to automatically generate Cypher queries against a Neo4j instance to transform raw data into the semantic domain. We have identified patterns in the transformation

Data set	Source	#node types					#edge	Memory	Time
	Size (GB)	:Event	:Activity	:Entity	:Attribute	:Relation	types	(GB)	(mins)
Library	0.002	1	1	3	1	1	7	1	0.5
BPIC'14	0.08	2	1	7	0	0	11	1	7.7
BPIC'15	0.11	1	1	3	0	0	7	1	4.4
BPIC'16	1.06	4	1	13	0	0	17	5	162.4
BPIC'17	0.29	1	1	5	0	0	4	1	19.7
BPIC'19	0.52	1	1	7	0	0	8	5	31.2
SAP	0.01	1	1	4	0	2	5	1	2.0
Manufacturing	0.03	1	1	4	0	0	1	1	3.7

Table 2: Type information of EKGs and execution time for 8 datasets

rules, providing a simple interface (i.e. the semantic header) for analysts to manipulate event data. We tested OCED-PG on eight different data sets[2] by constructing an EKG for each dataset based on a semantic header, demonstrating the feasibility of our approach. Tab. 2 gives an overview of the data sets, the domain knowledge applied, the memory allocated, and the execution time.

Our implementation of OCED is not intended to be a data transport format but is designed for analysis. We have established key concepts for specifying semantic headers of process event data, allowing analysts to customize the semantic model for the specific process and analysis without compromising the data storage format. EKGs have already been demonstrated to be beneficial for further enrichment and analysis of the data. Previous research has applied EKGs for multi-entity process discovery and conformance checking [12], task identification [15], concept drift detection, and inference of missing entity identifiers [17].

Our work serves as a proof-of-concept, but it has certain limitations. First, the semantic header only allows queries to enrich nodes and to create nodes and relationships from existing nodes and subgraphs; any other semantic inference will require extensions to the semantic header. Second, PG-schema is used as a conceptual idea, but neither the semantic header nor the database actually implements it. Third, the implementation is not optimized for efficiency or performance; for instance, since we perform a query for each entry in the semantic header, we have to loop over all the record nodes for each entry in the worst case. Lastly, our method does not take into account streaming data or a standardized import/export for data in an OCED data transport format.

Acknowledgement. The research underlying this paper was partially supported by AutoTwin EU GA n. 101092021.

References

1. van der Aalst, W.M.P.: Object-centric process mining: Dealing with divergence and convergence in event data. In: SEFM 2019. LNCS, vol. 11724, pp. 3–25. Springer (2019)
2. van der Aalst, W.M.P.: Twin transitions powered by event data - using object-centric process mining to make processes digital and sustainable. In: ATAED 2023. CEUR Workshop Proceedings, vol. 3424. CEUR-WS.org (2023)

[2] See `https://zenodo.org/record/8296559`

3. van der Aalst, W.M.P., Berti, A.: Discovering object-centric petri nets. Fundam. Informaticae **175**(1-4), 1–40 (2020)
4. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Computing Surveys **40**(1), 1:1–1:39 (2008)
5. Angles, R., et al.: Pg-schema: Schemas for property graphs. ACM on Management of Data (PACMMOD, 2023) **1**(2), 198:1–198:25 (2023)
6. Artale, A., Kovtunova, A., Montali, M., van der Aalst, W.M.P.: Modeling and reasoning over declarative data-aware processes with object-centric behavioral constraints. In: BPM 2019. LNCS, vol. 11675, pp. 139–156. Springer (2019)
7. Barenholz, D., Montali, M., Polyvyanyy, A., Reijers, H.A., Rivkin, A., van der Werf, J.M.E.M.: There and back again - on the reconstructability and rediscoverability of typed jackson nets. In: PETRI NETS 2023. LNCS, vol. 13929, pp. 37–58. Springer (2023)
8. Bonifati, A., Fletcher, G.H.L., Voigt, H., Yakovets, N.: Querying Graphs. Synthesis Lectures on Data Management, Morgan & Claypool Publishers (2018)
9. Calvanese, D., Jans, M., Kalayci, T.E., Montali, M.: Extracting event data from document-driven enterprise systems. In: CAiSE 2023. LNCS, vol. 13901, pp. 193–209. Springer (2023)
10. Calvanese, D., Kalayci, T.E., Montali, M., Santoso, A.: OBDA for log extraction in process mining. In: Reasoning Web (RW, 2017). LNCS, vol. 10370, pp. 292–345. Springer (2017)
11. Dumas, M., Fournier, F., Limonad, L., et al.: AI-augmented business process management systems: A research manifesto. ACM Trans. Manag. Inf. Syst. **14**(1), 11:1–11:19 (2023)
12. Fahland, D.: Process mining over multiple behavioral dimensions with event knowledge graphs. In: Process Mining Handbook, vol. 448, pp. 274–319. Springer (2022)
13. Ghahfarokhi, A.F., Park, G., Berti, A., van der Aalst, W.M.P.: OCEL: A standard for object-centric event logs. In: ADBIS 2021 Short Papers. Communications in Computer and Information Science, vol. 1450, pp. 169–175. Springer (2021)
14. Hogan, A., et al.: Knowledge graphs. ACM Comput. Surv. **54**(4), 71:1–71:37 (2022)
15. Klijn, E.L., Mannhardt, F., Fahland, D.: Classifying and detecting task executions and routines in processes using event graphs. In: BPM Forum. pp. 212–229. Springer International Publishing, Cham (2021)
16. Lu, X., Nagelkerke, M., van de Wiel, D., Fahland, D.: Discovering interacting artifacts from ERP systems. IEEE Trans. Serv. Comput. **8**(6), 861–873 (2015)
17. Swevels, A., Dijkman, R., Fahland, D.: Inferring missing entity identifiers from context using event knowledge graphs. In: BPM 2023. LNCS (2023)
18. Wynn, M.T., Lebherz, J., van der Aalst, W.M.P., Accorsi, R., Ciccio, C.D., Jayarathna, L., Verbeek, H.M.W.: Rethinking the input for process mining: Insights from the XES survey and workshop. In: ICPM 2023 Workshops. LNBIP, vol. 433, pp. 3–16. Springer
19. Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., Zakharyaschev, M.: Ontology-based data access: A survey. In: Artificial Intelligence (IJCAI, 2018). pp. 5511–5519. ijcai.org (2018)
20. Xiao, G., Ding, L., Cogrel, B., Calvanese, D.: Virtual knowledge graphs: An overview of systems and use cases. Data Intell. **1**(3), 201–223 (2019)
21. 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. LNBIP, vol. 468, pp. 466–478. Springer (2022)