OCEL (Object-Centric Event Log) 2.0 Specification

Alessandro Berti, István Koren, Jan Niklas Adams,
Gyunam Park, Benedikt Knopp, Nina Graves, Majid Rafiei, Lukas Liß,
Leah Tacke Genannt Unterberg, Yisong Zhang, Christopher Schwanen, Marco Pegoraro,
Wil M.P. van der Aalst

Chair of Process and Data Science, RWTH Aachen University

ocel@pads.rwth-aachen.de

Version: 2.0
Date: October 16, 2023
Standard Document URL:
https://www.ocel-standard.org/2.0/ocel20_specification.pdf
Validation Schemes:

- XML: https://www.ocel-standard.org/2.0/ocel20-schema-xml.xsd
- JSON: https://www.ocel-standard.org/2.0/ocel20-schema-json.json
- Relational: https://www.ocel-standard.org/2.0/ocel20-schema-relational.pdf

Abstract

Object-Centric Event Logs (OCELs) form the basis for Object-Centric Process Mining (OCPM). OCEL 1.0 was first released in 2020 and triggered the development of a range of OCPM techniques. OCEL 2.0 forms the new, more expressive standard, allowing for more extensive process analyses while remaining in an easily exchangeable format. In contrast to the first OCEL standard, it can depict changes in objects, provide information on object relationships, and qualify these relationships to other objects or specific events. Compared to XES, it is more expressive, less complicated, and better readable. OCEL 2.0 offers three exchange formats: a relational database (SQLite), XML, and JSON format. This OCEL 2.0 specification document provides an introduction to the standard, its metamodel, and its exchange formats, aimed at practitioners and researchers alike.

Contents

1 Scope and Structure of this Document

This document introduces the OCEL 2.0 standard to record and exchange object-centric event logs. The purpose of the standard is to guide the implementation of conformant process mining tools, and to provide the basis for the development of training material and other resources for users.

OCEL 2.0 and its metamodel are designed from the ground up to facilitate the exchange of event logs coming from a wide variety of information systems. Unlike traditional exchange formats, events may refer to any number of objects of different types. It is intended to be interoperable with data extracted from a wide variety of databases, systems, or applications. Likewise, the format aims to be equally compatible with existing and emerging Object-Centric Process Mining (OCPM) techniques. It is anticipated that OCEL 2.0 will become the default exchange format for OCPM tools, whether these are research prototypes or commercial tools.

This document is structured as follows. Section 2 explains object-centric process mining and discusses the limitations of current standards for recording object-centric event logs. Section 3 introduces the OCEL 2.0 standard and discusses its advantages over past object-centric standardization attempts. Section 4 contains the formal definitions of the standard. Section 5 illustrates OCEL 2.0 using a running example. Sections 6, 7 and 8 describe the practical implementation of the standard, using relational, XML, and JSON formats, respectively. Finally, Section 9 concludes this document.

2 Introduction to Object-Centric Process Mining

The first process mining algorithms were developed in the late 1990-ties [1, 4]. Initially, adoption was limited, with just a handful of researchers working on the topic. However, over time, the field matured. Currently, there are over 40 vendors offering process mining software (cf. `www.processmining.org`) and advisory firms such as Gartner consider these to form a new and substantial category of tools [11]. Many of the world's largest companies already use process mining to improve their operational processes (across all economic sectors), and adoption is expected to increase in the coming years. The increasing maturity of the process-mining discipline is also reflected by the success of the International Conference on Process Mining (ICPM) and the large number of process-mining papers in other conferences (e.g., BPM and CAiSE) and journals.

However, traditional process mining considers processes involving single cases, their events, and event attributes. The approach falls short when dealing with complex, multi-dimensional processes, where events possibly relate to a variety of entities or *objects* that interact and evolve over time [2]. Traditional event data are based on the assumption that each event refers to precisely one case. The same applies to mainstream process modeling notations like Directly-Follows-Graphs (DFGs), BPMN models, UML activity diagrams, workflow nets, and process trees. However, most real-life events involve multiple objects. Traditional process mining approaches require the *flattening* of event data in order to satisfy this assumption. This may lead to misleading analysis results. Process mining results also tend to become more complex because different objects get intertwined while trying to straitjacket the processes. Changing the viewpoint (e.g., looking at the process from a different angle) also implies changing the case notion and going back to the source systems to extract other event data. This leads to redundancies in event data and unnecessary repetitions. Moreover, many compliance and performance problems arise at the intersection points of processes, systems, and organizations.

Object-Centric Process Mining (OCPM) represents a paradigm shift, intended to address and overcome the inherent limitations of traditional case-centric process mining methods [2]. OCPM starts from the actual events and objects that leave traces in ERP (Enterprise Resource Planning), CRM (Customer Relationship Management), MES (Manufacturing Execution System), and other IT systems. In the databases of such systems, one-to-one relationships are the exception. Most relationships are one-to-many or many-to-many. As a result, data need to be transformed to be able to assign events to a single case, leading to all the problems mentioned before. Therefore, there is consensus among experienced process miners that *Object-Centric Event Data* (OCED) provide a much better abstraction of reality than the classical case-based event logs. OCEL 1.0, released in 2020 [7], was the first standard for storing OCED and triggered the development of OCPM techniques (e.g., discovering object-centric process models). OCEL 2.0 extends OCEL 1.0, leveraging experiences gathered while developing and applying these OCPM techniques.

Before discussing in what way OCEL 2.0 extends OCEL 1.0, we first need to introduce some terminology and basic concepts.

- *Events*: Object-centric process mining works on discrete events. They represent the various actions or activities that occur within a system or process, such as approving an order, shipping an item, or making a payment. Every event is unique and corresponds to a specific action or observation at a specific point in time. Events are atomic (i.e., do not take time), have a timestamp, and may have additional attributes. Events are typed.

- *Event Types*: Events are categorized into different types based on their nature or function. For example, a procurement process might have event types such as Order Created, Order Approved, or Invoice Sent. Each type of event represents a specific kind of action that can take place in the process. Each event is of exactly one type. Sometimes, we use the term *activity* to refer to an event type.

- *Objects*: In object-centric process mining, objects represent the entities that are involved in events. These might be physical items like products in a supply chain, machines, workers, or abstract/information entities like orders, invoices, or contracts in a procurement process. Objects have attributes with values, e.g., prices. These values may change over time.

- *Object Types*: Each object is of one type. The object is an instantiation of its type. Object types might include categories like Product, Order, Invoice, or Supplier.

Events and objects may be related. In particular, OCPM techniques exploit the following two relationships.

- *Event-to-Object (E2O) Relationships*: Events are associated with objects. This relationship describes that an object affects an event or that an event affects an object. In contrast to traditional event logs, events can be related to multiple objects. Furthermore, these relationships can be qualified differently, describing the role an object plays in the occurrence of this specific event. Consider, for example, a meeting event involving multiple participant objects. Using a qualifier, it is possible to distinguish between regular participants and the organizer of the meeting.

- *Object-to-Object (O2O) Relationships*: Objects can also be related to other objects outside the context of an event. For example, an employee may be part of an organizational unit. In addition to the mere existence of a relation, this relationship can also be qualified (e.g., part-of, reports-to, or belongs-to).

Recent standardization attempts have addressed some but not all of these requirements. The first OCEL format (OCEL 1.0) provided an event log standard that could capture events related to multiple objects with attributes but did not include Object-to-Object (O2O) relationship, qualifiers for either O2O and E2O relationships, or changing object attribute values [7, 8]. OCEL 2.0 addresses these limitations by providing a new metamodel and three storage formats, including a relational implementation of the standard. We will address OCEL 1.0, its limitations, and how OCEL 2.0 enriches the metamodel of OCEL 1.0 in more detail in the following section.

3 Metamodel of the OCEL 2.0 Standard

Standards for storing object-centric event data serve as a crucial backbone in managing and analyzing complex process data. They provide a coherent, uniform, and structured approach to representing, storing, and exchanging event data across multiple systems, platforms, and applications. The adoption of a standard has several significant benefits:

- *Interoperability*: a standard promotes seamless data interchange between diverse systems. It eliminates data silos by ensuring that event logs are represented in a universally understandable format.
- *Scalability*: a well-structured standard allows efficient handling of data, enabling it to scale with increasing complexity and volume. It ensures that the data remains manageable, reducing the overhead of dealing with unstructured or inconsistently structured logs.
- *Data Integrity and Consistency*: the standardization of event logs upholds the consistency and integrity of data across different sources. It provides a uniform structure to data, making it less prone to inconsistencies and errors, thereby improving the overall data quality.
- *Simplifies Analysis*: by adhering to a standard, the interpretation and analysis of event logs are significantly simplified. It enables the use of standard analysis tools and methods, fostering easy comparability and benchmarking of results.
- *Future-Proofing*: standards also future-proof data, ensuring that it remains accessible, reusable, and comprehensible even as technologies evolve.

The first comprehensive standard for storing event data was the IEEE Standard for eXtensible Event Stream (XES) [10]. XES became an official IEEE standard in 2016 [5]. The revised standard (IEEE 1849-2023) was published on 8 September 2023 and will be valid for another ten years [9]. XES has played a major role in the development of the field. However, within the process mining community, there seems to be a consensus that a paradigm shift is needed. The development and adoption of a standard for storing object-centric event data are vital for realizing the full potential of process mining and other data-driven analytics methods. It paves the way for more effective, efficient, and reliable data management and analysis strategies.

The first version of the object-centric event log standard, OCEL 1.0, was a big step forward for object-centric process mining [7,8]. It can store various types of events and objects in one log and link objects of different types to each event, giving a more detailed picture. OCEL 1.0 also allowed for adding multiple attributes to each event and object, providing even more information. This made data analysis deeper and more insightful. We provided OCEL 1.0 specifications in both JSON and XML formats. Several OCEL 1.0 data sets were provided and the availability of the standard fueled the development of a range of OCPM techniques, e.g., discovering object-centric Petri nets, discovering object-centric DFGs, checking conformance on object-centric process models, clustering object-centric event data, object-centric predictive methods, etc. [2,3,6].

Although OCEL 1.0 can be considered a success, the time has come to extend the standard. OCEL 1.0 has a few deliberate limitations. In 2020, there were hardly any OCPM techniques, and the goal was to keep the standard as simple and lean as possible. However, with the rapid development of the field, the first OCEL standard can now be perceived as an incomplete solution for object-centric process mining. In 2021, a survey was conducted by the IEEE Task Force on Process Mining [12]. The goal was to collect requirements for a new standard succeeding XES. The online survey with 289 participants, spanning the roles of practitioners, researchers, software vendors and end-users, showed the need for supporting object-centricity. This resulted in the so-called "OCED Working Group" of the IEEE Task Force on Process Mining. Input for the discussion was an early version of the OCEL 2.0 metamodel (similar to the model in [2]). Unfortunately, the discussions in the OCED Working Group did not converge after 1.5 years of discussion. This was due to conflicting requirements (expressiveness versus simplicity), different implementation paradigms (relational versus graph-based), and a lack of clarity on who would implement things. Therefore, after a delay of two years, the OCEL team decided to release OCEL 2.0, including example event logs, reference implementations, libraries, and documentation. OCEL 2.0 aims to strike a middle ground between simplicity and expressiveness.

Figure 1 shows the meta model using a simplified UML-like notation (just using classes, associations, and multiplicities). Compared to OCEL 1.0 there are several commonalities. There are objects and events, and these are typed. Events have a timestamp and any number of additional attributes. Also, objects can have attributes. There is a many-to-many relationship between events and objects. There are also several differences. OCEL 2.0 was extended to address the limitations discussed before.

- *Object-to-Object (O2O) Relationships*: OCEL 2.0 allows a deeper understanding of how objects interact within a business process. It shows that objects are part of a complex network of relationships and actions. Capturing these relationships can reveal insights about process performance and inefficiencies and allows for advanced analytics techniques like network analysis and predictive modeling.

- *Dynamic Object Attribute Values*: OCEL 2.0 adopts a dynamic approach where attribute values can change over time. Instead of having a single, fixed value, an object attribute may have a value that changes during the process. This gives a more realistic view of process instances by recognizing that object attributes change over time due to events and progression.

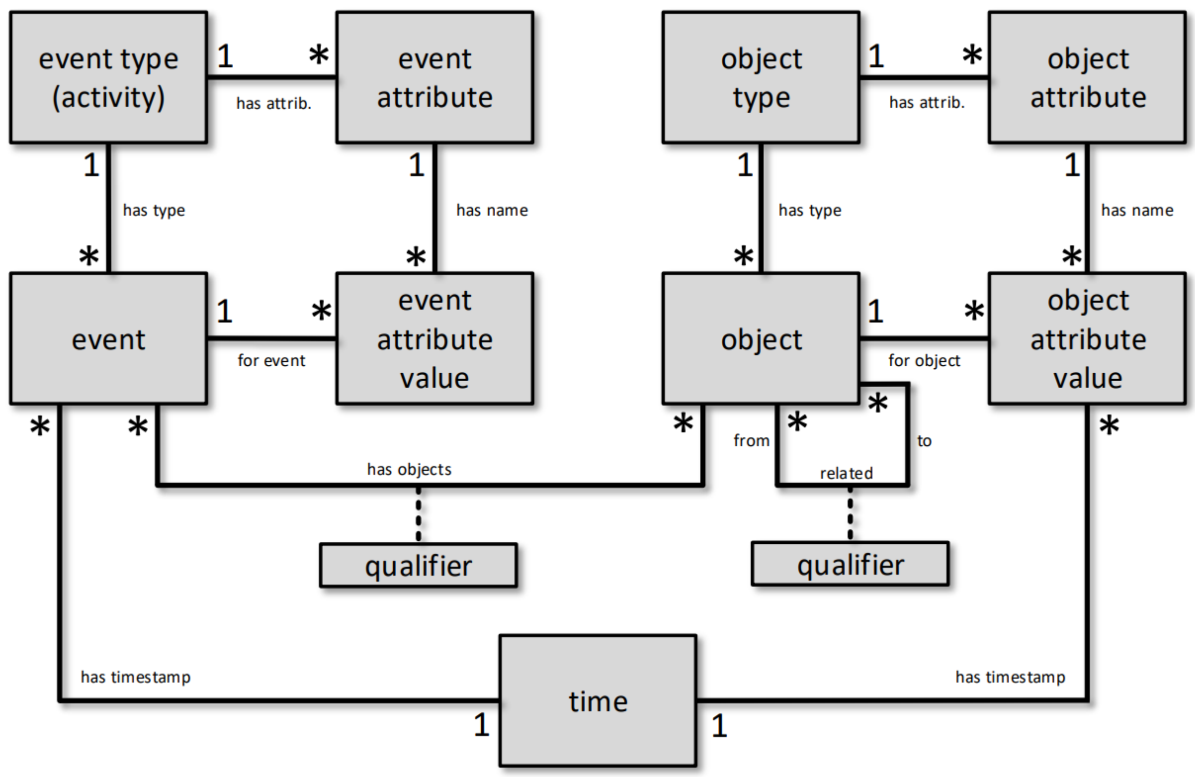


Figure 1: OCEL 2.0 metamodel

- *Relationship Qualifiers*: OCEL 2.0 offers capabilities to express qualifiers for relationships, both for Object-to-Object (O2O) and Event-to-Object (E2O) relationships. E2O relationship qualifiers describe in which role an object takes part in an event, while O2O relationship qualifiers can further characterize the association between two objects.

Apart from these conceptual extensions, the specifications have been enhanced. This makes the standard more scalable and easier to use in practical situations (e.g., in the context of a relational database). Notable differences are:

- *Relational Specification based on Dense Tables*: One major feature of OCEL 2.0 is its data structure using dense tables. Each table corresponds to a unique event or object type, storing only relevant attributes. This results in efficient use of storage space and less data redundancy. It also scales well, allowing for easy addition of new event or object types. The separate tables make the data more accessible and easy to understand, improving both efficiency and analysis in process mining.

- *Improved XML Specification*: The XML specification in OCEL 2.0 has been significantly upgraded to handle complex data better. Essential information for events and objects is now directly within the corresponding tags, making the data more readable. It also includes the ability to show object-to-object relationships and track attribute changes over time, providing a better view of how objects evolve.

4 Formal Definitions

The metamodel Figure 1 is supported by a formalization that adds more details. The theoretical foundation is crucial for understanding and using OCEL 2.0. These definitions form the basis for concrete exchange formats discussed later. The connection between theory and practice ensures that both the relational model and XML schema respect the standard's principles, enhancing its usefulness for object-centric event logging. Readers will see how these concepts turn into practical solutions, improving their understanding and use of OCEL 2.0 in process mining. We also encourage authors writing scientific papers using OCEL 2.0 to adopt these formal definitions and thus improve reliability.

First, Definition 1 introduces some concepts (universes) needed in the remainder.

Definition 1 (Universes)**.** *Let $\mathbb{U}_{\Sigma}$ be the universe of strings. We define the following pairwise disjoint universes:*

- $\mathbb{U}_{ev} \subseteq \mathbb{U}_{\Sigma}$ *is the universe of events.*
- $\mathbb{U}_{etype} \subseteq \mathbb{U}_{\Sigma}$ *is the universe of event types (i.e., activities).*
- $\mathbb{U}_{obj} \subseteq \mathbb{U}_{\Sigma}$ *is the universe of objects.*
- $\mathbb{U}_{otype} \subseteq \mathbb{U}_{\Sigma}$ *is the universe of object types.*
- $\mathbb{U}_{attr} \subseteq \mathbb{U}_{\Sigma}$ *is the universe of attribute names.*
- $\mathbb{U}_{val}$ *is the universe of attribute values.*
- $\mathbb{U}_{time}$ *is the universe of timestamps (with $0 \in \mathbb{U}_{time}$ as the smallest element and $\infty \in \mathbb{U}_{time}$ as the largest element)*
- $\mathbb{U}_{qual} \subseteq \mathbb{U}_{\Sigma}$ *is the universe of qualifiers.*

Note that the universes are assumed to be pairwise disjoint, i.e., objects cannot be used as events, etc. $e \in \mathbb{U}_{ev}$ will be used to denote an event, $et \in \mathbb{U}_{etype}$ will be used to denote an event type, $o \in \mathbb{U}_{obj}$ will be used to denote an object, $ot \in \mathbb{U}_{otype}$ will be used to denote an object type, $ea \in \mathbb{U}_{attr}$ will be used to denote an event attribute, $oa \in \mathbb{U}_{attr}$ will be used to denote an object attribute, $v \in \mathbb{U}_{val}$ will be used to denote an attribute value, $t \in \mathbb{U}_{time}$ will be used to denote a timestamp, and $q \in \mathbb{U}_{qual}$ will be used to denote a qualifier.

We assume a total ordering on timestamps, with $0 \in \mathbb{U}_{time}$ as the earliest timestamp and $\infty \in \mathbb{U}_{time}$ as the latest timestamp (i.e., for any $t \in \mathbb{U}_{time}$: $0 \leq t \leq \infty$). These are added for notational convenience, e.g., we can use 0 for missing timestamps and the start of the process, and ∞ as the end time. We would like to emphasize that these two reference timestamps (i.e., 0 and ∞) are chosen for convenience. In the formalization, time is mapped on the non-negative reals, but concrete implementations will use, for example, the ISO 8601 time format.

Definition 2 provides the formal definition for object-centric event logs, describing all the basic concepts introduced in Section 2 (events, objects, event types, object types, and event-to-object relationships), and introducing object-to-object relationships and dynamic object attribute values.

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

- *$E \subseteq \mathbb{U}_{ev}$ is the set of events.*
- *$O \subseteq \mathbb{U}_{obj}$ is the set of objects.*
- *$evtype : E \rightarrow \mathbb{U}_{etype}$ assigns types to events.*
- *$time : E \rightarrow \mathbb{U}_{time}$ assigns timestamps to events.*
- *$EA \subseteq \mathbb{U}_{attr}$ is the set of event attributes.*
- *$eatype : EA \rightarrow \mathbb{U}_{etype}$ assigns event types to event attributes.*
- *$eaval : (E \times EA) \not\rightarrow \mathbb{U}_{val}$ assigns values to event attributes (not all the attributes are mapped for each event).*
- *$objtype : O \rightarrow \mathbb{U}_{otype}$ assigns types to objects.*
- *$OA \subseteq \mathbb{U}_{attr}$ is the set of object attributes.*
- *$oatype : OA \rightarrow \mathbb{U}_{otype}$ assigns object types to object attributes.*
- *$oaval : (O \times OA \times \mathbb{U}_{time}) \not\rightarrow \mathbb{U}_{val}$ assigns values to object attributes.*
- *$E2O \subseteq E \times \mathbb{U}_{qual} \times O$ are the qualified event-to-object relations.*
- *$O2O \subseteq O \times \mathbb{U}_{qual} \times O$ are the qualified object-to-object relations.*

such that

- *$dom(eaval) \subseteq \{(e, ea) \in E \times EA \mid evtype(e) = eatype(ea)\}$ to ensure that only existing event attributes can have values.*
- *$dom(oaval) \subseteq \{(o, oa, t) \in O \times OA \times \mathbb{U}_{time} \mid objtype(o) = oatype(oa)\}$ to ensure that only existing object attributes can have values.*

The stipulations embedded in the final two criteria of the definition ensure that each event and object is limited to possessing attribute values pertinent to its respective event or object type. Additionally, these guidelines also mandate that the set of attributes is distinct and non-overlapping for each individual event and object type, guaranteeing a disjoint attribute set across all types.

In order to facilitate the following examples and explanations, we introduce the following notations given an OCEL L:

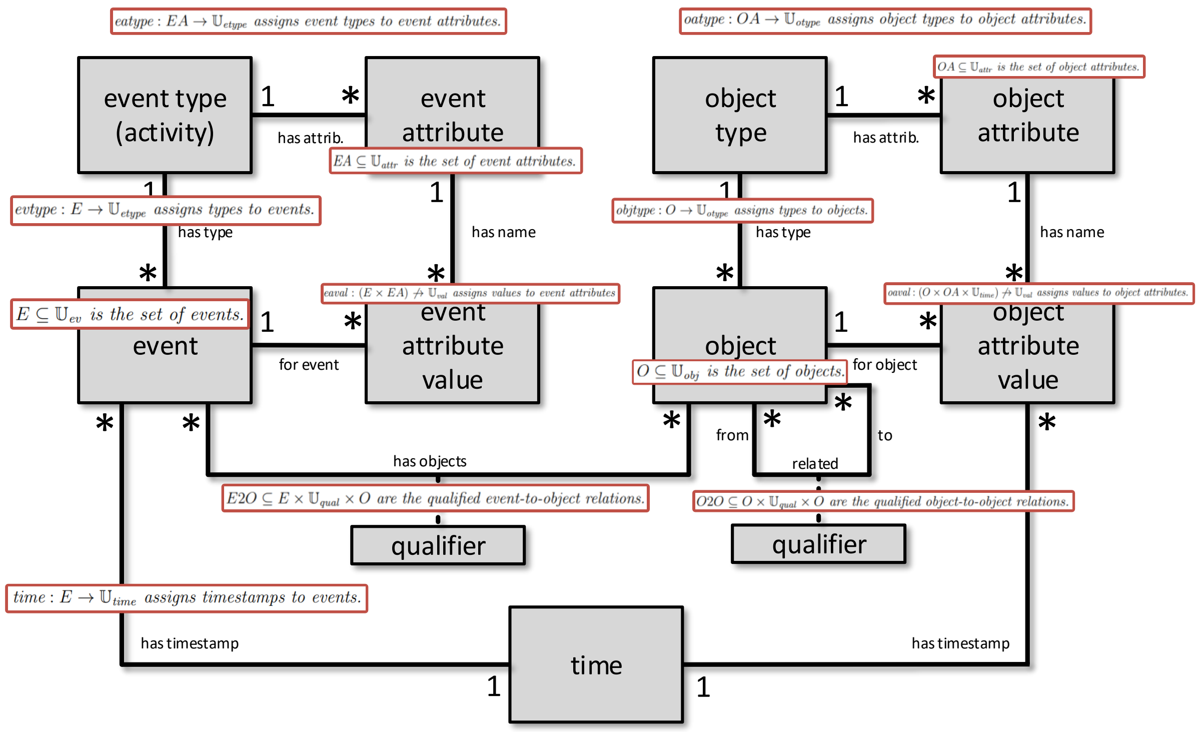
- $ET(L) = \{\text{evtype}(e) \mid e \in E\}$ is the set of event types.
- $OT(L) = \{\text{objtype}(o) \mid o \in O\}$ is the set of object types.
- For any event $e \in E$ and event attribute ea $\in \mathbb{U}_{attr}$:
  - $\text{eaval}_{ea}(e) = \text{eaval}(e, ea)$ if $(e, ea) \in \text{dom(eaval)}$.
  - $\text{eaval}_{ea}(e) = \perp$ if $(e, ea) \notin \text{dom(eaval)}$.
- For any object $o \in O$, object attribute oa $\in \mathbb{U}_{attr}$ and time $t \in \mathbb{U}_{time}$:

  - $\text{oaval}^t_{oa}(o) = \text{oaval}(o, oa, t')$ if there exists a $t' \in \mathbb{U}_{time}$ such that $t' \leq t$ and $(o, oa, t') \in \text{dom}(\text{oaval})$ such that there is no $t'' \in \mathbb{U}_{time}$ such that $t' < t'' \leq t$ and $(o, oa, t'') \in \text{dom}(\text{oaval})$.
  - If no such t' exists, then $\text{oaval}^t_{oa}(o) = \bot$.

  Hence, oaval^t_{oa} provides us with the latest object attribute value at time t.

- $\text{oaval}_{oa}(o) = \text{oaval}^{\infty}_{oa}(o)$ is the final value for the object attribute in the event log.

Note that oaval describes object attribute updates at particular points in time. Function $\text{oaval}^t_{oa}(o)$ allows us to determine the value of object attributes at any point in time, thus clarifying the semantics. A missing timestamp for a value assignment can be interpreted as time 0. If this is the only value assignment for an object-attribute combination, then the value for an object attribute is always the same. This way, we can handle static object-attribute values.



$eatype : EA \rightarrow \mathbb{U}_{etype}$ assigns event types to event attributes.
$oatype : OA \rightarrow \mathbb{U}_{otype}$ assigns object types to object attributes.
event type (activity)
1
has attrib.
*
event attribute
$EA \subseteq \mathbb{U}_{attr}$ is the set of event attributes.
object type
1
has attrib.
*
$OA \subseteq \mathbb{U}_{attr}$ is the set of object attributes.
object attribute
1
$evtype : E \rightarrow \mathbb{U}_{etype}$ assigns types to events.
has type
1
has name
1
$objtype : O \rightarrow \mathbb{U}_{otype}$ assigns types to objects.
has type
1
has name
*
*
$eaval : (E \times EA) \not\rightarrow \mathbb{U}_{val}$ assigns values to event attributes
*
*
$oaval : (O \times OA \times \mathbb{U}_{time}) \not\rightarrow \mathbb{U}_{val}$ assigns values to object attributes.
$E \subseteq \mathbb{U}_{ev}$ is the set of events.
event
1
for event
*
event attribute value
object
$O \subseteq \mathbb{U}_{obj}$ is the set of objects.
1
for object
*
object attribute value
*
*
*
from
*
*
to
*
has objects
related
$E2O \subseteq E \times \mathbb{U}_{qual} \times O$ are the qualified event-to-object relations.
$O2O \subseteq O \times \mathbb{U}_{qual} \times O$ are the qualified object-to-object relations.
qualifier
qualifier
$time : E \rightarrow \mathbb{U}_{time}$ assigns timestamps to events.
has timestamp
time
has timestamp
1
1

Figure 2: Linking the OCEL 2.0 metamodel to the formalization in Definition 2.

Object attribute values are deliberately *not* connected to events. The definition allows for events without objects or objects without events. Events should correspond to relevant activities, and therefore, there should not be the need to promote individual object attribute changes to events. Also, the object-to-object relations may exist independent of events. The only explicit connections between events and objects are the event-to-object relations (i.e., *E2O*). However, for an event e happening at time t involving object o with attribute oa we can look up the corresponding value at the time of the event via $\text{oaval}^t_{oa}(o)$. Hence, there is no limitation.

Figure 2 links the OCEL metamodel in Figure 1 to the formalization just provided. Sections 6, 7, and 8 map the formalization onto concrete relational, XML, and JSON formats.

5 Running Example

In order to visualize the concepts of Definition 2 and propose some implementations of OCEL 2.0, we describe an example object-centric setting supporting the following situations:

1. A purchase requisition (**PR1**) is created and approved, requesting 500 cows, and a purchase order (**PO1**) is created on top of the purchase requisition. The quantity of the purchase order **PO1** is then changed to 600 cows. Two invoices **R1** and **R2** are received (for the first 500 cows and for the additional 100 cows), which are paid separately by the payments **P1** and **P2**.

2. Mario is an unethical employee who places purchase orders of notebooks without getting proper approval. An invoice **R3** is received before an order **PO2** is formally inserted in the system. Therefore, Sam, who is a financial controller, blocks the payment of the invoice. However, Mario manages to override Sam and puts himself as an approver of the invoice; therefore, the invoice is paid in the **P4** payment.

In this example, the attributes at the object level evolve, highlighting one of the main features of OCEL 2.0. Also, we provide qualified event-to-object and object-to-object relationships, specifically in Table 20 and Table 21, highlighting the other main novelties.

For this example:

- We identify the following sets (in relation to Definition 2):
  - (sets of objects) $O = \{\mathbf{PR1}, \mathbf{PO1}, \mathbf{R1}, \mathbf{R2}, \mathbf{P1}, \mathbf{P2}, \mathbf{R3}, \mathbf{PO2}, \mathbf{R3}, \mathbf{P3}\}$.
  - (sets of events) $E = \{e1, e2, e3, e4, e5, e6, e7, e8, e9, e10, e11, e12, e13\}$.
  - (sets of attributes at the event level) $EA = \{\text{pr\_creator}, \text{pr\_approver}, \text{po\_creator}, \text{po\_editor}, \text{invoice\_inserter}, \text{invoice\_blocker}, \text{invoice\_block\_rem}, \text{payment\_inserter}\}$ where:
    - **pr_creator** is the resource that created the purchase requisition in the system.
    - **pr_approver** is the resource that approved the purchase requisition in the system.
    - **po_creator** is the resource that created the purchase order in the system.
    - **po_editor** is the resource that changed some parameters of the purchase order.
    - **invoice_inserter** is the resource that inserted the invoice in the system.
    - **invoice_blocker** is the resource that blocked the payment of a given invoice.
    - **invoice_block_rem** is the resource that removed the payment block.
    - **payment_inserter** is the resource that performed the payment.
  - (sets of attributes at the object level) $OA = \{\text{pr\_product}, \text{pr\_quantity}, \text{po\_product}, \text{po\_quantity}, \text{is\_blocked}\}$ where:
    - **pr_product** is the product requested in the purchase requisition.
    - **pr_quantity** is the quantity requested in the purchase requisition.
    - **po_product** is the product bought in the purchase order.

  - **po_quantity** is the quantity bought in the purchase order.
  - **is_blocked** relates to a payment block on the invoice.

- The object types corresponding to the given objects (objtype in Definition 2) are described in Table 1. Moreover, the attributes defined for the object types (oatype in Definition 2) are also described in Table 1.
- The event types are described in Table 2, along with the corresponding attributes (eatype in Definition 2) and event identifiers.
- A high-level tabular view is provided in Table 3. In particular, the event type (evtype in Definition 2) and timestamp (time in Definition 2) of each event is defined.

Table 1: High-level view on the object types of the object-centric event log (running example).

Object Type	**Attributes**	**Object IDs**
Purchase Requisition	pr_product, pr_quantity	**PR1**
Purchase Order	po_product, po_quantity	**PO1, PO2**
Invoice	is_blocked	**R1, R2, R3**
Payment	$\emptyset$	**P1, P2**

Table 2: High-level view on the event types of the object-centric event log (running example).

Event Type	**Attributes**	**Event IDs**
Create Purchase Requisition	pr_creator	e1
Approve Purchase Requisition	pr_approver	e2
Create Purchase Order	po_creator	e3, e10
Change PO Quantity	po_editor	e4
Insert Invoice	invoice_inserter	e5, e6, e9
Set Payment Block	invoice_blocker	e11
Remove Payment Block	invoice_block_rem	e12
Insert Payment	payment_inserter	e7, e8, e13

Table 3: High-level tabular view on the object-centric event log (running example), showing the list of events along with the related objects. In this view, no event/object attribute is reported, no qualifier is reported (see Table 20 for the event-to-object relationships qualifiers), and no object-to-object relationship is described.

Event ID	Event Type	Timestamp	Related Objects
e1	Create Purchase Requisition	2022-01-09 15:00	**PR1**
e2	Approve Purchase Requisition	2022-01-09 16:30	**PR1**
e3	Create Purchase Order	2022-01-10 09:15	**PR1, PO1**
e4	Change PO Quantity	2022-01-13 12:00	**PO1**
e5	Insert Invoice	2022-01-14 12:00	**PO1, R1**
e6	Insert Invoice	2022-01-16 11:00	**PO1, R2**
e7	Insert Payment	2022-01-30 23:00	**R1, P1**
e8	Insert Payment	2022-01-31 22:00	**R2, P2**
e9	Insert Invoice	2022-02-02 09:00	**R3**
e10	Create Purchase Order	2022-02-02 17:00	**R3, PO2**
e11	Set Payment Block	2022-02-03 07:30	**R3**
e12	Remove Payment Block	2022-02-03 23:30	**R3**
e13	Insert Payment	2022-02-28 23:00	**R3, P3**

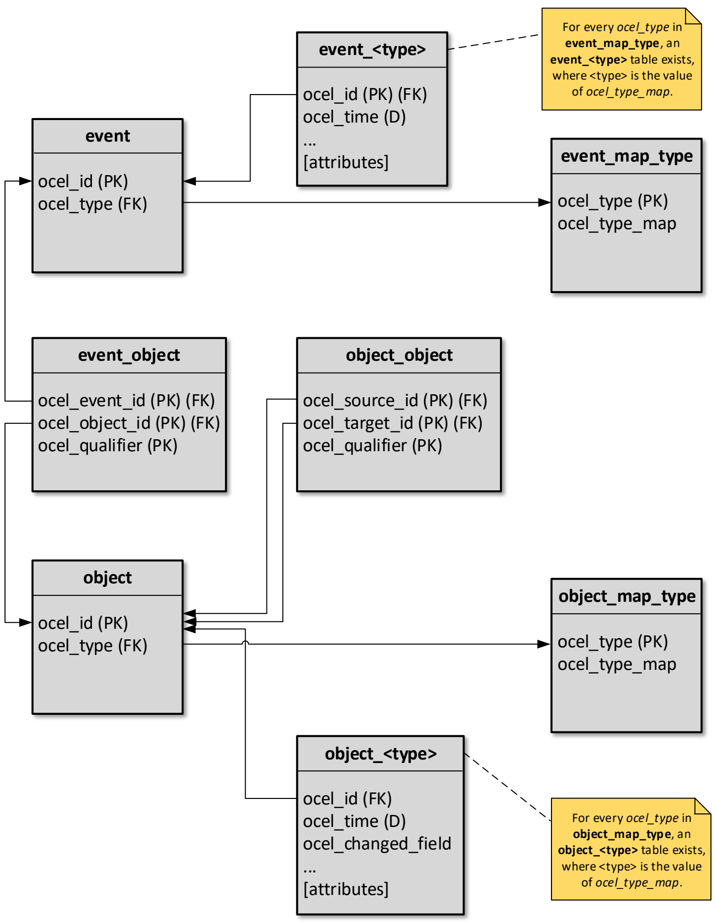


Figure 3: General relational schema of the proposed relational implementation.

6 Relational SQLite Format

We propose a relational implementation of the standard, which adheres to Definition 2. In this implementation, starting from an object-centric event log L, we have:

- We have a table, **event_map_type**, reporting the distinct event types ($ET(L)$); and a table, **object_map_type**, reporting the distinct object types ($OT(L)$).
- We have a table, **event**, reporting the event type (evtype in Definition 2) for each event, and a table, **object**, reporting the object type (objtype in Definition 2) for each object. This is described in Subsection 6.2.
- For every event type in $ET(L)$, we have a different table. More specifically, if et $\in ET(L)$ is the event type, we have a table called **event_**$\oplus map_{ET}$**(et)**, where $map_{ET} : ET(L) \rightarrow \mathbb{U}_{\Sigma}$ is an injective function[1] mapping the event types to unique identifiers, containing the events of the given event type along with their timestamp and attributes. This is described in Subsection 6.3.
- For every object type in $OT(L)$, we have a different table. More specifically, if $ot \in OT(L)$ is the object type, we have a table called **object_**$\oplus map_{OT}$**(ot)**, where $map_{OT} : OT(L) \rightarrow \mathbb{U}_{\Sigma}$ is an injective function[2] mapping the object types to unique identifiers, containing the objects of the given object type along with the history of their attributes. This is described in Subsection 6.4.
- We have a table, **event_object**, containing the event-to-object relationships (E2O in Definition 2). This is described in Subsection 6.5.
- We have a table, **object_object**, containing the object-to-object relationships (O2O in Definition 2). This is described in Subsection 6.6.

A general representation of the relational schema of OCEL 2.0 is portrayed in Figure 3. The overall relational schema of the running example OCEL 2.0 log is shown in Figure 4.

[1]Which can be the identity function.
[2]Which can be the identity function.

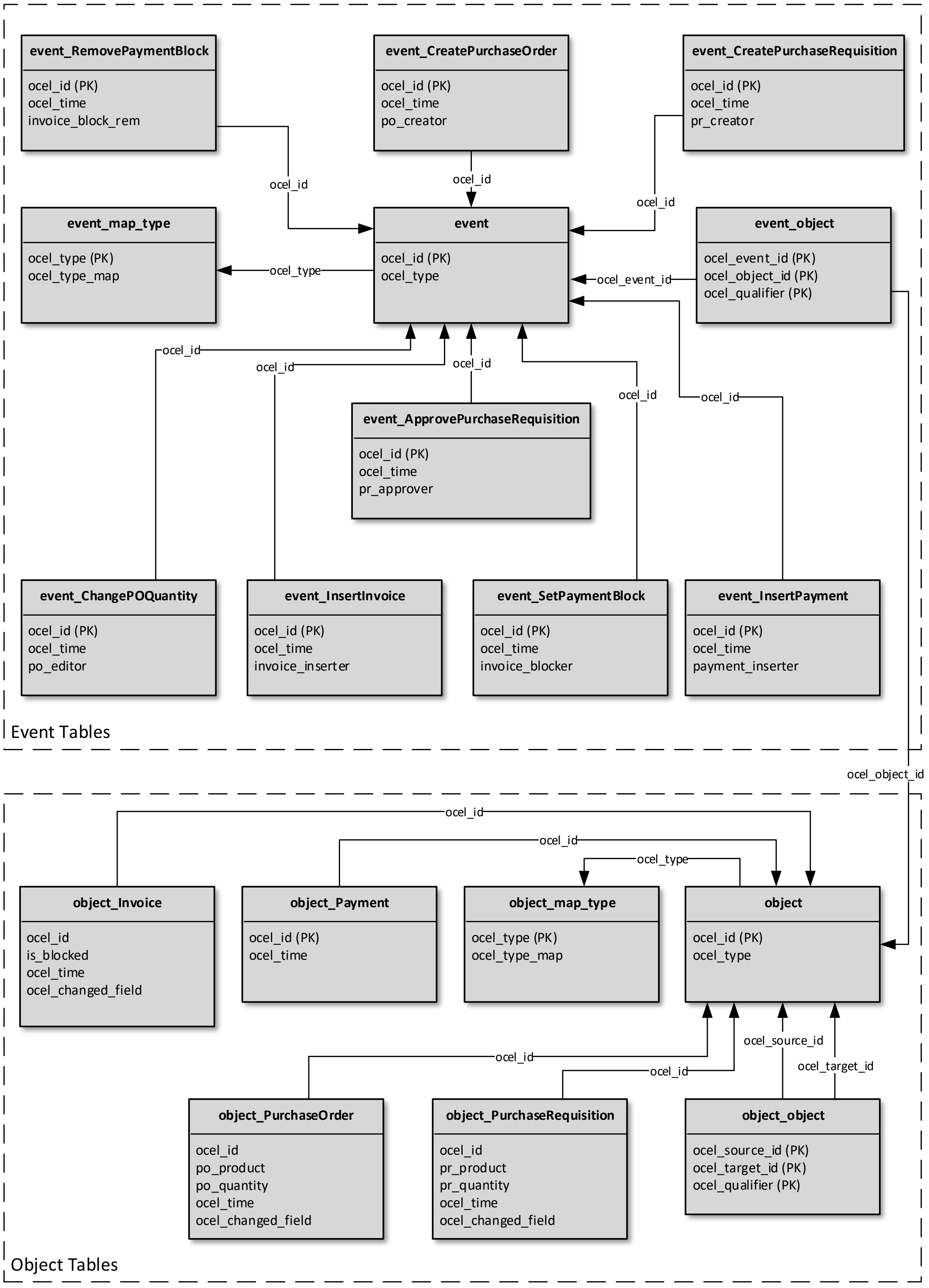


Figure 4: Relational schema of the running example OCEL 2.0 log.

6.1 Tables for the Distinct Event/Object Types

In our implementation, we define the table **event_map_type** (Table 4) defining the event types along with their unique identifier obtained by applying the injective function $map_{ET} : ET(L) \rightarrow \mathbb{U}_{\Sigma}$. This unique identifier is used to link the event type to a table containing the attributes of the events of the given event type (see Subsection 6.3). The event type should be set as the primary key to avoid duplication.

Table 4: [Proposed relational implementation] Distinct event types: **event_map_type**

ocel_type [PK]	**ocel_type_map**
Approve Purchase Requisition	ApprovePurchaseRequisition
Change PO Quantity	ChangePOQuantity
Create Purchase Order	CreatePurchaseOrder
Create Purchase Requisition	CreatePurchaseRequisition
Insert Invoice	InsertInvoice
Insert Payment	InsertPayment
Remove Payment Block	RemovePaymentBlock
Set Payment Block	SetPaymentBlock

Also, the table **object_map_type** (Table 5) defines the object types along with their unique identifier obtained applying the injective function $map_{OT} : OT(L) \rightarrow \mathbb{U}_{\Sigma}$. This unique identifier is used to link the object type to a table containing the attributes of the objects of the given object type (see Subsection 6.4). The object type should be set as the primary key to avoid duplication.

Table 5: Proposed relational implementation: Distinct object types

ocel_type [PK]	**ocel_type_map**
Invoice	Invoice
Payment	Payment
Purchase Order	PurchaseOrder
Purchase Requisition	PurchaseRequisition

6.2 Events and Objects Tables

In the proposed implementation, we use several tables to store information related to an event/object type. However, we also create two additional tables, hosting the event/object identifiers. This is done to map the event/object to a specific event/object type table, and to allow for the definition of E2O/O2O tables with proper foreign keys (directed towards the events and the objects tables).

This is done in Table 6 (having the event identifier, *ocel_id*, as primary key) and Table 7 (having the object identifier, *ocel_id*, as primary key).

Table 6: [Proposed Relational Implementation] General Events Table
event

ocel_id [PK]	**ocel_type FK**
e1	Create Purchase Requisition
e2	Approve Purchase Requisition
e3	Create Purchase Order
e4	Change PO Quantity
e5	Insert Invoice
e6	Insert Invoice
e7	Insert Payment
e8	Insert Payment
e9	Insert Invoice
e10	Create Purchase Order
e11	Set Payment Block
e12	Remove Payment Block
e13	Insert Payment

Table 7: [Proposed Relational Implementation] General Objects Table
object

ocel_id [PK]	**ocel_type FK**
PR1	Purchase Requisition
PO1	Purchase Order
PO2	Purchase Order
R1	Invoice
R2	Invoice
R3	Invoice
P1	Payment
P2	Payment

6.3 Event Type Tables

Having defined the table **event** as the general table for the events (having the identifier of the event as the primary key), we define event-type-specific tables. These allow for the storage of event-type-specific attributes without having a gigantic "sparse table" hosting all the events with all the different attributes (with an empty value for most of them). Therefore, if $et \in ET(L)$ is the event type, we have a table called **event_**$\oplus map_{ET}$**(et)**, where $map_{ET} : ET(L) \rightarrow \mathbb{U}_{\Sigma}$ is an injective function[3] mapping the event types to unique identifiers, containing the events of the given event type along with their timestamp and attributes. In particular, all the attributes associated with the given event type (eatype in Definition 2) are columns of the table, and the values for each event are populated accordingly (eaval in Definition 2). Moreover, the timestamp is a column of the event-type-specific tables. The event identifier column is pointing (as a foreign key) to the event identifier column of the **event** table. Examples follow (Table 8, Table 9, Table 10, Table 11, Table 12, Table 13, Table 14, Table 15) for all the event types of the running example log.

Table 8: [Proposed relational implementation] Event Type Table: **event_CreatePurchaseRequisition**

ocel_id [PK] [FK]	**ocel_time**	**pr_creator**
e1	2022-01-09 15:00	Mike

Table 9: [Proposed relational implementation] Event type table: **event_ApprovePurchaseRequisition**

ocel_id [PK] [FK]	**ocel_time**	**pr_approver**
e2	2022-01-09 16:30	Tania

Table 10: [Proposed relational implementation] Event type table: **event_CreatePurchaseOrder**

ocel_id [PK] [FK]	**ocel_time**	**po_creator**
e3	2022-01-10 09:15	Mike
e10	2022-02-02 17:00	Mario

[3] Which can be the identity function.

Table 11: [Proposed relational implementation] Event type table: **event_ChangePOQuantity**

ocel_id [PK] [FK]	**ocel_time**	**po_editor**
e4	2022-01-13 12:00	Mike

Table 12: [Proposed relational implementation] Event type table: **event_InsertInvoice**

ocel_id [PK] [FK]	**ocel_time**	**invoice_inserter**
e5	2022-01-14 12:00	Luke
e6	2022-01-16 11:00	Luke
e9	2022-02-02 09:00	Mario

Table 13: [Proposed relational implementation] Event type table: **event_SetPaymentBlock**

ocel_id [PK] [FK]	**ocel_time**	**invoice_blocker**
e11	2022-02-03 07:30	Sam

Table 14: [Proposed relational implementation] Event type table: **event_RemovePaymentBlock**

ocel_id [PK] [FK]	**ocel_time**	**invoice_block_rem**
e12	2022-02-03 23:30	Mario

Table 15: [Proposed relational implementation] Event type table: **event_InsertPayment**

ocel_id [PK] [FK]	**ocel_time**	**payment_inserter**
e7	2022-01-30 23:00	Robot
e8	2022-01-31 22:00	Robot
e13	2022-02-28 23:00	Robot

6.4 Object Type Tables

Having defined the table **object** as the general table for the objects (having the identifier of the object as the primary key), we want to define object-type-specific tables for equivalent reasons to the ones explained in Subsection 6.3. Therefore, if $ot \in OT(L)$ is the object type, we have a table called **object_**$\oplus map_{OT}$**(ot)**, where $map_{OT} : OT(L) \rightarrow \mathbb{U}_{\Sigma}$ is an injective function[4] mapping the object types to unique identifiers, containing the objects of the given object type along with the history of their attributes. In particular, all the attributes associated with the given object type (oatype in Definition 2) are columns of the table, and the values for each object are populated accordingly. The timestamp column here highlights the history of the values of the different attributes. We made the following design choices:

- In accordance with the definition of OCEL 2.0, rows possessing the smallest feasible timestamp, specifically *1970-01-01 00:00 UTC* - which equates to 0 in Definition 2 - correspond to the initial values of all the attributes for a given object. This uniform timestamp selection, symbolic of the starting point or "epoch", facilitates a consistent reference frame across all objects, aligning seamlessly with the structure proposed by OCEL 2.0. The deliberate choice of this fixed date for the timestamp of 0 is not arbitrary; instead, it serves a clear purpose by fostering improved compatibility and coherence with the OCEL 2.0 definition.
- The rows having a different timestamp have a value in the column *ocel_changed_field*, which highlights which other column has been changed.

This allows (according to *oaval* in Definition 2) to reconstruct the value of a given attribute at a specified point in time. For example, given the purchase order **PO1**, the quantity of the order at **2022-01-11 10:00** is 500 cows, but it becomes 600 cows at **2022-01-13 13:00**. Also, consider the invoice $R3$, which was blocked and then released.

The object identifier column is pointing (as a foreign key) to the object identifier column of the **object** table.

Examples follow (Table 16, Table 17, Table 18, Table 19) for all the object types of the running example log.

Table 16: [Proposed relational implementation] Object type table: **object_PurchaseRequisition**

ocel_id [FK]	**ocel_time**	**pr_product**	**pr_quantity**	**ocel_changed_field**
PR1	1970-01-01 00:00 UTC	Cows	500	

[4] Which can be the identity function.

Table 17: [Proposed relational implementation] Object type table: **object_PurchaseOrder**

ocel_id [FK]	**ocel_time**	**po_product**	**po_quantity**	**ocel_changed_field**
PO1	1970-01-01 00:00 UTC	Cows	500	
PO1	2022-01-13 12:00 UTC		600	po_quantity
PO2	1970-01-01 01:00 UTC	Notebooks	1	

Table 18: [Proposed relational implementation] Object type table: **object_Invoice**

ocel_id [FK]	**ocel_time**	**is_blocked**	**ocel_changed_field**
R1	1970-01-01 00:00 UTC	No	
R2	1970-01-01 00:00 UTC	No	
R3	1970-01-01 00:00 UTC	No	
R3	2022-02-03 07:30 UTC	Yes	is_blocked
R3	2022-02-03 23:30 UTC	No	is_blocked

Table 19: [Proposed relational implementation] Object type table: **object_Payment**

ocel_id [FK]	**ocel_time**	**ocel_changed_field**
P1	1970-01-01 00:00 UTC	
P2	1970-01-01 00:00 UTC	
P3	1970-01-01 00:00 UTC	

6.5 Event-to-Object (E2O) Relationships

The **event_object** table contains the event-to-object relationships (E2O in Definition 2). Therefore, it contains the correlated event identifier and object identifier (foreign key to **event.ocel_id** and **object.ocel_id** respectively) with a qualifier explaining the nature of the relationship. A primary key is set on **event_object** containing all the columns, therefore "realizing" the set proposed in Definition 2. The **event_object** table of the running example is proposed in Table 20. Note that we can have an event related with different qualifiers to the same object.

Table 20: [Proposed relational implementation] Table containing the event-to-object (**event_object**) relationships

ocel_event_id [PK] [FK]	**ocel_object_id [PK] [FK]**	**ocel_qualifier [PK]**
e1	PR1	Regular placement of PR
e2	PR1	Regular approval of PR
e3	PR1	Created order from PR
e3	PO1	Created order with identifier
e4	PO1	Change of quantity
e5	PO1	Invoice created starting from the PO
e5	R1	Invoice created with identifier
e5	PO1	Invoice created starting from the PO
e6	R2	Invoice created with identifier
e6	PO1	Invoice created starting from the PO
e7	R1	Payment for the invoice
e7	P1	Payment inserted with identifier
e8	R2	Payment for the invoice
e8	P2	Payment inserted with identifier
e9	R3	Invoice created with identifier
e10	R3	Purchase order created with maverick buying from
e10	PO2	Purchase order created with identifier
e11	R3	Payment block due to unethical maverick buying
e12	R3	Payment block removed ...
e13	R3	Payment for the invoice
e13	P3	Payment inserted with identifier

6.6 Object-to-Object (O2O) Relationships

The **object_object** table contains the object-to-object relationships (O2O in Definition 2). Therefore, it contains the correlated object identifiers (source and target; both are foreign keys to **object.ocel_id**) with a qualifier explaining the nature of the relationship. A primary key is set on **object_object** containing all the columns, therefore "realizing" the set proposed in Definition 2. The **object_object** table of the running example is proposed in Table 21. Note that we can have the same couple of objects related through different qualifiers.

Table 21: [Proposed relational implementation] Table containing the object-to-object (**object_object**) relationships

ocel_source_id [PK] [FK]	**ocel_target_id [PK] [FK]**	**ocel_qualifier [PK]**
PR1	PO1	PO from PR
PO1	R1	Invoice from PO
PO1	R2	Invoice from PO
R1	P1	Payment from invoice
R2	P2	Payment from invoice
PO2	R3	Maverick buying
R3	P3	Payment from invoice

6.7 Constraints on the Relational Implementation

Clearly, all the elements introduced using the metamodel can be mapped onto the table structures proposed. However, we need to ensure consistency. Therefore, the following constraints are implemented on the proposed relational implementation:

- The uniqueness of the event/object types in the tables **event_map_type** and **object_map_type** is ensured by setting the type as the primary key.
- The uniqueness of the events/objects in the overall **event** and **object** tables is ensured by setting the identifier as the primary key.
- The uniqueness of the event identifiers in the specific event type tables is ensured by setting the identifier as the primary key. Since the same objects can appear (by design choice) several times in the specific object type tables, we cannot set the identifier as the primary key in the given setting.
- There is a foreign key between the specific event type tables and the generic **event** table. This ensures that every identifier appearing in the specific event type tables has been added to the **event** table.
- There is a foreign key between the specific object type table and the generic **object** table. This ensures that every identifier appearing in the specific object type tables has been added to the **object** table.
- There is a foreign key between the generic **event** table and **event_map_type**, ensuring that every event can be mapped to a specific event type table.
- There is a foreign key between the generic **object** table and **object_map_type**, ensuring that every object is mapped to a specific object type table.
- The **event_object** table has two foreign keys, directed towards **event.ocel_id** and **object.ocel_id** respectively. This ensures that every identifier appearing in the **event_object** table is effectively an event or an object. Moreover, the three columns together form the primary key, ensuring that no duplicate rows are contained in the table.
- The **object_object** table has two foreign keys, both directed towards **object.ocel_id**. This ensures that every identifier appearing in the **object_object** table is effectively an object. Moreover, the three columns together form the primary key, ensuring that no duplicate rows are contained in the table.

We can ensure that every event/object is added to the correct event type/object type table by a trigger that checks during the insertion time the **event_map_type/object_map_type** tables to compare the name of the current table with the expected table into which the event/object should be assigned.

We also provide some offline validation constraints as SQL queries. These are available at https://www.ocel-standard.org/2.0/ocel20-schema-relational.pdf.

7 XML Format

We propose an XML implementation following Definition 2. The timestamps are assumed to follow the ISO format specification `https://en.wikipedia.org/wiki/ISO_8601`.

The XML schema is organized as follows. There is a root element with the tag **log**. The log element has the following children:

- An element with tag **object-types**, containing as many **object-type** elements as types in $OT(L)$. Each **object-type** has a **name** property (which is the object type) and a single child with tag **attributes**:
  - For every attribute in oatype(ot), the element **attributes** has a child with tag **attribute** and properties **name** (which is the attribute) and **type** (the type of the attribute, which should be considered during the parsing of the values).
- An element with tag **event-types**, containing as many **event-type** elements as types in $OT(L)$. Each **event-type** has a **name** property (which is the event type) and a single child with tag **attributes**:
  - For every attribute in eatype(et), the element **attributes** has a child with tag **attribute** and properties **name** (which is the attribute) and **type** (the type of the attribute, which should be considered during the parsing of the values).
- An element with tag **events**, containing as many **event** elements as many events are in E. An **event** is characterized by:
  - Its properties **id** (the identifier of the event), **type** (the event type of the event, given by evtype in Definition 2) and **time** (the timestamp of the event, given by time in Definition 2).
  - A child with tag **objects**, containing the related objects to the event (E2O in Definition 2; we define the function:

    $$\text{relobj}(e) = \{(o, q) \mid (e', q, o) \in \text{E2O} \ \wedge \ e' = e\}$$

    which associates to every event a set of objects along with the qualifier of the relationship). In particular, for every event-to-object relationship an entry **relobj** is created, having as properties the **object-id** (related object identifier) and **qualifier** (the qualifier of the event-to-object relationship).
  - A child with tag **attributes**, having as many children **attribute** as many attributes are related to the event (the domain of eaval$_{ea}$ in Definition 2):
    * Each **attribute** is characterized by a **name** property and the corresponding value is reported as text of the (XML) element.
- An element with tag **objects**, containing as many **object** elements as many objects are in O. An **object** is characterized by:
  - Its properties **id** (the identifier of the object) and **type** (the object type of the object, given by objtype in Definition 2).
  - An element with tag **attributes**, containing the different **attribute** of the object.

– Each **attribute** is characterized by a **time** property (the timestamp in which the value for the given attribute was recorded), a **name** property, and the corresponding value is reported as the text of the (XML) element. An attribute is valid from the specified **time** (until an attribute with the same name and greater timestamp is recorded).

– A child with tag **objects**, containing the related objects to the given object (O2O in Definition 2; we define the function:

$$\text{relobj}(o) = \{(o'', q) \mid (o', q, o'') \in \text{O2O} \ \land \ o' = o\}$$

which associates to every object a set of objects along with the qualifier of the relationship). In particular, for every object-to-object relationship an entry **relobj** is created, having as properties the **object-id** (related object identifier) and **qualifier** (the qualifier of the object-to-object relationship).

In the remainder of this section, we show an example file and the XSD (XML Schema Definition) that can be used to check consistency.

7.1 XML Example

An example (on the running example log) follows.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<log>
  <object-types>
    <object-type name="Invoice">
      <attributes>
        <attribute name="is_blocked" type="string"/>
      </attributes>
    </object-type>
    <object-type name="Payment">
      <attributes/>
    </object-type>
    <object-type name="Purchase Order">
      <attributes>
        <attribute name="po_product" type="string"/>
        <attribute name="po_quantity" type="string"/>
      </attributes>
    </object-type>
    <object-type name="Purchase Requisition">
      <attributes>
        <attribute name="pr_product" type="string"/>
        <attribute name="pr_quantity" type="string"/>
      </attributes>
    </object-type>
  </object-types>
  <event-types>
    <event-type name="Approve Purchase Requisition">
      <attributes>
        <attribute name="pr_approver" type="string"/>
      </attributes>
    </event-type>
    <event-type name="Change PO Quantity">
      <attributes>
        <attribute name="po_editor" type="string"/>
      </attributes>
    </event-type>
    <event-type name="Create Purchase Order">
      <attributes>
        <attribute name="po_creator" type="string"/>
      </attributes>
    </event-type>
    <event-type name="Create Purchase Requisition">
      <attributes>
        <attribute name="pr_creator" type="string"/>
      </attributes>
    </event-type>
    <event-type name="Insert Invoice">
      <attributes>
        <attribute name="invoice_inserter" type="string"/>
      </attributes>
    </event-type>
    <event-type name="Insert Payment">
      <attributes>
        <attribute name="payment_inserter" type="string"/>
      </attributes>
    </event-type>
    <event-type name="Remove Payment Block">
      <attributes>
        <attribute name="invoice_block_rem" type="string"/>
      </attributes>
    </event-type>
    <event-type name="Set Payment Block">
      <attributes>
        <attribute name="invoice_blocker" type="string"/>
      </attributes>
    </event-type>
  </event-types>
  <objects>
    <object id="R1" type="Invoice">
      <attributes>
        <attribute name="is_blocked" time="1970-01-01T00:00:00Z">No</attribute>
      </attributes>
      <objects>
        <relationship object-id="P1" qualifier="Payment from invoice"/>
      </objects>
    </object>
    <object id="R2" type="Invoice">
      <attributes>
        <attribute name="is_blocked" time="1970-01-01T00:00:00Z">No</attribute>
      </attributes>
      <objects>
        <relationship object-id="P2" qualifier="Payment from invoice"/>
      </objects>
    </object>
    <object id="R3" type="Invoice">
      <attributes>
        <attribute name="is_blocked" time="1970-01-01T00:00:00Z">No</attribute>
        <attribute name="is_blocked" time="2022-02-03T07:30:00Z">Yes</attribute>
        <attribute name="is_blocked" time="2022-02-03T23:30:00Z">No</attribute>
      </attributes>
      <objects>
        <relationship object-id="P3" qualifier="Payment from invoice"/>
      </objects>
    </object>
    <object id="P1" type="Payment">
      <attributes/>
    </object>
    <object id="P2" type="Payment">
      <attributes/>
    </object>
    <object id="P3" type="Payment">
      <attributes/>
    </object>
    <object id="PO1" type="Purchase Order">
      <attributes>
        <attribute name="po_product" time="1970-01-01T00:00:00Z">Cows</attribute>
        <attribute name="po_quantity" time="1970-01-01T00:00:00Z">500</attribute>
        <attribute name="po_quantity" time="2022-01-13T12:00:00Z">600</attribute>
      </attributes>
      <objects>
        <relationship object-id="R1" qualifier="Invoice from PO"/>
        <relationship object-id="R2" qualifier="Invoice from PO"/>
      </objects>
    </object>
    <object id="PO2" type="Purchase Order">
      <attributes>
        <attribute name="po_product" time="1970-01-01T00:00:00Z">Notebooks</attribute>
        <attribute name="po_quantity" time="1970-01-01T00:00:00Z">1</attribute>
      </attributes>
      <objects>
        <relationship object-id="R3" qualifier="Maverick buying"/>
      </objects>
    </object>
    <object id="PR1" type="Purchase Requisition">
      <attributes>
        <attribute name="pr_product" time="1970-01-01T00:00:00Z">Cows</attribute>
        <attribute name="pr_quantity" time="1970-01-01T00:00:00Z">500</attribute>
      </attributes>
      <objects>
        <relationship object-id="PO1" qualifier="PO from PR"/>
      </objects>
    </object>
  </objects>
  <events>
    <event id="e1" type="Create Purchase Requisition" time="2022-01-09T15:00:00Z">
      <attributes>
        <attribute name="pr_creator">Mike</attribute>
      </attributes>
      <objects>
        <relationship object-id="PR1" qualifier="Regular placement of PR"/>
      </objects>
    </event>
    <event id="e2" type="Approve Purchase Requisition" time="2022-01-09T16:30:00Z">
      <attributes>
        <attribute name="pr_approver">Tania</attribute>
      </attributes>
      <objects>
        <relationship object-id="PR1" qualifier="Regular approval of PR"/>
      </objects>
    </event>
    <event id="e3" type="Create Purchase Order" time="2022-01-10T09:15:00Z">
      <attributes>
        <attribute name="po_creator">Mike</attribute>
      </attributes>
      <objects>
        <relationship object-id="PR1" qualifier="Created order from PR"/>
        <relationship object-id="PO1" qualifier="Created order with identifier"/>
      </objects>
    </event>
    <event id="e4" type="Change PO Quantity" time="2022-01-13T12:00:00Z">
      <attributes>
        <attribute name="po_editor">Mike</attribute>
      </attributes>
      <objects>
        <relationship object-id="PO1" qualifier="Change of quantity"/>
      </objects>
    </event>
    <event id="e5" type="Insert Invoice" time="2022-01-14T12:00:00Z">
      <attributes>
        <attribute name="invoice_inserter">Luke</attribute>
      </attributes>
      <objects>
        <relationship object-id="PO1" qualifier="Invoice created starting from the PO"/>
        <relationship object-id="R1" qualifier="Invoice created with identifier"/>
      </objects>
    </event>
    <event id="e6" type="Insert Invoice" time="2022-01-16T11:00:00Z">
      <attributes>
        <attribute name="invoice_inserter">Luke</attribute>
      </attributes>
      <objects>
        <relationship object-id="PO1" qualifier="Invoice created starting from the PO"/>
        <relationship object-id="R2" qualifier="Invoice created with identifier"/>
      </objects>
    </event>
    <event id="e7" type="Insert Payment" time="2022-01-30T23:00:00Z">
      <attributes>
        <attribute name="payment_inserter">Robot</attribute>
      </attributes>
      <objects>
        <relationship object-id="R1" qualifier="Payment for the invoice"/>
        <relationship object-id="P1" qualifier="Payment inserted with identifier"/>
      </objects>
    </event>
    <event id="e8" type="Insert Payment" time="2022-01-31T22:00:00Z">
      <attributes>
        <attribute name="payment_inserter">Robot</attribute>
      </attributes>
      <objects>
        <relationship object-id="R2" qualifier="Payment for the invoice"/>
        <relationship object-id="P2" qualifier="Payment created with identifier"/>
      </objects>
    </event>
    <event id="e9" type="Insert Invoice" time="2022-02-02T09:00:00Z">
      <attributes>
        <attribute name="invoice_inserter">Mario</attribute>
      </attributes>
      <objects>
```

```
        <relationship object-id="R3" qualifier="Invoice created with identifier"/>
      </objects>
    </event>
    <event id="e10" type="Create Purchase Order" time="2022-02-02T17:00:00Z">
      <attributes>
        <attribute name="po_creator">Mario</attribute>
      </attributes>
      <objects>
        <relationship object-id="R3" qualifier="Purchase order created with maverick buying from"/>
        <relationship object-id="PO2" qualifier="Purchase order created with identifier"/>
      </objects>
    </event>
    <event id="e11" type="Set Payment Block" time="2022-02-03T07:30:00Z">
      <attributes>
        <attribute name="invoice_blocker">Mario</attribute>
      </attributes>
      <objects>
        <relationship object-id="R3" qualifier="Payment block due to unethical maverick buying"/>
      </objects>
    </event>
    <event id="e12" type="Remove Payment Block" time="2022-02-03T23:30:00Z">
      <attributes>
        <attribute name="invoice_block_rem">Mario</attribute>
      </attributes>
      <objects>
        <relationship object-id="R3" qualifier="Payment block removed ..."/>
      </objects>
    </event>
    <event id="e13" type="Insert Payment" time="2022-02-28T23:00:00Z">
      <attributes>
        <attribute name="payment_inserter">Robot</attribute>
      </attributes>
      <objects>
        <relationship object-id="R3" qualifier="Payment for the invoice"/>
        <relationship object-id="P3" qualifier="Payment inserted with identifier"/>
      </objects>
    </event>
  </events>
</log>
```

7.2 XML Schema Definition

A machine-readable XML Schema Definition (XSD) file is provided to check whether an example XML OCEL 2.0 is valid, see https://www.ocel-standard.org/2.0/ocel20-schema-xml.xsd Numerous tools are available to validate an XML file against an XSD file.

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="attribute">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute type="xs:string" name="name" use="required"/>
          <xs:attribute name="type" use="optional">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:enumeration value="string"/>
                <xs:enumeration value="time"/>
                <xs:enumeration value="integer"/>
                <xs:enumeration value="float"/>
                <xs:enumeration value="boolean"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
          <xs:attribute type="xs:dateTime" name="time" use="optional"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="attributes">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element ref="attribute" maxOccurs="unbounded" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="object-type">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="attributes"/>
      </xs:sequence>
      <xs:attribute type="xs:string" name="name" use="required"/>
    </xs:complexType>
    <xs:key name="objectTypeAttributeKey">
      <xs:selector xpath="attributes/attribute"/>
      <xs:field xpath="@name"/>
    </xs:key>
  </xs:element>
  <xs:element name="event-type">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="attributes"/>
      </xs:sequence>
      <xs:attribute type="xs:string" name="name" use="required"/>
    </xs:complexType>
    <xs:key name="eventTypeAttributeKey">
      <xs:selector xpath="attributes/attribute"/>
      <xs:field xpath="@name"/>
    </xs:key>
  </xs:element>
  <xs:element name="object">
    <xs:complexType mixed="true">
      <xs:sequence>
        <xs:element ref="attributes" minOccurs="0"/>
        <xs:element ref="objects" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute type="xs:string" name="object-id" use="optional"/>
      <xs:attribute type="xs:string" name="qualifier" use="optional"/>
      <xs:attribute type="xs:string" name="id" use="optional"/>
      <xs:attribute type="xs:string" name="type" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="objects">
    <xs:complexType>
      <xs:sequence>
```

```
      <xs:element ref="object" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="event">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="attributes"/>
      <xs:element ref="objects"/>
    </xs:sequence>
    <xs:attribute type="xs:string" name="id" use="required"/>
    <xs:attribute type="xs:string" name="type" use="required"/>
    <xs:attribute type="xs:dateTime" name="time" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="object-types">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="object-type" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="event-types">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="event-type" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="events">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="event" maxOccurs="unbounded" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="log">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="object-types"/>
      <xs:element ref="event-types"/>
      <xs:element ref="objects"/>
      <xs:element ref="events"/>
    </xs:sequence>
  </xs:complexType>
  <xs:key name="objectTypeKey">
    <xs:selector xpath="object-types/object-type"/>
    <xs:field xpath="@name"/>
  </xs:key>
  <xs:key name="objectKey">
    <xs:selector xpath="objects/object"/>
    <xs:field xpath="@id"/>
  </xs:key>
  <xs:key name="eventTypeKey">
    <xs:selector xpath="event-types/event-type"/>
    <xs:field xpath="@name"/>
  </xs:key>
  <xs:key name="eventKey">
```

```xml
      <xs:selector xpath="events/event"/>
      <xs:field xpath="@id"/>
    </xs:key>
    <xs:keyref name="objectTypeRef" refer="objectTypeKey">
      <xs:selector xpath="objects/object"/>
      <xs:field xpath="@type"/>
    </xs:keyref>
    <xs:keyref name="eventTypeRef" refer="eventTypeKey">
      <xs:selector xpath="events/event"/>
      <xs:field xpath="@type"/>
    </xs:keyref>
    <xs:keyref name="objectObjectRef" refer="objectKey">
      <xs:selector xpath="objects/object/objects/object"/>
      <xs:field xpath="@object-id"/>
    </xs:keyref>
    <xs:keyref name="eventObjectRef" refer="objectKey">
      <xs:selector xpath="events/event/objects/object"/>
      <xs:field xpath="@object-id"/>
    </xs:keyref>
    <xs:keyref name="objectAttributeRef" refer="objectTypeAttributeKey">
      <xs:selector xpath="objects/object/attributes/attribute"/>
      <xs:field xpath="@name"/>
    </xs:keyref>
    <xs:keyref name="eventAttributeRef" refer="eventTypeAttributeKey">
      <xs:selector xpath="events/event/attributes/attribute"/>
      <xs:field xpath="@name"/>
    </xs:keyref>
  </xs:element>
</xs:schema>
```

8 JSON Format

The JSON format provides a lightweight structure for web-native process mining applications. It is conceptually similar to the XML format with its top-level arrays `events`, `eventTypes`, `objects`, and `objectTypes`.

In the following, we describe these four top-level properties in detail.

- The top-level `event` array contains event objects with the properties `id`, `type` (referencing the name of an event type), and `time` (ISO format). An event's `attributes` are structured into an array of attribute objects with `name` and `value` properties. The event's event-to-object relationships are listed in the *relationships* array with `objectId` and `qualifier`.
- The top-level `eventTypes` array contains event type objects with a `name` and a list of attributes with `name` and `value` properties. Valid types are **string**, **time**, **integer**, **float**, and **boolean**.
- The top-level `object` array contains a list of objects as JSON object, with properties `id` and `type` (referencing the name of an object type). The attributes property contains an array of attributes with the properties `name`, `time` (ISO format), and `value`.
- Finally, the top-level objectTypes array contains object type description objects with a `name` and a list of attributes with `name` and `value` properties. Valid types are **string**, **time**, **integer**, **float**, and **boolean**.

8.1 JSON Example

As an example, we show the running example formatted as a JSON document.

```json
{
  "objectTypes": [
    {
      "name": "Invoice",
      "attributes": [
        {
          "name": "is_blocked",
          "type": "string"
        }
      ]
    },
    {
      "name": "Payment",
      "attributes": []
    },
    {
      "name": "Purchase Order",
      "attributes": [
        {
          "name": "po_product",
          "type": "string"
        },
        {
          "name": "po_quantity",
          "type": "string"
        }
      ]
    },
    {
      "name": "Purchase Requisition",
      "attributes": [
        {
          "name": "pr_product",
          "type": "string"
        },
        {
          "name": "pr_quantity",
          "type": "string"
        }
      ]
    }
  ],
  "eventTypes": [
    {
      "name": "Approve Purchase Requisition",
      "attributes": [
        {
          "name": "pr_approver",
          "type": "string"
        }
      ]
    },
    {
      "name": "Change PO Quantity",
      "attributes": [
        {
          "name": "po_editor",
          "type": "string"
        }
      ]
    },
    {
      "name": "Create Purchase Order",
      "attributes": [
        {
          "name": "po_creator",
          "type": "string"
        }
      ]
    },
    {
      "name": "Create Purchase Requisition",
      "attributes": [
        {
          "name": "pr_creator",
          "type": "string"
        }
      ]
    },
    {
      "name": "Insert Invoice",
      "attributes": [
        {
          "name": "invoice_inserter",
          "type": "string"
        }
      ]
    },
    {
      "name": "Insert Payment",
      "attributes": [
        {
          "name": "payment_inserter",
          "type": "string"
        }
      ]
    },
    {
      "name": "Remove Payment Block",
      "attributes": [
        {
          "name": "invoice_block_rem",
          "type": "string"
        }
      ]
    },
    {
      "name": "Set Payment Block",
      "attributes": [
        {
          "name": "invoice_blocker",
          "type": "string"
        }
      ]
    }
  ],
  "objects": [
    {
      "id": "R1",
      "type": "Invoice",
      "attributes": [
        {
          "name": "is_blocked",
          "time": "1970-01-01T00:00:00Z",
          "value": "No"
        }
      ],
      "relationships": [
        {
          "objectId": "P1",
          "qualifier": "Payment from invoice"
        }
      ]
    },
    {
      "id": "R2",
      "type": "Invoice",
      "attributes": [
        {
          "name": "is_blocked",
          "time": "1970-01-01T00:00:00Z",
          "value": "No"
        }
      ],
      "relationships": [
        {
          "objectId": "P2",
          "qualifier": "Payment from invoice"
        }
      ]
    },
    {
      "id": "R3",
      "type": "Invoice",
      "attributes": [
        {
          "name": "is_blocked",
          "time": "1970-01-01T00:00:00Z",
          "value": "No"
        },
        {
          "name": "is_blocked",
          "time": "2022-02-03T07:30:00Z",
          "value": "Yes"
        },
        {
          "name": "is_blocked",
          "time": "2022-02-03T23:30:00Z",
          "value": "No"
        }
      ],
      "relationships": [
        {
          "objectId": "P3",
          "qualifier": "Payment from invoice"
        }
      ]
    },
    {
      "id": "P1",
      "type": "Payment"
    },
    {
      "id": "P2",
      "type": "Payment"
    },
    {
      "id": "P3",
      "type": "Payment"
    },
    {
      "id": "PO1",
      "type": "Purchase Order",
      "attributes": [
        {
          "name": "po_product",
          "time": "1970-01-01T00:00:00Z",
          "value": "Cows"
      },
      {
        "name": "po_quantity",
        "time": "1970-01-01T00:00:00Z",
        "value": "500"
      },
      {
        "name": "po_quantity",
        "time": "2022-01-13T12:00:00Z",
        "value": "600"
      }
    ],
    "relationships": [
      {
        "objectId": "R1",
        "qualifier": "Invoice from PO"
      },
      {
        "objectId": "R2",
        "qualifier": "Invoice from PO"
      }
    ]
  },
  {
    "id": "PO2",
    "type": "Purchase Order",
    "attributes": [
      {
        "name": "po_product",
        "time": "1970-01-01T00:00:00Z",
        "value": "Notebooks"
      },
      {
        "name": "po_quantity",
        "time": "1970-01-01T00:00:00Z",
        "value": "1"
      }
    ],
    "relationships": [
      {
        "objectId": "R3",
        "qualifier": "Maverick buying"
      }
    ]
  },
  {
    "id": "PR1",
    "type": "Purchase Requisition",
    "attributes": [
      {
        "name": "pr_product",
        "time": "1970-01-01T00:00:00Z",
        "value": "Cows"
      },
      {
        "name": "pr_quantity",
        "time": "1970-01-01T00:00:00Z",
        "value": "500"
      }
    ],
    "relationships": [
      {
        "objectId": "PO1",
        "qualifier": "PO from PR"
      }
    ]
  }
],
"events": [
  {
    "id": "e1",
    "type": "Create Purchase Requisition",
    "time": "2022-01-09T15:00:00Z",
    "attributes": [
      {
        "name": "pr_creator",
        "value": "Mike"
      }
    ],
    "relationships": [
      {
        "objectId": "PR1",
        "qualifier": "Regular placement of PR"
      }
    ]
  },
  {
    "id": "e2",
    "type": "Approve Purchase Requisition",
    "time": "2022-01-09T16:30:00Z",
    "attributes": [
      {
        "name": "pr_approver",
        "value": "Tania"
      }
    ],
    "relationships": [
      {
        "objectId": "PR1",
        "qualifier": "Regular approval of PR"
      }
    ]
  },
  {
    "id": "e3",
    "type": "Create Purchase Order",
    "time": "2022-01-10T09:15:00Z",
    "attributes": [
      {
        "name": "po_creator",
        "value": "Mike"
      }
    ],
    "relationships": [
      {
        "objectId": "PR1",
        "qualifier": "Created order from PR"
      },
      {
        "objectId": "PO1",
        "qualifier": "Created order with identifier"
      }
    ]
  },
  {
    "id": "e4",
    "type": "Change PO Quantity",
    "time": "2022-01-13T12:00:00Z",
    "attributes": [
      {
        "name": "po_editor",
        "value": "Mike"
      }
    ],
    "relationships": [
      {
        "objectId": "PO1",
        "qualifier": "Change of quantity"
      }
    ]
  },
  {
    "id": "e5",
    "type": "Insert Invoice",
    "time": "2022-01-14T12:00:00Z",
    "attributes": [
      {
        "name": "invoice_inserter",
        "value": "Luke"
      }
    ],
    "relationships": [
      {
        "objectId": "PO1",
        "qualifier": "Invoice created starting from the PO"
      },
      {
        "objectId": "R1",
        "qualifier": "Invoice created with identifier"
      }
    ]
  },
  {
    "id": "e6",
    "type": "Insert Invoice",
    "time": "2022-01-16T11:00:00Z",
    "attributes": [
      {
        "name": "invoice_inserter",
        "value": "Luke"
      }
    ],
    "relationships": [
      {
        "objectId": "PO1",
        "qualifier": "Invoice created starting from the PO"
      },
      {
        "objectId": "R2",
        "qualifier": "Invoice created with identifier"
      }
    ]
  },
  {
    "id": "e7",
    "type": "Insert Payment",
    "time": "2022-01-30T23:00:00Z",
    "attributes": [
      {
        "name": "payment_inserter",
        "value": "Robot"
      }
    ],
    "relationships": [
      {
        "objectId": "R1",
        "qualifier": "Payment for the invoice"
      },
      {
        "objectId": "P1",
        "qualifier": "Payment inserted with identifier"
      }
    ]
  },
  {
    "id": "e8",
    "type": "Insert Payment",
    "time": "2022-01-31T22:00:00Z",
    "attributes": [
      {
        "name": "payment_inserter",
        "value": "Robot"
      }
    ],
    "relationships": [
      {
        "objectId": "R2",
        "qualifier": "Payment for the invoice"
      },
      {
        "objectId": "P2",
        "qualifier": "Payment created with identifier"
      }
    ]
  },
  {
    "id": "e9",
    "type": "Insert Invoice",
    "time": "2022-02-02T09:00:00Z",
    "attributes": [
      {
        "name": "invoice_inserter",
          "value": "Mario"
        }
      ],
      "relationships": [
        {
          "objectId": "R3",
          "qualifier": "Invoice created with identifier"
        }
      ]
    },
    {
      "id": "e10",
      "type": "Create Purchase Order",
      "time": "2022-02-02T17:00:00Z",
      "attributes": [
        {
          "name": "po_creator",
          "value": "Mario"
        }
      ],
      "relationships": [
        {
          "objectId": "R3",
          "qualifier": "Purchase order created with maverick buying from"
        },
        {
          "objectId": "PO2",
          "qualifier": "Purchase order created with identifier"
        }
      ]
    },
    {
      "id": "e11",
      "type": "Set Payment Block",
      "time": "2022-02-03T07:30:00Z",
      "attributes": [
        {
          "name": "invoice_blocker",
          "value": "Mario"
        }
      ],
      "relationships": [
        {
          "objectId": "R3",
          "qualifier": "Payment block due to unethical maverick buying"
        }
      ]
    },
    {
      "id": "e12",
      "type": "Remove Payment Block",
      "time": "2022-02-03T23:30:00Z",
      "attributes": [
        {
          "name": "invoice_block_rem",
          "value": "Mario"
        }
      ],
    "relationships": [
      {
        "objectId": "R3",
        "qualifier": "Payment block removed ..."
      }
    ]
  },
  {
    "id": "e13",
    "type": "Insert Payment",
    "time": "2022-02-28T23:00:00Z",
    "attributes": [
      {
        "name": "payment_inserter",
        "value": "Robot"
      }
    ],
    "relationships": [
      {
        "objectId": "R3",
        "qualifier": "Payment for the invoice"
      },
      {
        "objectId": "P3",
        "qualifier": "Payment inserted with identifier"
      }
    ]
  }
 ]
}
```

8.2 JSON Schema Definition

We defined a validation schema for the OCEL 2.0 JSON specification. The schema is reported in the following snippet and can be downloaded from https://www.ocel-standard.org/2.0/ocel20-schema-json.json.

```
{
    "$schema": "http://json-schema.org/draft-07/schema#",
    "type": "object",
    "properties": {
        "eventTypes": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "name": { "type": "string" },
                    "attributes": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "name": { "type": "string" },
                                "type": { "type": "string" }
                            },
                            "required": ["name", "type"]
                        }
                    }
                },
                "required": ["name", "attributes"]
            }
        },
        "objectTypes": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "name": { "type": "string" },
                    "attributes": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "name": { "type": "string" },
                                "type": { "type": "string" }
                            },
                            "required": ["name", "type"]
                        }
                    }
                },
                "required": ["name", "attributes"]
            }
        },
        "events": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "id": { "type": "string" },
                    "type": { "type": "string" },
                    "time": { "type": "string", "format": "date-time" },
                    "attributes": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "name": { "type": "string" },
                                "value": { "type": "string" }
                            },
                            "required": ["name", "value"]
                        }
                    },
                    "relationships": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "objectId": { "type": "string" },
                                "qualifier": { "type": "string" }
                            },
                            "required": ["objectId", "qualifier"]
                        }
                    }
                },
                "required": ["id", "type", "time"]
            }
        },
        "objects": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "id": { "type": "string" },
                    "type": { "type": "string" },
                    "relationships": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "objectId": { "type": "string" },
                                "qualifier": { "type": "string" }
                            },
                            "required": ["objectId", "qualifier"]
                        }
                    },
                    "attributes": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "name": { "type": "string" },
                                "value": { "type": "string" },
                                "time": { "type": "string", "format": "date-time" }
                            },
                            "required": ["name", "value", "time"]
                        }
                    }
                },
                "required": ["id", "type"]
            }
        }
    },
    "required": ["eventTypes", "objectTypes", "events", "objects"]
}
```

9 Conclusion

This document provides a comprehensive introduction to the OCEL 2.0 standard. Given the increasing importance of Object-Centric Process Mining (OCPM), it is important to be able to standardize Object-Centric Event Data (OCED). OCEL 2.0 aims to provide a middle ground between simplicity and expressiveness, building upon experiences with OCEL 1.0 over the past three years. We first provided a contextual understanding of the object-centric process mining landscape and then discussed the motivation and the necessities that led to the creation of the OCEL 2.0 standard.

Why is this relevant?

- Using OCEL 2.0, it is possible to create a system-agnostic, single source of truth. Event data should capture real business-relevant events without being limited by a single-case notion.

- We no longer need to create a new event log for each process (or view on a selected process). Using traditional event logs, there may be overlapping logs that refer to products, suppliers, etc. This leads to duplication and inconsistencies. Using OCEL 2.0, views can be created on demand without going back to the source systems.

- OCEL 2.0 and the OCPM techniques that build upon it allow us to stay closer to reality, also allowing organizations to uncover problems that live at the intersection points of processes and organizational units.

Through a detailed presentation of formal definitions, we built the mathematical foundation that equips readers to effectively utilize and understand the OCEL 2.0 standard in a scientific context. Our illustrative running example served to bridge the gap between abstract theory and practical application, enabling readers to fully grasp how the principles of OCEL 2.0 come into play.

The detailed overview of the relational SQLite, XML, and JSON implementations demonstrated the versatility and compatibility of the standard across multiple technological contexts. Together, these make OCEL 2.0 an accessible and practical choice for both academics and industry practitioners.

Compared to OCEL 1.0, OCEL 2.0 allows for changes in objects, is able to relate objects directly, and can qualify relationships between objects and events. We hope that this will fuel new OCPM techniques using these extensions. The relational SQLite implementation also shows that this standard goes beyond file formats like XML, JSON, and Excel. Because the reference storage format for the XES standard was XML, people often misunderstood the XES standard. The XES metamodel can be implemented in many different ways. However, some vendors used the bulkiness XML as an excuse not to support XES, thus blocking any form of standardization. However, the critical point is the standardization and unification of concepts (not the serialization of data using a specific syntax). Therefore, we encourage researchers and software companies to come up with new storage formats implementing the OCEL 2.0 metamodel and provide conversations. The SQLite, XML, and JSON formats are just examples, and we provide tools to convert any one of them into the two other formats. However, we encourage developers to create novel, highly scalable formats. This is one of the reasons why we kept OCEL 2.0 as simple and as concrete as possible.

We also hope that OCEL 2.0 will also be the basis for creating *standard object and event types* for different application domains. Organizations struggle to use ontologies and related technologies because the added value of extensively modeling data is not so clear. OCPM can address this problem. Event data in OCEL 2.0 format enables process discovery, conformance checking, performance analysis, and operational support without the need to process the data further. However, any organization has standard processes like Order-to-Cash (O2C) and Procure-to-Pay (P2P) that talk about suppliers, customers, orders, items, shipments, etc. There is no need to reinvent the wheel. Also, one would like to keep these things system-agnostic. Definitions of object types and event types and their attributes need to be standardized. It is possible to define taxonomies of object types and event types using inheritance notions [2]. This creates possibilities for both generative and discriminative Artificial Intelligence (AI). Therefore, researchers and solution providers should focus on creating standard object and event types and the corresponding normative object-centric process models. This will prevent organizations from starting from scratch when applying process mining and AI for the first time.

Lastly, it is essential to note that the rapidly evolving field of object-centric process mining continues to present new challenges and opportunities. As such, the OCEL 2.0 standard, despite its substantial contribution, should be seen as a stepping stone in this exciting journey rather than a final destination. Furthermore, this standard is intended to help pave the path for the development of process mining techniques supporting the journey to more sustainable operational practices. We encourage further research and development efforts to build on this foundation, with the aim of continuously advancing the field to new heights of innovation and practical value.

You can find further details of OCEL 2.0, example event logs, and tool support, on our homepage:

`https://www.ocel-standard.org`

Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC-2023 Internet of Production - 390621612. We also thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

References

[1] W.M.P. van der Aalst. *Process Mining: Data Science in Action.* Springer-Verlag, Berlin, 2016.

[2] W.M.P van der Aalst. Object-Centric Process Mining: Unraveling the Fabric of Real Processes. *Mathematics*, 11(12):2691, 2023.

[3] W.M.P. van der Aalst and A. Berti. Discovering Object-Centric Petri Nets. *Fundamenta Informaticae*, 175(1-4):1–40, 2020.

[4] W.M.P. van der Aalst and J. Carmona, editors. *Process Mining Handbook*, volume 448 of *Lecture Notes in Business Information Processing.* Springer-Verlag, Berlin, 2022.

[5] G. Acampora, A. Vitiello, B. Di Stefano, W. van der Aalst, C. Günther, and E. Verbeek. IEEE 1849: The XES Standard - The Second IEEE Standard Sponsored by IEEE Computational Intelligence Society. *IEEE Computational Intelligence Magazine*, 12(2):4–8, 2017.

[6] J.N. Adams, G. Park, S. Levich, D. Schuster, and W.M.P. van der Aalst. A Framework for Extracting and Encoding Features from Object-Centric Event Data. In J. Troya, B.Medjahed, M.Piattini, L. Yao, P. Fernández, and A. Ruiz-Cortés, editors, *International Conference on Service-Oriented Computing (ICSOC 2022)*, volume 13740 of *Lecture Notes in Computer Science*, pages 36–53. Springer-Verlag, Berlin, 2022.

[7] A.F. Ghahfarokhi, G. Park, A. Berti, and W.M.P. van der Aalst. OCEL Standard. www.ocel-standard.org, 2020.

[8] A.F. Ghahfarokhi, G. Park, A. Berti, and W.M.P. van der Aalst. OCEL: A Standard for Object-Centric Event Logs. In L. Bellatreche, M. Dumas, and P. Karras, editors, *New Trends in Database and Information Systems (Short Papers ADBIS 2021)*, volume 1450 of *Communications in Computer and Information Science*, pages 169–175. Springer-Verlag, Berlin, 2021.

[9] IEEE. IEEE Standard for eXtensible Event Stream (XES) for Achieving Interoperability in Event Logs and Event Streams. *IEEE Std 1849-2023 (Revision of IEEE Std 1849-2016)*, pages 1–55, 2023. 10.1109/IEEESTD.2023.10267858.

[10] IEEE Task Force on Process Mining. XES Standard Definition. www.xes-standard.org, 2010.

[11] M. Kerremans, K. Iijima, A. Sachelarescu, N. Duffy, and D. Sugden. Magic Quadrant for Process Mining Tools, Gartner Research Note GG00774746. www.gartner.com, 2023.

[12] M.T. Wynn, J. Lebherz, W.M.P. van der Aalst, R. Accorsi, C. Di Ciccio, L. Jayarathna, and H.M.W. Verbeek. Rethinking the Input for Process Mining: Insights from the XES Survey and Workshop. In J. Munoz-Gama and X. Lu, editors, *Process Mining Workshops of the International Conference on Process Mining (Revised Selected Papers)*, volume 433 of *Lecture Notes in Business Information Processing*, pages 3–16. Springer-Verlag, Berlin, 2021.