Process and Data Science Group

RWTH Aachen University

OCEL Standard

Authors:
Anahita Farhang Ghahfarokhi
Gyunam Park
Alessandro Berti
Wil van der Aalst

January 8th, 2020

Contents

1 Introduction

Event logs are the starting point to apply process mining techniques. Classic event logs consist of events that are characterized by a case identifier, referring to a unique process instance (e.g., *patient*, *manufacturing product*, etc.), an activity, a timestamp at which the event occurred, and some additional attributes such as blood pressure, weight, etc. The XES standard[1] is an XML-based standard for such traditional event logs. The XES standard describes event logs where a single case notion (i.e., process instance) need to be chosen.

However, in reality, we face information systems such as SAP ERP systems that support processes that cannot be reduced to a single case notion. In these processes various objects interact and one event may involve a mixture of objects, e.g., *orders*, *items*, and *packages*. Therefore, a unifying case notion is missing. Selecting one of the object types as a case notion provides a specific view on the process and may lead to convergence and divergence problems. Moreover, the resulting process models will be inherently incomplete. Fortunately, we can also extract object-centric event logs (i.e., event logs with multiple case notions) from such systems. Such logs are positioned in-between the data in the system and "flat" logging formats such as XES. The purpose of this document is to provide a general standard for Object-Centric Event Logs (OCELs).

A few logging formats have been proposed to tackle the problem. These formats are not widely adopted because of their complexity and performance problems (e.g., XOC allows reconstructing the entire database state). An informal way to represent OCELs is in terms of two tables. One table represents the event records, where each row corresponds to a distinct event, as shown in Table 1. The other table represents the relevant information of objects in the information systems, as described in Table 2.

The purpose of the OCEL standard is to provide a general standard to interchange event data with multiple case notions. The goal is to exchange data between information systems and process mining analysis tools. When developing the OCEL standard, we started from the following goals:

- *Interoperability*: with the provision of the OCEL standard and JSON/XML serializations of OCEL, we want to support a widespread collection of languages and systems.
- *Generalization*: the standard supports the storage of events, objects, and their attributes. Furthermore, the standard can be extended.
- *Provision of a collection of examples*: example logs, extracted from information systems supporting some widespread business processes, are provided for the OCEL

[1] `https://xes-standard.org`

standard.

- *Tool/Library Support*: to support the implementation of OCEL in custom applications, tool/library support shall be provided.

Table 1: Informal representation of the events of an OCEL. Each row (except the header) represents an event.

id	activity	timestamp	item	order	package	prepaid-amount	weight	total-weight
e_1	*place order*	*2020-07-09 08:20:01.527+01:00*	$\{ i_1, i_2 \}$	$\{ o_1 \}$		*200.0*		
e_2	*check availability*	*2020-07-09 08:21:01.527+01:00*	$\{ i_1 \}$				*10.0*	
e_3	*load package*	*2020-07-09 08:22:01.527+01:00*			$\{ p_1 \}$			*100.0*

Table 2: Informal representation of the objects of an OCEL. Each row (except the header) represents an object.

id	type	customer	costs	color	size
o_1	*order*	*Apple*	*3500.0*		
i_2	*item*			*green*	*small*

The rest of the document is organized as follows: Section 2 proposes a conceptualization of OCELs. Section 3 defines formally OCELs. Section 4 introduces the specification of the format. Section 5 describes two serializations using JSON and XML, resulting in concrete representations that can be exchanged between systems. Section 6 provides some realistic logs expressed in the JSON and XML formats. Section 7 provides library support for OCEL. Appendix A in Section 8 contains the schema of both JSON-OCEL and XML-OCEL.

2 Object-centric Event Log: Conceptualization

In this section, we provide a conceptualization of Object-Centric Event Logs (OCEL). Figure 1 shows the class diagram for this conceptualization. An object-centric event log contains events and objects related to a business process, such as Order-to-Cash (O2C) and Purchase-to-Pay (P2P) processes in ERP systems.

Each event represents an execution record of an underlying business process. An event is associated with an identifier, an activity, and a timestamp. The activity and timestamp represent what happens in the execution and when it happens, respectively. In the first row of Table 1, the event, having identifier of e_1, is associated with the execution of *place order* at *2020-07-09 08:20:01.527+01:00*. Also, the execution of an event involves a set of related objects. In this event, three objects (o_1, i_1, and i_2) are involved in its execution. An event may involve several event attributes. Each event attribute has an attribute name and attribute value. Attribute names and attribute values are related

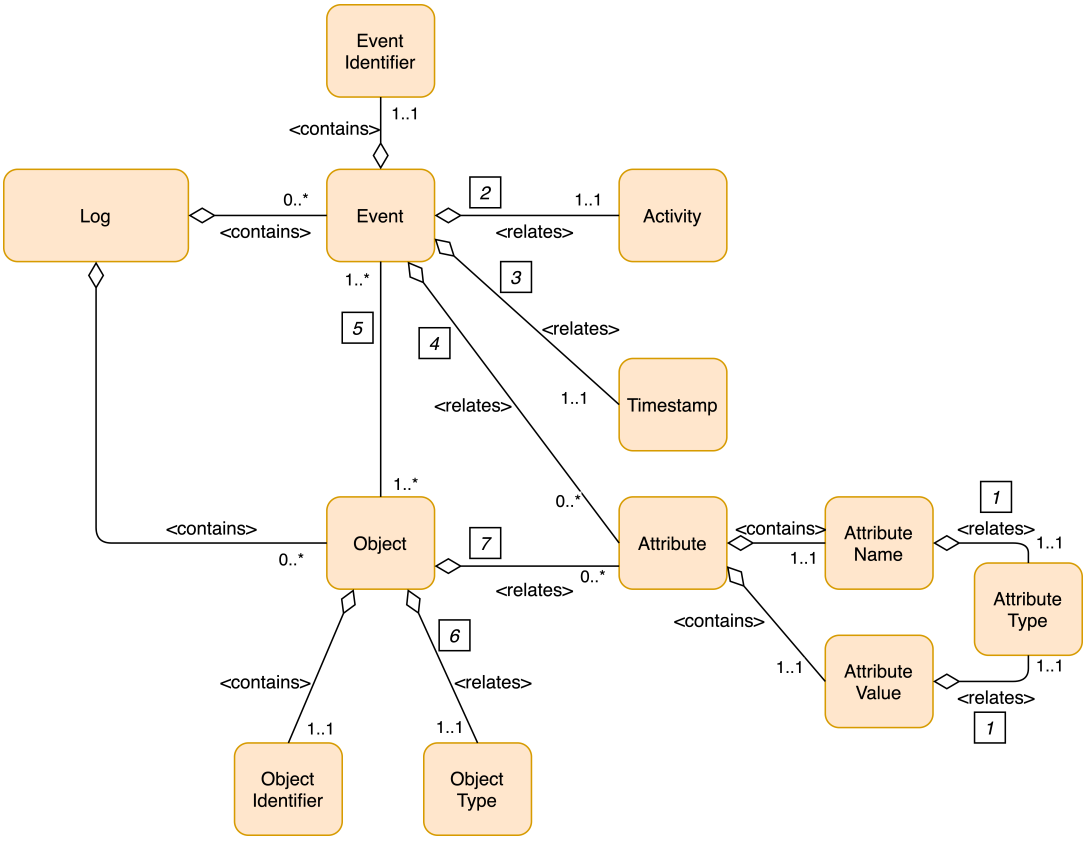

Figure 1: The UML class diagram for conceptualizing the OCEL.

to attribute types. For example, e_1 has an attribute *prepaid-amount* (attribute name), and its value is *200.0* (of type *float*). The attribute name and attribute value that are contained in an attribute related to single event, have the same type.

Objects represent physical and informational entities composing business processes such as materials, documents, products, invoices, etc. Each object has an identifier and is associated with an object type. In the second row of Table 2, the object with identifier i_1 is an item (i.e., object type). Moreover, objects may contain multiple attributes. Each attribute has its own name and value that are associated with a type. For example, i_2 has the *green* color and *small* size.

As shown in Figure 1, there exist many-to-many relationships between objects and events. Multiple objects can be involved in an event (e.g., o_1, i_1, and i_2 are involved in e_1 in Table 1), and an object can be involved in many events (e.g., i_1 is involved in e_1 and e_2 in Table 1).

The numbers in the squares of Figure 1 indicate the functions that associate the sources'

values to the targets (e.g., activity values to events). In the following section, we will provide the formal definition of OCELs based on the conceptualization.

3 Object-centric Event Log: Formal Definitions

This section presents the definition of an OCEL. First, we define the universes that are used in the definitions.

Definition 1 (Universes)**.** Below are the universes used in the formal definition of OCEL:

- U_e is the universe of event identifiers.
  $Example$: $U_e = \{e_1, e_2, e_3, \ldots\}$
- U_{act} is the universe of activities.
  $Example$: $U_{act} = \{$ *place order*, *check availability*, $\ldots\}$
- U_{att} is the universe of attribute names.
  $Example$: $U_{att} = \{$ *resource*, *weight*, $\ldots\}$
- U_{val} is the universe of attribute values.
  $Example$: $U_{val} = \{$ *500*, *1000*, *Mike*, $\ldots\}$
- U_{typ} is the universe of attribute types.
  $Example$: $U_{typ} = \{$ *string*, *integer*, *float*, $\ldots\}$
- U_o is the universe of object identifiers.
  $Example$: $U_o = \{o_1, i_1, \ldots\}$
- U_{ot} is the universe of objects types.
  $Example$: $U_{ot} = \{$ *order*, *item*, $\ldots\}$
- U_{timest} is the universe of timestamps.
  $Example$: $U_{timest} = \{$ *2020-07-09T08:21:01.527+01:00*, $\ldots\}$

Using the universes, we define OCELs.

Definition 2 (Object-Centric Event Log)**.** An object-centric event log is a tuple $L = (E, AN, AV, AT, OT, O, \pi_{typ}, \pi_{act}, \pi_{time}, \pi_{vmap}, \pi_{omap}, \pi_{otyp}, \pi_{ovmap}, \leq)$ such that:

- $E \subseteq U_e$ is the set of event identifiers. E is a subset of U_e. As shown in Figure 1, each event is related to an event identifier.
  $Example$: the first event shown in Table 1 is related to the event identifier e_1.
- $AN \subseteq U_{att}$ is the set of attributes names. AN is a subset of U_{att}. As shown in Figure 1, an event or object can have zero to many attributes. An attribute contains a name.

$Example$: in Table 1 *resource*, *prepaid-amount*, *weight*, and *total-weight* are attribute names and, in Table 2, *costs*, *color*, and *size* are attribute names.

- $AV \subseteq U_{val}$ is the set of attribute values (with the requirement that $\mathrm{AN} \cap \mathrm{AV} = \emptyset$). AV is a subset of U_{val}. As shown in Figure 1, an event or object can have zero to many attributes. An attribute contains a value.
  $Example$: in Table 1 *200.0*, *Anahita*, and *10.0* are attribute values, and in Table 2, *Apple*, *green*, and *3500.0* are examples of attribute values.

- $AT \subseteq U_{typ}$ is the set of attribute types. AT is a subset of U_{typ}. Therefore, each member of AT can be a type for the attributes that are related to either an event or an object.
  $Example$: the type of attribute *resource* is Table 1 is *string*.

- $OT \subseteq U_{ot}$ is the set of object types. OT is a subset of U_{ot}. Each object is related to an object type, as shown in Figure 1.
  $Example$: in Table 2, for the first object, the type is *order*.

- $O \subseteq U_o$ is the set of object identifiers. O is a subset of U_o. Each object is related to an object identifier, as shown in Figure 1.
  $Example$: the first object in Table 2 is related to the object identifier o_1.

- $\pi_{typ} : AN \cup AV \rightarrow AT$ is the function associating an attribute name or value to its corresponding type (also see Relation 1 in Figure 1). Each attribute name and attribute value in the event log is associated with a type. For each event and object, we need to check the consistency between the type of the attribute name and the attribute value associated with that. We will check this consistency in the following definitions.

  $Example$: for the attributes in Table 1, π_{typ}(*prepaid-amount*) = *float*, π_{typ}(*200.0*) = *float*.

- $\pi_{act} : E \rightarrow U_{act}$ is the function associating an event (identifier) to its activity (also see Relation 2 in Figure 1). Each event in the event log is associated with an activity.
  $Example$: for the first event shown in Table 1, the activity is *place order*.

- $\pi_{time} : E \rightarrow U_{timest}$ is the function associating an event (identifier) to a timestamp (also see Relation 3 in Figure 1). Each event in the event log is associated with a timestamp.
  $Example$: for the first event shown in Table 1, the timestamp is *2020-07-09T08:21:01.527+01:00*.

- $\pi_{vmap} : E \rightarrow (AN \not\rightarrow AV)$ such that

$$\pi_{typ}(n) = \pi_{typ}(\pi_{vmap}(e)(n)) \quad \forall e \in E \; \forall n \in \mathrm{dom}(\pi_{vmap}(e))$$

is the function associating an event (identifier) to its attribute value assignments (also see Relation 4 in Figure 1). As shown in Figure 1, an event is related to attributes that contain an attribute name and attribute value. As discussed in π_{typ}, each attribute name and attribute value have a type. They should have the same type for one event. Therefore, we check whether the type of the attribute name and the attribute value associated with that match with each other.

$Example$: for the first event in Table 1, $\pi_{vmap}(e_1)(\textit{prepaid-amount}) = \textit{200.0}$

- $\pi_{omap} : E \to \mathcal{P}(O)$ is the function associating an event (identifier) to a set of related object identifiers (also see Relation 5 in Figure 1). As shown in Figure 1 and through Relation 5, an event can contain many objects.
  $Example$: the first event in Table 1 is related to three objects $\pi_{omap}(e_1) = \{o_1, i_1, i_2\}$.
- $\pi_{otyp} \in O \to OT$ assigns precisely one object type to each object identifier (also see Relation 6 in Figure 1).
  $Example$: for the first object in Table 2, $\pi_{otyp}(o_1) = \textit{order}$.
- $\pi_{ovmap} : O \to (AN \not\to AV)$ such that

$$\pi_{typ}(n) = \pi_{typ}(\pi_{ovmap}(o)(n)) \quad \forall n \in \mathrm{dom}(\pi_{ovmap}(o)) \; \forall o \in O$$

  is the function associating an object to its attribute value assignments (also see Relation 7 in Figure 1). As shown in Figure 1, an object is related to attributes that contain an attribute name and attribute value. As discussed in π_{typ}, each attribute name and attribute value have a type. They should have the same type for one object. Therefore, we check whether the type of the attribute name and the attribute value associated with that match with each other.
  $Example$: for the second object in Table 2, $\pi_{ovmap}(i_2)(\textit{color}) = \textit{green}$.
- $\leq$ is a total order (i.e., it respects the antisymmetry, transitivity, and connexity properties). A possible way to define a total order is to consider the timestamps associated with the events as a pre-order (i.e., assuming some arbitrary, but fixed, order for events having the same timestamp).

4 Specification

This section introduces the specification to implement the OCEL based on the formal definition. To this end, we first introduce the meta-model for the specification of the OCEL. Afterward, we illustrate the specification in more detail by connecting the meta-model to the formal definition.

For the presentation of the specification, we take the minimal OCEL informally described in Tables 1 and 2.

4.1 Meta-model for the specification of OCEL

The meta-model for the specification of OCEL is shown in Figure 2 as a UML 2.0 class diagram. The meta-model defines three classes, i.e., log, event, and object. The description for each class is as follows:

- **Log:** The log class contains an arbitrary number of events and objects as described in Section 2. A log may also contain global elements such as global log element, global event element, and global object element. These log elements will be specified in Section 4.2.
- **Event:** Each event represents an execution record of an underlying business process. As explained in Section 2, an event contains required elements (e.g., id, activity, timestamp, and relevant objects) and possibly also optional elements (e.g., event attributes). In Section 4.2, it will be specified how these elements are implemented based on the meta-model.
- **Object:** Each object indicates the information of an object instance in the business process. As introduced in Section 2, an object contains required (e.g., type) and optional (e.g., *color* and *size*) elements. In Section 4.2, it will be specified how these elements are implemented based on the meta-model.

The log, event, and object classes only define the structure of logs. The actual information of the classes are realized with the element class.

- **Element:** An element is composed of a key and value(s). The key is string-based, whereas the value may be *string*, *timestamp*, *integer*, *float*, and *boolean*. Below is the description for each value type:
  - **String** elements hold literal information which is generally untyped and of arbitrary length., e.g., *prepaid-amount*.
  - **Timestamp** elements hold information about a specific point in time (with milliseconds precision). The timestamp format is according to the *xs:dateTime* data type[2], e.g., *2020-07-09T08:21:01.527+01:00*.
  - **Integer** elements have a numeric value (a discrete integer number), e.g., *2*.
  - **Float** elements have a float value (a floating-point number), e.g., *23.4*.
  - **Boolean** elements are elements that can only be true or false, e.g., $True$.

[2]See `https://www.w3schools.com/xml/schema_dtypes_date.asp`

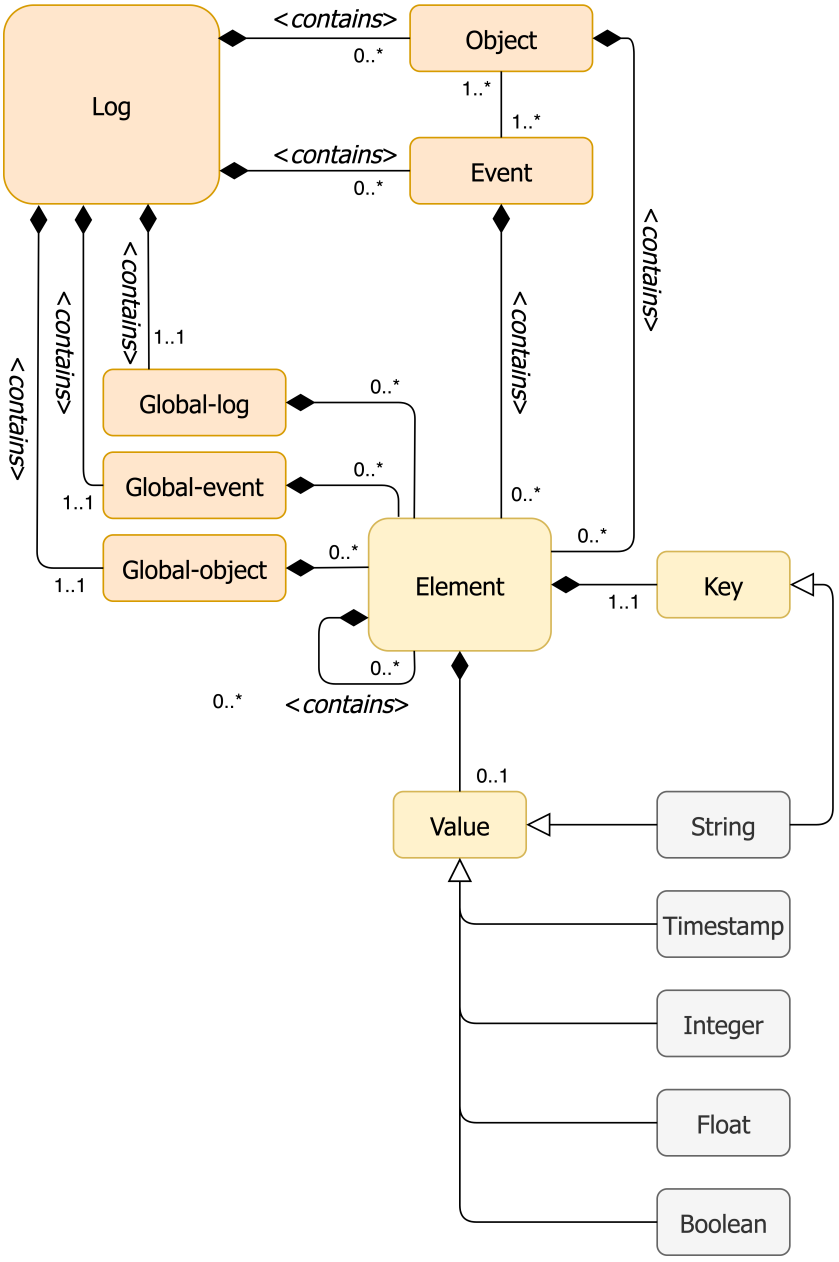

Figure 2: The UML class diagram for the complete meta-model for the OCEL standard showing the classes and their relations.

An element can be nested, i.e., a parent element can contain child elements. Among nested elements, we have two important categories:

- We define a **list** as a nested element where the child elements' keys are the same and their values have the same data type. For instance,

  ocel:attribute-names: [*name*:*color*, *name*:*costs*, *name*:*weight*]

  is a list. For the remainder of this document, we denote the list by discarding the child elements' keys, e.g., *ocel:attribute-names*: [*color*, *costs*, *weight*].

- We define a **map** as a nested element whose child elements have different keys. For instance,

  ocel:ovmap: { *color*: *green*, *size*: *small* }

  is a map since the keys (i.e., *color* and *size*) are different.

The conceptualization model in Figure 1 is related to the meta-model in Figure 2. The log, event, and object entities in Figure 1 can be mapped to the log, event, and object entities in Figure 2. The values that are associated with the event (e.g., event identifier, activity and timestamp in Figure 1) are realized with the elements (at the event level) in Figure 2. Also, the values related to the object (e.g., object identifier and type in Figure 1) are realized with the elements in Figure 2.

4.2 Detailed Specification

This section explains how the classes in the meta model are implemented with the elements. As described in Figure 2, an OCEL is composed of global log, global event, global-object, event, and object classes. The actual information in each class is realized with the elements. For each class, we define required and optional elements. Figure 3 shows an overview of the element formation. For instance, a global log class contains elements such as version, attribute names, and object types. In the following, we explain each element with a short description, a key, an expected value type, and an example. Moreover, we clarify the connection to the formal definition in Section 3, as shown in Figure 3. As design choices, we do not implement the set of attribute values (i.e., AV), the set of attribute types (i.e., AT), and the function that associates an attribute to its type (i.e., π_{typ}) in the current version of the standard. The reasoning is as follows:

1. AV is infinite by nature, making the following serialization unwieldy.
2. AT and π_{typ} are implicitly available in the contemporary implementation options, making the serialization redundant.

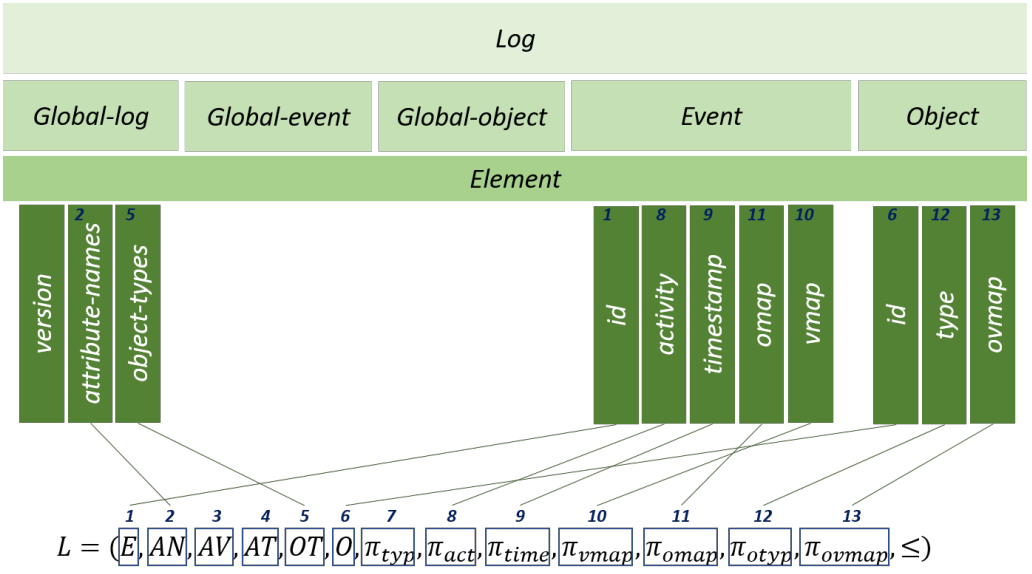

Figure 3: The detailed elements of OCEL and their connection to the formal definition.

Furthermore, the specification involves implementation-specific information such as version name and default values for missing event and object information.

First, an OCEL contains a global log, global event, and global object element.

- **global-log**: specifies the version, attribute names, and object types that compose the log.
  - *key:* ocel:global-log
  - *value:* map with its child elements having list value type.
  - *required:* yes
  - *example:*
    ocel:global-log:
    ocel:version: *0.1*,
    ocel:attribute-names: [*cost*, *color*, *size*],
    ocel:object-types: [*customer*, *order*, *item*]
  - **version**: the version of the OCEL standard followed by the log.
    - *key:* ocel:version
    - *value:* string
    - *required:* yes

    - *example: ocel:version*: *0.1*
  - **attribute-names**: a list of attribute names that are used in event or object elements. It is linked to AN in the definition.
    - *key:* ocel:attribute-names
    - *value:* list of strings
    - *required:* yes
    - *example: ocel:attribute-names*: [*color*, *costs*, *weight*]
  - **object-types**: a list of object types that are used in the log. It is linked to OT in the definition, and each object is mapped onto its type through π_{otyp}.
    - *key:* ocel:object-types
    - *value:* list of strings
    - *example: ocel:object-types*: [*customer*, *item*, *order*]
- **global-event**: specifies some default values for the elements when they are not specified by the event's attribute map; hence, these elements are specified for all the events.
  - *key:* ocel:global-event
  - *value:* map with its child elements having a string value type.
  - *required:* yes
  - *example: ocel:global-event*: {*id*: *invalid*, *activity*: *invalid*, *timestamp*: *invalid*}

  The above example define that every event in the log has valid elements with key *id*, *activity*, and *timestamp*. When the values are missing for the elements, the default values (i.e., *invalid*) fill these missing values.
- **global-object**: specifies some default values for the elements when they are not specified by the object's attribute map; hence, these elements are specified for all the objects.
  - *key:* ocel:global-object
  - *value:* map with its child elements having a string value type.
  - *required:* yes
  - *example: ocel:global-object*: {*id*: *invalid*, *type*: *invalid*}

The above example defines that every object in the log has valid elements with keys *id* and *type*. Missing values are replaced with *invalid*.

- **Event:** an event contains the *id*, *activity*, *timestamp*, *omap*, and *vmap* elements.
  - **id**: each event should have an identifier that is a string. It is linked to E in the definition, and each event is related to an event identifier in E.
    - *key:* ocel:id
    - *value:* string
    - *required:* yes
    - *example: ocel:id*: e_1
  - **activity**: each event should have an activity that shows the name of the task executed. In the definition, each event is related to an activity through π_{act}.
    - *key:* ocel:activity
    - *value:* string
    - *required:* yes
    - *example: ocel:activity*: *place order*
  - **timestamp**: each event should have a timestamp that shows the point of time at which the event occurred. In the definition, each event is related to a timestamp through π_{time}.
    - *key:* ocel:timestamp
    - *value:* date
    - *required:* yes
    - *example: ocel:timestamp*: *2020-07-09T08:20:01.527+01:00*
  - **omap**: each event can have an *omap* that shows the list of objects involved in that event. In the definition, the π_{omap} function associates each event to a set of related objects.
    - *key:* ocel:omap
    - *value:* list of strings
    - *required:* yes
    - *example: ocel:omap*: $[o_1, i_1, i_2]$

  - **vmap**: each event can have a *vmap* that is a nested element. In the definition, each event is mapped to its attribute value mapping using the π_{vmap} function.
    * *key:* ocel:vmap
    * *value:* map with its child elements having a string value type.
    * *required:* no
    * *example: ocel:vmap*: { *resource* : *Alessandro*, *prepaid-amount*: *200.0*}

- **Object:** an object contains the *id*, *type*, *ovmap* elements.
  - **id**: each object should have an id that is a string. It is linked to O in the definition, and each object is related to an object identifier in O.
    * *key:* ocel:id
    * *value:* string
    * *required:* yes
    * *example: ocel:id*: o_1
  - **type**: each object should have a *type* that is a *string*. It is linked to π_{otyp} in definition, that associates a type to the object.
    * *key:* ocel:type
    * *value:* string
    * *required:* yes
    * *example: ocel:type*: *order*
  - **ovmap**: each object can have an *ovmap* that is a nested element. Using the π_{ovmap} function, each object is mapped onto its attribute value mapping.
    * *key:* ocel:ovmap
    * *value:* map with its child elements having a string value type.
    * *required:* no
    * *example: ocel:ovmap*: { *color* : *green*, *size* : *small* }

5 Serialization of OCEL

In this section, we propose two implementations of the OCEL standard and provide the corresponding serialization (JSON/XML). To illustrate the two serializations, we take

as example the event log informally described in Tables 1 and 2. While the example is (deliberately) minimal, it is large enough to introduce the two serializations.

5.1 XML Serialization of OCEL

The XML serialization of OCEL (XML-OCEL) follows the specification proposed in Figure 2. An example is provided in Listing 1. The example shows the serialization of the event log informally described in Tables 1 and 2.

The element with the *log* tag contains four mandatory elements as follows:

- The global with *scope='log'* provides access to three properties:
  - The log has an element *version* to explain the version of the OCEL standard.
  - A list with key *attribute-names*, that contains a list of the attribute names used in the events/objects. Each attribute name is specified as a string with key *attribute-name* and the attribute's name as value.
  - A list with key *object-type*, that contains a list of the object types associated to the log's objects. Each object type is specified as a string with key *object-type* and the object type as value.
- The global with *scope='event'* specifies the mandatory elements of events and the elements' values when they are not directly provided as properties of the event. These are specified as XML elements having the type as tag, the attribute name as key, and the default value as value.
- The global with *scope='object'* specifies the mandatory elements of objects and the elements' values when they are not directly provided as properties of the object. These are specified as XML elements having the type as tag, the attribute name as key, and the default value as value.

Also, the log element contains different events (each one having the *event* tag, and being contained inside the *events* element), and different objects (each one having the *object* tag, and being contained inside the *objects* element). Each event and object contains different properties (the type of each property is given by the tag, the name is given by the key).

Each event contains an identifier (with key *id*, and having type *string*), an activity (with key *activity*, and having type *string*), and a timestamp (with key *timestamp*, and having type *date*). Moreover, each event is associated with an object map (with key *omap*, and having type *list*) and an attribute map (with key *vmap*, and having type *list*). The attribute map contains the attributes associated with the event.

Each object contains an identifier (with key *id*, and having type *string*), is associated with an object type (with key *type*, and having type *string*), and to an attribute map (with key *ovmap*, and having type *map*). The attribute map contains the attributes associated with the object. As an implementation choice, the prefix *ocel:* has been elided from the XML-OCEL format's keys. Listing 1 contains only an example log. The validation constraints are reported in Appendix 8.

Listing 1: XML-OCEL example

```
<?xml version='1.0' encoding='UTF-8'?>
<log>
  <global scope="log">
    <string key="version" value="0.1" />
    <list key="attribute-names">
      <string key="name" value="color" />
      <string key="name" value="costs" />
      <string key="name" value="customer" />
      <string key="name" value="prepaid-amount" />
      <string key="name" value="resource" />
      <string key="name" value="size" />
      <string key="name" value="total-weight" />
      <string key="name" value="weight" />
    </list>
    <list key="object-types">
      <string key="type" value="customer" />
      <string key="type" value="item" />
      <string key="type" value="order" />
      <string key="type" value="package" />
      <string key="type" value="product" />
    </list>
  </global>
  <global scope="event">
    <string key="id" value="__INVALID__" />
    <string key="activity" value="__INVALID__" />
    <string key="timestamp" value="__INVALID__" />
    <string key="omap" value="__INVALID__" />
  </global>
  <global scope="object">
    <string key="id" value="__INVALID__" />
    <string key="type" value="__INVALID__" />
  </global>
  <events>
    <event>
      <string key="id" value="e1" />
      <string key="activity" value="place_order" />
      <date key="timestamp" value="2020-07-09 08:20:01.527+01:00" />
      <list key="omap">
        <string key="object-id" value="i1" />
        <string key="object-id" value="o1" />
        <string key="object-id" value="i2" />
      </list>
      <list key="vmap">
        <string key="resource" value="Alessandro" />
        <float key="prepaid-amount" value="200.0" />
      </list>
    </event>
    <event>
      <string key="id" value="e2" />
      <string key="activity" value="check_availability" />
```

```
51        <date key="timestamp" value="2020-07-09 08:21:01.527+01:00" />
52        <list key="omap">
53          <string key="object-id" value="i1" />
54        </list>
55        <list key="vmap">
56          <string key="resource" value="Anahita" />
57          <float key="weight" value="10.0" />
58        </list>
59      </event>
60      <event>
61        <string key="id" value="e3" />
62        <string key="activity" value="load_package" />
63        <date key="timestamp" value="2020-07-09 08:22:01.527+01:00" />
64        <list key="omap">
65          <string key="object-id" value="r1" />
66          <string key="object-id" value="p1" />
67        </list>
68        <list key="vmap">
69          <string key="resource" value="Gyunam" />
70          <float key="total-weight" value="100.0" />
71        </list>
72      </event>
73    </events>
74    <objects>
75      <object>
76        <string key="id" value="o1" />
77        <string key="type" value="order" />
78        <list key="ovmap">
79          <string key="customer" value="Apple" />
80          <float key="costs" value="3500.0" />
81        </list>
82      </object>
83      <object>
84        <string key="id" value="i1" />
85        <string key="type" value="item" />
86        <list key="ovmap" />
87      </object>
88      <object>
89        <string key="id" value="i2" />
90        <string key="type" value="item" />
91        <list key="ovmap">
92          <string key="color" value="green" />
93          <string key="size" value="small" />
94        </list>
95      </object>
96      <object>
97        <string key="id" value="p1" />
98        <string key="type" value="package" />
99        <list key="ovmap" />
100     </object>
101     <object>
102       <string key="id" value="r1" />
103       <string key="type" value="product" />
104       <list key="ovmap" />
105     </object>
106   </objects>
107 </log>
```

5.2 JSON Serialization of OCEL

The JSON serialization of OCEL (JSON-OCEL) follows the specification proposed in Figure 2. An example is provided in Listing 1. The example shows the serialization of the event log informally described in Tables 1 and 2. In this implementation, a log is an object containing different properties.

The four mandatory elements of the log are contained as follows:

- The *ocel:global-log* property, that provides access to two properties:
  - The log has property *ocel:version* to explain the version of OCEL standard.
  - The *ocel:attribute-names* property, related to a list of attribute names of the log's events/objects.
  - The *ocel:object-types* property, related to a list of object types for the log objects.
- The *ocel:global-event* property, that specifies the mandatory elements of events and the values of the elements when they are not directly provided as properties of the event. These are specified in a dictionary.
- The *ocel:global-object* property, that specifies the mandatory elements of objects and the values of the elements when they are not directly provided as properties of the object. These are specified in a dictionary.

The events are contained in the *ocel:events* key. The events are expressed as a map between the event identifier and a dictionary containing the activity (*ocel:activity*), the timestamp (*ocel:timestamp*), the object map (*ocel:omap*), that contains a list of the related objects, and the attribute map (*ocel:vmap*), which corresponds to each attribute name an attribute value.

The objects are contained in the *ocel:objects* key. The objects are expressed as a map between the object identifier and a dictionary containing the object type (*ocel:type*) and the attribute map (*ocel:ovmap*), which corresponds to each attribute name an attribute value.

Listing 2 contains only an example log. The validation constraints are reported in Appendix 8.

Listing 2: JSON-OCEL example

```
1 {
2   "ocel:global-log": {
3     "ocel:version": "1.0",
4     "ocel:attribute-names": [
5       "color",
6       "costs",
```

```
      "customer",
      "prepaid_amount",
      "resource",
      "size",
      "total_weight",
      "weight"
    ],
    "ocel:object_types": [
      "customer",
      "item",
      "order",
      "package",
      "product"
    ]
  },
  "ocel:global_event": {
    "ocel:activity": "__INVALID__"
  },
  "ocel:global_object": {
    "ocel:type": "__INVALID__"
  },
  "ocel:events": {
    "e1": {
      "ocel:activity": "place_order",
      "ocel:timestamp": "2020-07-09 08:20:01.527+01:00",
      "ocel:omap": [
        "i1",
        "o1",
        "i2"
      ],
      "ocel:vmap": {
        "resource": "Alessandro",
        "prepaid_amount": 200.0
      }
    },
    "e2": {
      "ocel:activity": "check_availability",
      "ocel:timestamp": "2020-07-09 08:21:01.527+01:00",
      "ocel:omap": [
        "i1"
      ],
      "ocel:vmap": {
        "resource": "Anahita",
        "weight": 10.0
      }
    },
    "e3": {
      "ocel:activity": "load_package",
      "ocel:timestamp": "2020-07-09 08:22:01.527+01:00",
      "ocel:omap": [
        "r1",
        "p1"
      ],
      "ocel:vmap": {
        "resource": "Gyunam",
        "total_weight": 100.0
      }
    }
  },
  "ocel:objects": {
    "o1": {
```

```
68        "ocel:type": "order",
69        "ocel:ovmap": {
70          "customer": "Apple",
71          "costs": 3500.0
72        }
73      },
74      "i1": {
75        "ocel:type": "item",
76        "ocel:ovmap": {
77          "color": NaN,
78          "size": NaN
79        }
80      },
81      "i2": {
82        "ocel:type": "item",
83        "ocel:ovmap": {
84          "color": "green",
85          "size": "small"
86        }
87      },
88      "p1": {
89        "ocel:type": "package",
90        "ocel:ovmap": {}
91      },
92      "r1": {
93        "ocel:type": "product",
94        "ocel:ovmap": {}
95      }
96    }
97  }
```

6 Available Event Logs

This section provides links to a few more realistic examples of logs in the OCEL standard (using both the JSON and XML serializations), with more events and objects than the original example.

Order Management log
XML serialization: `http://ocel-standard.org/temp/1.0/running-example.xmlocel`
JSON serialization: `http://ocel-standard.org/temp/1.0/running-example.jsonocel`
Number of events: 22367 **Number of objects:** 11522
Description: The log represents a synthetic order management system, with many different activities and object types (orders, items, products, and customers).

SAP ERP IDES instance - O2C log
XML serialization: `http://ocel-standard.org/temp/1.0/o2c.xmlocel`
JSON serialization: `http://ocel-standard.org/temp/1.0/o2c.jsonocel`
Number of events: 98350 **Number of objects:** 107767
Description: The log, extracted from an SAP IDES instance, represents an order-to-

cash process and contains object types (customers, orders, the delivery, and the invoice).

SAP ERP IDES instance - P2P log
XML serialization: `http://ocel-standard.org/temp/1.0/p2p.xmlocel`
JSON serialization: `http://ocel-standard.org/temp/1.0/p2p.jsonocel`
Number of events: 24854 **Number of objects:** 74489
Description: The log, extracted from an SAP IDES instance, represents a procure-to-pay process and contains the orders sent to the suppliers, the events related to the retrieval of the materials, and the invoicing.

7 Library Support

The OCEL standard and serializations are defined in a system independent manner. Each tools supplier can implement support for handling OCEL from scratch. However, to facilitate adoption, a reference implementation is provided in the form of a Python library supporting the OCEL standard. The library is available at the address `https://github.com/OCEL-standard/ocel-support` (repository) and in the Pip package *ocel-standard*. The library can be imported using the *import ocel* command, and provides three main functionalities:

- Importing of JSON-OCEL and XML-OCEL: using the command *ocel.import_log(log_path)*, it is possible to import the given event log in a log object that resembles the specification.
- Exporting of JSON-OCEL and XML-OCEL: using the command *ocel.export_log(log_object, log_path)*, it is possible to export the log object to the specified path.
- Validation of JSON-OCEL and XML-OCEL: using the command *ocel.validate(log_path, schema_path)*, it is possible to validate an event log against the OCEL schema. The schemas for JSON-OCEL and XML-OCEL are available inside the folder *schemas* of the repository.

The library supports an interface to query easily the lob object. The description of the interface and of the provided commands is available at the repository of the library.

Note that, while the provided library is Python-specific, the JSON-OCEL and XML-OCEL formats can be implemented in any programming language that supports JSON and XML parsing.

8 Appedix A - Validation

This section describes the schemas of the two implementations (XML/JSON). These are useful to verify if an XML/JSON file follows the OCEL specification.

8.1 Verification of XML-OCEL

Below is the XSD stylesheet definition for the XML-OCEL serialization. This can be verified using an XML validator (as example, `https://www.freeformatter.com/xml-validator-xsd.html`).

The following constraints are checked by the XML schema:

- The allowed types for the attributes are the ones allowed by the specification (string, date, integer, float, boolean, and list).
- The values of simple attributes are validated against their corresponding XML specification:
  - The values associated to attributes of type string are validated against the XML *xs:string* specification.
  - The values associated to attributes of type date are validated against the XML *xs:date* specification.
  - The values associated to attributes of type integer are validated against the XML *xs:int* specification.
  - The values associated to attributes of type float are validated against the XML *xs:double* specification.
  - The values associated to attributes of type boolean are validated against the XML *xs:boolean* specification.
- The validation of the nested attributes is defined in the schema. If a simple attribute, or a list, contains nested attributes, these should be valid according to the schema.
- The root object of the serialization can contain the following elements:
  - *Globals*
  - *Events*: the tag can contain many XML objects of type Event.
  - *Objects*: the tag can contain many XML objects of type Object.

Listing 3: XSD stylesheet definition for XML-OCEL

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
<!-- This file describes the XML serialization of the OCL format for object-centric event log data. -->
<!-- (c) 2020 by Chair of Process and Data Science (http://www.pads.rwth-aachen.de) -->
<!-- Date: Nov. 13, 2020 -->
<!-- Version 0.1 -->
<!-- Author: Gyunam Park (gnpark@pads.rwth-aachen.de) -->
<!-- Draft for the formal format of object-centric event log data -->
<xs:element name="log" type="LogType"/>
<!-- Attributables -->
<xs:complexType name="AttributableType">
<xs:choice minOccurs="0" maxOccurs="unbounded">
<xs:element name="string" minOccurs="0" maxOccurs="unbounded" type="AttributeStringType"/>
<xs:element name="date" minOccurs="0" maxOccurs="unbounded" type="AttributeDateType"/>
<xs:element name="int" minOccurs="0" maxOccurs="unbounded" type="AttributeIntType"/>
<xs:element name="float" minOccurs="0" maxOccurs="unbounded" type="AttributeFloatType"/>
<xs:element name="boolean" minOccurs="0" maxOccurs="unbounded" type="AttributeBooleanType"/>
<xs:element name="id" minOccurs="0" maxOccurs="unbounded" type="AttributeIDType"/>
<xs:element name="list" minOccurs="0" maxOccurs="unbounded" type="AttributeListType"/>
<xs:element name="container" minOccurs="0" maxOccurs="unbounded" type="AttributeContainerType"/>
</xs:choice>
</xs:complexType>
<!-- String attribute -->
<xs:complexType name="AttributeStringType">
<xs:complexContent>
<xs:extension base="AttributeType">
<xs:attribute name="value" use="required" type="xs:string"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<!-- Date attribute -->
<xs:complexType name="AttributeDateType">
<xs:complexContent>
<xs:extension base="AttributeType">
<xs:attribute name="value" use="required" type="xs:dateTime"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<!-- Integer attribute -->
<xs:complexType name="AttributeIntType">
<xs:complexContent>
<xs:extension base="AttributeType">
<xs:attribute name="value" use="required" type="xs:long"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<!-- Floating-point attribute -->
<xs:complexType name="AttributeFloatType">
<xs:complexContent>
<xs:extension base="AttributeType">
<xs:attribute name="value" use="required" type="xs:double"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<!-- Boolean attribute -->
<xs:complexType name="AttributeBooleanType">
<xs:complexContent>
<xs:extension base="AttributeType">
<xs:attribute name="value" use="required" type="xs:boolean"/>
</xs:extension>
</xs:complexContent>
```

```
</xs:complexType>
<!-- ID attribute -->
<xs:complexType name="AttributeIDType">
<xs:complexContent>
<xs:extension base="AttributeType">
<xs:attribute name="value" use="required" type="xs:string"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<!-- List attribute -->
<xs:complexType name="AttributeListType">
<xs:complexContent>
<xs:extension base="AttributeType"> </xs:extension>
</xs:complexContent>
</xs:complexType>
<!-- Container attribute -->
<xs:complexType name="AttributeContainerType">
<xs:complexContent>
<xs:extension base="AttributableType"> </xs:extension>
</xs:complexContent>
</xs:complexType>
<!-- Globals definition -->
<xs:complexType name="GlobalsType">
<xs:complexContent>
<xs:extension base="AttributableType">
<xs:attribute name="scope" type="xs:NCName" use="required"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<!-- Attribute -->
<xs:complexType name="AttributeType">
<xs:complexContent>
<xs:extension base="AttributableType">
<xs:attribute name="key" use="required" type="xs:Name"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<!-- Elements may contain attributes -->
<xs:complexType name="ElementType">
<xs:complexContent>
<xs:extension base="AttributableType"/>
</xs:complexContent>
</xs:complexType>
<!-- Logs are elements that may contain executions -->
<xs:complexType name="LogType">
<xs:complexContent>
<xs:extension base="ElementType">
<xs:sequence>
<xs:element name="global" minOccurs="0" maxOccurs="4" type="GlobalsType"/>
<xs:element name="events" minOccurs="0" maxOccurs="unbounded" type="EventsType"/>
<xs:element name="objects" minOccurs="0" maxOccurs="unbounded" type="ObjectsType"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<!-- Executions are elements that may contain events -->
<xs:complexType name="EventsType">
<xs:complexContent>
<xs:extension base="ElementType">
<xs:sequence>
<xs:element name="event" minOccurs="0" maxOccurs="unbounded" type="EventType"/>
```

```
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<!-- Events are elements -->
<xs:complexType name="EventType">
<xs:complexContent>
<xs:extension base="ElementType">
</xs:extension>
</xs:complexContent>
</xs:complexType>
<!-- Specifications are elements that may contain data -->
<xs:complexType name="ObjectsType">
<xs:complexContent>
<xs:extension base="ElementType">
<xs:sequence>
<xs:element name="object" minOccurs="0" maxOccurs="unbounded" type="ObjectType"/>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<!-- Objects are elements -->
<xs:complexType name="ObjectType">
<xs:complexContent>
<xs:extension base="ElementType">
</xs:extension>
</xs:complexContent>
</xs:complexType>
</xs:schema>
```

8.2 Verification of JSON-OCEL

Below is the JSON schema for the JSON-OCEL serialization. This can be verified using a JSON validator (as example, `https://www.jsonschemavalidator.net/`).

The following constraints are checked by the JSON schema:

- The JSON serialization should contain a dictionary of events (inside the *ocel:events* key) and a dictionary of objects (inside the *ocel:objects* key).
- Each event should have:
  - An identifier (*ocel:id*) of type string.
  - An activity (*ocel:activity*) of type string.
  - A timestamp (*ocel:timestamp*) of type date.
  - An attribute map (*ocel:vmap*), that can be possibly empty.
- Each object should have:
  - An identifier (*ocel:id*) of type string.
  - An object type (*ocel:type*) of type string.

– An attribute map (*ocel:ovmap*), that can be possibly empty.

Listing 4: JSON schema for JSON-OCEL

```
{
	"$schema": "http://json-schema.org/schema#",
	"additionalProperties": true,
	"definitions": {
		"AttributeBooleanType": {
			"type": "boolean"
		},
		"AttributeDateType": {
			"type": "string",
			"format": "date-time"
		},
		"AttributeFloatType": {
			"type": "number"
		},
		"AttributeIntType": {
			"type": "integer"
		},
		"AttributeStringType": {
			"type": "string"
		},
		"ObjectType": {
			"type": "string"
		},
		"ObjectMappingType": {
			"type": "object"
		},
		"ValueMappingType": {
			"type": "object"
		},
		"EventType": {
			"properties": {
				"ocel:id": {"$ref": "#/definitions/AttributeStringType" },
				"ocel:activity": {"$ref": "#/definitions/AttributeStringType" },
				"ocel:timestamp": {"$ref": "#/definitions/AttributeDateType" },
				"ocel:vmap": {
					"items": {
						"$ref": "#/definitions/ValueMappingType"
					},
					"type": "object"
				},
				"ocel:omap": {"type": "array" }
			},
			"required": [
				"ocel:id", "ocel:activity", "ocel:timestamp", "ocel:omap", "ocel:vmap"
			],
			"type": "object"
		},
		"ObjectType": {
			"properties": {
				"ocel:id": {"$ref": "#/definitions/AttributeStringType" },
				"ocel:type": {"$ref": "#/definitions/AttributeStringType" },
				"ocel:ovmap": {
					"items": {
						"$ref": "#/definitions/ValueMappingType"
					},
					"type": "object"
				}
			},
```

```
                        "required": [
                                "ocel:id", "ocel:type", "ocel:ovmap"
                        ],
                        "type": "object"
                }
        },
        "description": "Schema for the JSON-OCEL implementation",
        "properties": {
                "ocel:events": {
                        "items": {
                                "$ref": "#/definitions/EventType"
                        },
                        "type": "object"
                },
                "ocel:objects": {
                        "items": {
                                "$ref": "#/definitions/ObjectMappingType"
                        },
                        "type": "object"
                }
        },
        "type": "object"
}
```