OBJECT-CENTRIC EVENT DATA (OCED)

Call for Action: Reference Implementations

TFPM OCED working group – 2023/03

This Call for Action aims at moving ahead with a common data exchange standard for process mining in industry and academia to fuel and accelerate further growth. Contextualizing the rationale first, we proceed to detail the Call for Action itself. Followed by a recap of developments to date, we move on to describe the OCED meta model in both its base and full form, as well as known limitations which have been discussed within the IEEE Task Force on Process Mining – OCED working group.

Rationale behind OCED

In the past decade, Process Mining has not only seen tremendous growth in the academic arena, but also started to establish itself as one of the predominant approaches to improving processes for larger companies of virtually any industry. Process Mining and related services have become a sizeable business - for software vendors, professional service firms, and commercial end user alike.

Such a trajectory naturally spurs substantial investment and advancement; however, it is also typical that – in an attempt to safeguard intellectual property – many new features, products and services are being shielded off from usage by other players in the ecosystem. Especially when this affects areas that are of concern to all market participants, such silos impede competition, innovation and the pace of further development. In many industries that are heavily reliant on the exchange of something, standardizing terms and conditions of such exchange led to a great leap forward for the entire ecosystem – think containers for global trade, internet protocol for global communication.

Process Mining itself is heavily reliant on the exchange of data, which typically originates from systems that were not designed around this use case and hence requires substantial transformation. Market participants have created different approaches to reduce the effort required for data transformation, but so far no data exchange format has seen enough adoption to be nominated as a de-facto standard. The relevance and magnitude of this bottleneck as one of the predominant effort drivers in process mining projects has been reconfirmed by the Task Force on Process Mining (TFPM)[1].

With the IEEE eXtensible Event Stream (XES) initiated in 2010, academia has established a data exchange format, which fueled tremendous growth in process mining research. Standardizing how academia captures, transfers, loads and interprets event data continues to pay dividends. However, process mining adoption by business entities and the academic progress in the past decade changed the requirements towards an up-to-date standard substantially. The IEEE TFPM hence proposes and invites to co-design the XES successor **Object-Centric Event Data (OCED)**:

- creating more competition and innovation by lowering the market entry barrier
- allowing commercial end users and professional service providers to focus their effort on process improvement and other value creating tasks rather than data transformation
- allowing software vendors to focus on value-adding differentiators, rather than creating the n^{th} data transformation engine and corresponding development environment
- improving security of investment for commercial end users
- creating a market for source system specific adapter modules translating data into OCED

[1] Rethinking the Input for Process Mining: Insights from the XES Survey and Workshop https://www.tf-pm.org/upload/1637654003748.pdf

Call for Action: Reference Implementations

We invite anyone in the process mining community (academia and Industry) to work on reference implementations of the proposed OCED meta-model.

We intend to move forward with several reference implementations which will be used to gather experiences among the community in terms of (1) the suitability of the proposed OCED meta-model, (2) the usefulness of the proposed constructs, and (3) the need for additional constructs or extensions that are required to be incorporated in the meta-model and consequentially into the derived reference implementations.

The planned timeline for reference implementations is as follows:

- Call for Action: March 2023
- 1st round of reference implementations: March - June 2023
- Feedback to the OCED working group: July 2023
- Update to the reference implantations: July - Sep 2023
- Presentation at the OCED symposium: Oct 2023

The scope of each reference implementation is to show (1) how the OCED meta-model concepts are mapped to/supported by the individual implementation and (2) how imports and/or exports of actual data works in practice. Although the maximum value for the community is created by openly sharing the inner workings of individual reference implementations (i.e open source), we appreciate that commercial entites need to protect their IP. We therefore accept submissions showcasing import and export functionality while preserving the semantical constructs of the OCED meta-model without disclosing the "inner storage engine or format". It is also worth noting that reference implementations might be of serialized nature (e.g. XML or JSON) or "live" models (e.g. relational database or graph structures).

If a research team, vendor or other entity is interested in participating in this initiative, please reach out to us via mailto:oced@tf-pm.org. We will have detailed discussions with interested parties regarding the requirements. You are also invited to share your synthetic/real-world datasets and proposed extensions to the OCED meta-model with the working group. We will offer one-to-one monthly check-ins throughout the implementation period.

The Path to OCED

The requirements for OCED were gathered through an online survey with 289 participants and a XES 2.0 workshop co-located with the 3rd International Conference on Process Mining (Eindhoven, 2021). This led to the following three observations: (1) the single-case-notion is very limiting leading to a disconnect between reality and the represented events, (2) the XES standard is too complex and many of its extensions are rarely used, and (3) XES is associated with a particular XML storage format making it impractical for many real-life use cases. Learning from the problems associated with XES, OCED needs to be object-centric, as simple as possible, and have a meta model decoupled from a particular storage format.

A core team of eight experts from academia and industry was formed to develop an initial proposal for the OCED Meta Model (OCED-MM). Diverse views and opinions were solicited and discussed among the core team. The resulting meta model captures the concepts that have a majority vote. It was circulated for feedback in two runs: first run on August, 2nd 2022 to respondents of the 2021 survey, registered attendees of the 2021 XES Workshop as well as the TFPM steering committee and advisory board, second run on September, 9th 2022 to all subscribers of the TFPM newsletter. As part

of the 4th International Conference on Process Mining (Bolzano, 2022) an XES Symposium[2] was held, in which the OCED-MM was presented and discussed. While many interesting thoughts and ideas have been brought to the attention of the OCED working group, the core concepts of the meta model remained in consensus. Hence the hereafter presented OCED-MM is moving ahead, targeting multiple reference implementations with different languages to generate feedback and improve the standard before an official certification via IEEE.

The OCED Meta Model – Base Model

The OCED-MM is comprised of a Base Model, as depicted in *Figure 1*, and two additional concepts captured in the full OCED-MM, depicted in Figure 5. Any reference implementation shall capture the concepts in the Base Model, while the additions in the Full Model are a recommended extension. Within the OCED working group the value of said additions is undisputed, however, reservations regarding the technical feasibility and usability of resulting elevated complexity should be tested through actual refernce implementations.

In order to present the concepts in manageable chunks, the Base Model in Figure 1 is structured into three groups highlighted in blue. **Group A** describes events, associated attributes and the time construct, which can be closely related to the XES standard and many non-object centric process mining solutions in the market. **Group B** describes the objects, associated attributes, as well as object relations. Inclusion of these concepts directy stems from real-world requirements as described in the *Path to OCED* above. **Group C** eventually connects the two concept clusters, Group A and B.

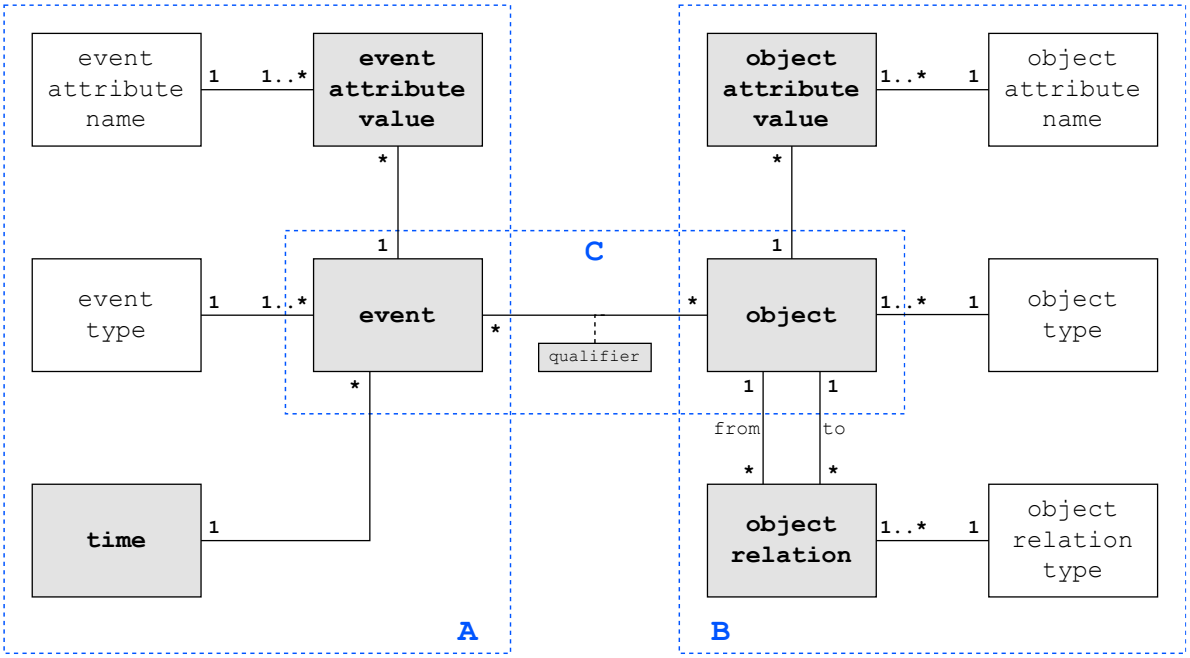

Figure 1 - Object-Centric Event Data Base Meta Model

Starting with **Group A**, the following event-centric concepts are captured:

1. **`event`** | An `event` is atomic, meaning it refers to something taking place at exactly one point in time rather than having a duration. Every `event` has exactly one `event type` (e.g. *[purchase order approved]*; captured as *string*), while each defined `event type` relates to at least one `event` (e.g. *[purchase order ID#298374 approved]*). Such `event types` can also be referred to as the names of the `events`.

[2] https://icpmconference.org/2022/program/xes-symposium/

2. **`time` |** A `time` captures both a timestamp conforming to ISO 8601-1:2019 and its resolution (ref. to the precision in which the timestamp was recorded). At minimum, the following precisions are to be differentiated: date, hour, minute, second, millisecond. If the time zone is omitted, all timestamps are treated as UTC. While the meta model does not prescribe a method to represent the timestamp-resolution-pair, the ISO standard 8601-2:2019 proposes uncertainty classifiers as a way to store both components in one *string*.

3. **`event attribute value` |** Each `event` can have an arbitrary number of `event attribute values` (e.g. *[transaction currency] = [USD]*), while each `event attribute value` must be related to exactly one `event`. Every `event attribute value` has exactly one `event attribute name` (e.g. *[user type] = [manual] or [RPA] or [automated]*), while each defined `event attribute name` relates to at least one `event attribute value`. Some information is typically represented with value-unit-pairs (e.g. price & currency) describing parts of the same logical information. In such cases it is good practice to indicate their relation by choosing the unit's `event attribute name` as the value's `event attribute name` suffixed with "_unit" (e.g. *[price] = [48.76]* & *[price_unit] = [USD]*). Please note that this is not prescribed by the meta model.

In **Group B**, the following object-centric concepts are captured:

4. **`object` |** An `object` either represents something tangible or abstract. Examples of tangible `objects` are persons, locations, machines, documents, document line items. Examples of abstract `objects` are legal entities, organizational constructs, electronic documents. To represent `objects` in accordance with the model, it not required to classify them into either of the two. Every `object` has exactly one `object type` (e.g. *[sales order]*; captured as *string*), while each defined `object type` relates to at least one `object` (e.g. *[sales order ID#12345]*).

5. **`object attribute value` |** An `object attribute value` represents a singular piece of information about an `object` and is captured as *string*, *boolean*, *integer*, *real*, *date*, *time* or *timestamp* (the latter three in accordance with ISO 8601-1:2019). Nested `object attribute values` are not supported. `Objects` can have an arbitrary number of `object attribute values`, while each `object attribute value` is related to exactly one `object`. Every `object attribute value` has exactly one `object attribute name` (e.g. *[colour variant]*; captured as *string*), while each defined `object attribute name` relates to at least one `object attribute value` (e.g. *[colour variant] = [blue]*). Some information is typically represented with value-unit-pairs describing parts of the same logical information. In such cases it is good practice to indicate their relation by choosing the unit's `object attribute name` as the value's `object attribute name` suffixed with "_unit" (e.g. *[weight] = [1897]* & *[weight_unit] = [kg]*). Please note that this is not prescribed by the meta model.

6. **`object relation` | An** `object relation` represents a link between a pair of `objects`. `Objects` can have an arbitrary number of `object relations`, while each `object relation` is referring to exactly two `objects`, one reflecting the relation's origin (from) and one pointing to the relation's target (to). Every `object relation` has exactly one `object relation type` (e.g. *purchase order line items* being related to their parent *purchase order*; their `object relation type` is *[child of]*; captured as *string*), while each defined `object relation type` is associated with at least one `object` (e.g. *[purchase order item ID#1423]* → `CHILD_OF` → *[purchase order ID#9877]*). Only a minimum set of `object relation types` are predefined (`CHILD_OF`, `PARENT_OF`), but additional `object relation types` can be introduced as part of the data capture and used to reflect semantics of the relation. The predefined object relation types capture implicit semantics to minimize the need for extensive data storage and transmission:

    a. `CHILD_OF` (opposite applies to `PARENT_OF`) defines an `object` relationship, in which an arbitrary number of child `objects` are related to exactly one parent `object` (N:A). This results in the following implicit semantics:

i. When a child `object` gets deleted, its `object relation` to the parent `object` is deleted in unison, however, the parent `object` itself is not affected.
ii. When a child `object` gets created, its `object relation` to the parent `object` is created in unison, however, the parent `object` itself is not affected.
iii. When a parent `object` gets deleted, all its child `objects` and their `object relations` to the parent are deleted in unison.
iv. When a parent `object` gets created, its initial children `objects` (i.e. the ones without a dedicated related creation event) are created in (e.g. when a purchase order is created with ten line items, solely the purchase order's link to the `event` needs to be captured).

b. Any `object relation type`, if not created explicitly (i.e. after both `objects` are in existence), is created implicitly with the `CREATE` of the second `object`.

c. Any `object relation type`, if not deleted explicitly (i.e. while both `objects` are remaining existence), is deleted implicitly with the `DELETE` of the first `object`.

In **Group C**, the two clusters are connected:

7. **`event` ↔ `object` |** An `event` and an `objects` can be related in a qualified (i.e. association class) manner, meaning their type of relationship is denoted. A minimum set of qualifiers is predefined (`CREATE`, `MODIFY` and `DELETE`), but additional qualifiers can be introduced as part of the data capture and used to reflect semantics of the relationship. Each `object` can be involved in an arbitrary number of `events`, while each `event` can be related to an arbitrary number of `objects`. This means, there can be `events` without `objects` and vice-versa.

The OCED Meta Model – Base Model Examples

Before detailing the additional concepts of the full OCED meta model and without going into any specific implementation format yet, the following shall exemplify how the meta model can be applied. For this, real-world scenarios in the context of a purchase-to-pay process have been selected.

I) PO creation | A purchase order document is created. In this context we focus on the PO header information only. Hence a PO `object` is created alongside a number of `object attributes`. For simplicity we solely present the PO's release status, which is created as non-released. The respective values are marked with red font in Figure 2, sparing unused parts of the meta model.

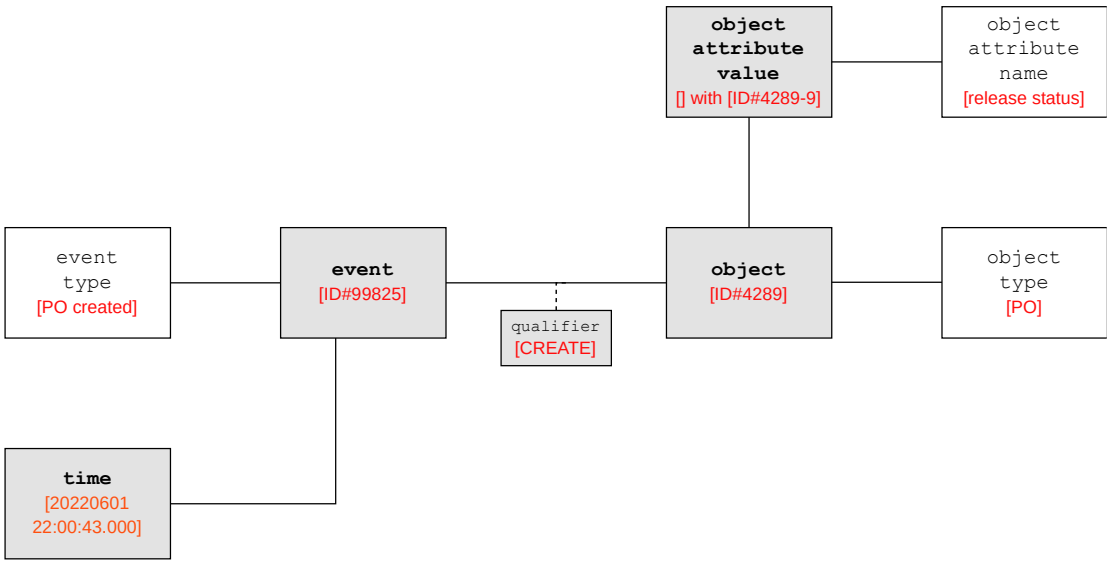

Figure 2: Example Scenario A - PO created

II) **PO release |** The purchase order is released. Data-wise this is reflected in an updated release status. The `object` itself does not change, but its `object attribute value` does. Please refer to Figure 3 for details.

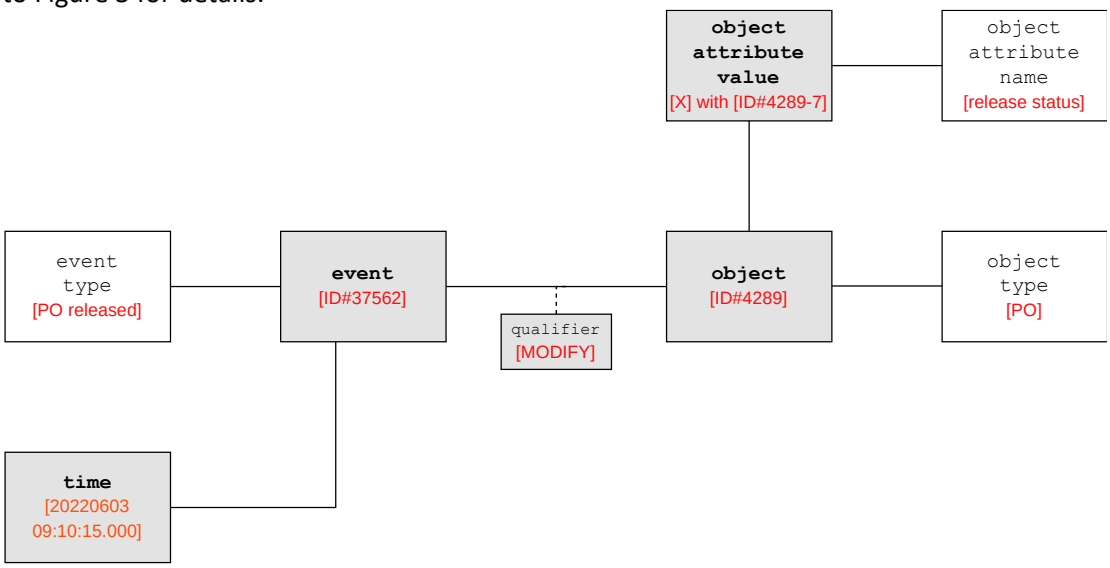

Figure 3: Example Scenario B – PO released

III) **Invoice receipt |** An invoice is receipt in relation to the purchase order created in scenario I. The invoice, one of its invoice line items, as well as their `object relation` get recorded. Another `object relation` is recorded for the link to the PO. Further PO details (i.e. `object attribute values`) of the purchase order are omitted in Figure 4 since they do not change. As per standard practice this invoice gets recorded with an active payment block. Since this `object attribute value` is associated to the invoice (header), it implicitly applies to its children, i.e. invoice line items, as well.

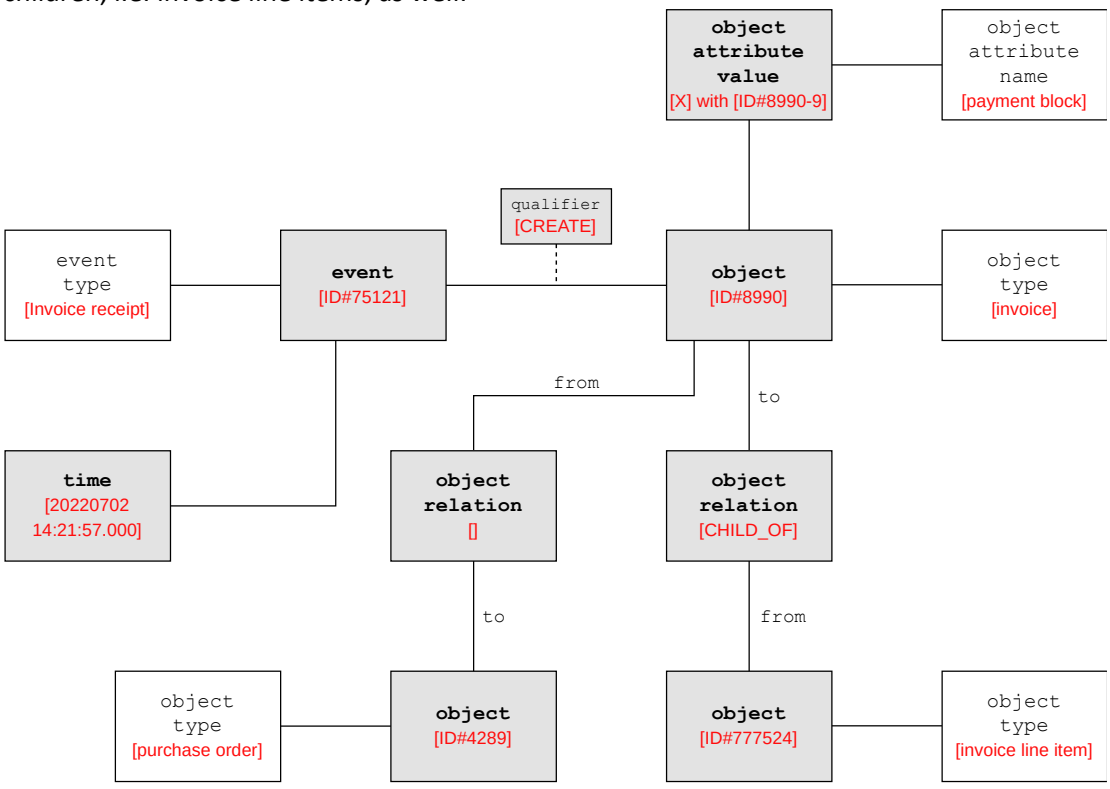

Figure 4: Example Scenario C - Invoice receipt

The OCED Meta Model – Full Model

In the following, the two additional concepts of the Full OCED Meta Model are explained.

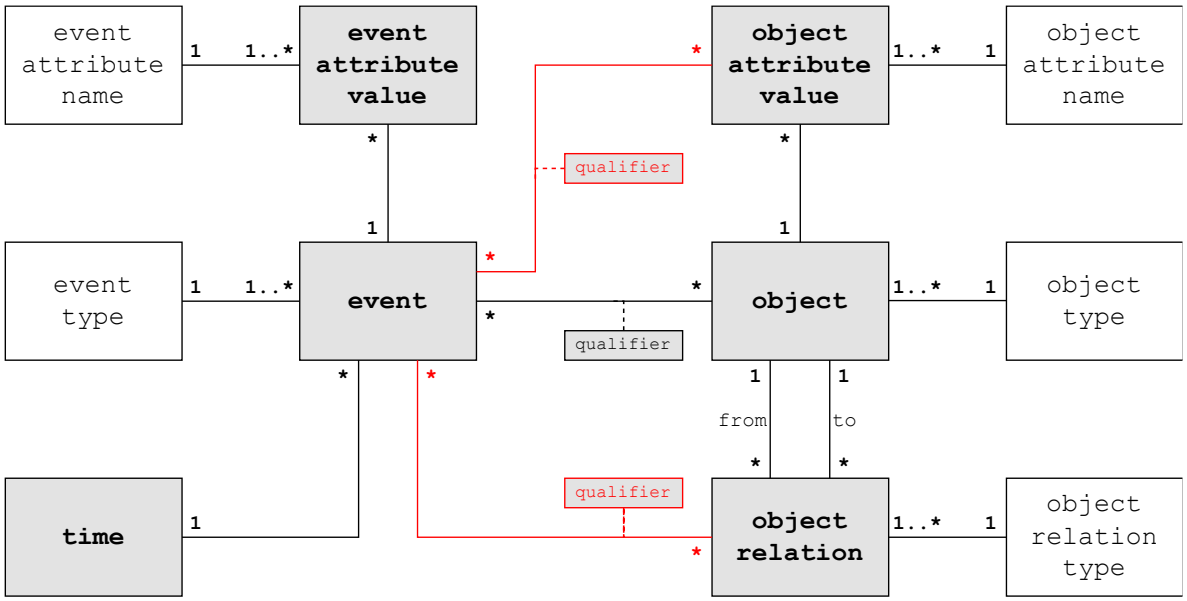

Figure 5: Object-Centric Event Data Meta Model (Full OCED-MM)

1. **`event` ↔ `object attribute`** | An `event` and an `object attribute value` can be related in a qualified (i.e. association class) manner, meaning their type of relationship is denoted. While a minimum set of qualifiers is predefined (`CREATE`, `MODIFY` and `DELETE`), additional qualifiers can be introduced as part of the data capture and used to reflect the semantics of the relationship. Each `object attribute value` can be involved in an arbitrary number of `events`, while each `event` can be related to an arbitrary number of `object attribute values`. This means, there can be `events` without `object attribute values` and vice-versa. In order to minimize the need to capture these relationships, any `object attribute value` that is not created explicitly (i.e. after it's `object` is in existence), is created implicitly with the `CREATE` of the `object` itself. When `objects` get deleted, all of their `object attribute values` are deleted implicitly.
2. **`event` ↔ `object relation`** | An `event` and an `object relation` can be related in a qualified (i.e. association class) manner, meaning their type of relationship is denoted. While a minimum set of qualifiers is predefined (`CREATE`, `MODIFY` and `DELETE`), additional qualifiers can be introduced as part of the data capture and used to reflect the semantics of the relationship. Each `object relation` can be involved in an arbitrary number of `events`, while each `event` can be related to an arbitrary number of `object relations`. This means, there can be `events` without `object relations` and vice-versa. When `objects` get deleted, all of their `object relations` are deleted implicitly.

Known Limitations

The meta model shall be kept as simple as possible, while retaining relevance in industry and academia. This implies that there are some known limitations (and corresponding workarounds):

1. **`event atomicity & no event ↔ event`** | An `event` is atomic (i.e has exactly one timestamp) and it is not possible to directly relate `events` to each other, hence, amongst others, causality relations, activities with both a start and end `event` and partial orders cannot be stored explicitly. `Events` may be *indirectly* related through shared (abstract) `objects`, however such semantics are not prescribed by the meta model. Examples:

- duration of an activity can be achieved thorugh an (abstract) proxy `object` linking to exactly two `events`: start and end
- transaction types can be represented through an (abstract) proxy `object` linking to multiple `events` (e.g. *[scheduled]*, *[started]*, *[completed]*, *[archived]*)
- grouping of `events` can be achieved through an (abstract) proxy `object` linking to multiple `events` (i.e. groups of `events`)

2. **`no nested attribute values`** | The data type of any `event attribute value` and `object attribute value` are consistent throughout the whole model, i.e., all values of the same `attribute name` can only be of type *string*, *boolean*, *integer*, *real*, *date*, *time* or *timestamp* (*date*, *time* and *timestamp* in accordance with ISO 8601-1:2019). Nested types are not supported. One can, however, define (abstract) proxy `objects` which are related to the main `object` via an `object relation` (e.g. `CHILD_OF`). With `object attribute values` of the child `objects`, nesting can be mimicked.

3. **`limited semantics`** | `Events`, `object types`, and `object` & `event attribute values` may carry semantics specific to selected domains. It is encouraged to establish such conventions, however these are not enforced by the meta model.

4. **`binary object relations`** | `Object relations` in the meta model are directed and binary. Even though some of the relationships in real life are tertiary or of higher order, such relationships are less frequent and can be represented by multiple binary `object relations`.

5. **`multiple ways to capture reality`** | The full meta model allows an `event` to refer to an `object`, an `object attribute value`, and/or an `object relation`. But an `object` can also refer to the same `object attribute value` and the same `object relation`. In case a log producer uses multiple of these linkage options simultaneously, cycles may be introduced leading to inconsistent data capture. Given the implicit semantics described before, such inconsistency can be prevented, however, this is not enforced by the meta model itself.

6. **`meta model vs. reference implementations`** | The meta model does not prescribe how things are stored or syntactically represented. For example, there may be one table per `event type` and one table per `object type`. The qualified relations may also correspond to tables. Things will be typed, but this is outside the scope of the meta model and needs to be detailed for reference implementations of this meta model.