From OCEL to DOCEL – Datasets and Automated Transformation

Alexandre Goossens[‡1], Adrian Rebmann[‡2] Johannes De Smedt[1], Jan Vanthienen[1] and Han van der Aa[2]

[1] Leuven Institute for Research on Information Systems (LIRIS), KU Leuven
{FirstName}.{LastName}@kuleuven.be
[2] Data and Web Science Group, University of Mannheim, Germany
{rebmann|han.van.der.aa}@uni-mannheim.de

Abstract. Object-centric event data represent processes from the point of view of all the involved object types. This perspective has gained interest in recent years as it supports the analysis of processes that previously could not be adequately captured, due to the lack of a clear case notion as well as an increasing amount of output data that needs to be stored. Although publicly available event logs are crucial artifacts for researchers to develop and evaluate novel process mining techniques, the currently available object-centric event logs have limitations in this regard. Specifically, they mainly focus on control-flow and rarely contain objects with attributes that change over time, even though this is not realistic, as the attribute values of objects can be altered during their lifecycle. This paper addresses this gap by providing two means of establishing object-centric datasets with dynamically evolving attributes. First, we provide event log generators, which allow researchers to generate customized, artificial logs with dynamic attributes in the recently proposed DOCEL format. Second, we propose and evaluate an algorithm to convert OCEL logs into DOCEL logs, which involves the detection of event attributes that capture evolving object information and the creation of dynamic attributes from these. Through these contributions, this paper supports the advancement of object-centric process analysis by providing researchers with new means to obtain relevant data to use during the development of new techniques.

Keywords: Object-centric processes · OCEL · DOCEL · Log Generator.

1 Introduction

Organizations often operate in complex environments with multiple objects interacting and participating in the same business process. To capture these different perspectives, the concept of object-centric processes has been proposed, in which multiple object types participate over the course of a business process [1]. In recent years, object-centric process mining has gained increasing interest in the research community with the introduction of novel event log formats such as eXtensible Object-Centric (XOC) logs [16], Object-Centric Behavioral Constraint (OCBC) models [2], and Object-Centric Event Logs (OCEL) [13]. Especially, OCEL is currently the most used object-centric event

[‡] Joint first authors

log format with its own evaluation metrics [3], visualization tool [11] and various analysis techniques [4, 5].

However, OCEL has some limitations by design. Particularly, the relationships between objects and attributes are not strictly defined and attributes that can change in value over time are challenging to deal with in OCEL [14]. This makes the analysis of attribute change difficult, because it is not always clear which event or object manipulated the attribute value, e.g. whether an update event changed the quantity of an order or the price of a product. To overcome this, there have been efforts to establish a new format for object-centric event data such as Data-aware Object-Centric Event Logs (DOCEL) [14]. The main aspect DOCEL introduced is the notion of static and dynamic attributes [14]. Static attributes are attributes that do not change over the course of a business process and can either be linked to an event or to an object. Conversely, dynamic attributes are attributes that can change over the course of a business process and are linked to both an object and to an event with the use of foreign keys. As of now, no general consensus has been reached, however, regarding the exact set of entities and relationships a meta model of such a data format shall have. Currently, the Object-Centric Event Data (OCED) is under development and will surely take into account the explicit representation of object evolution and their attributes.[1]

However, despite the benefits of moving to object-centric data with dynamic attributes, no datasets are available that can actually be leveraged by researchers for the development of process mining techniques that account for such attributes. This paper addresses this issue in two ways:

1. We propose two process-specific log generators to create customized object-centric event logs in the DOCEL format [14], which also generate dynamic object attributes, i.e., they create objects with attributes that change over time.
2. We propose an algorithm that takes an existing OCEL event log and automatically transforms it into an event log in the DOCEL format. Combined with earlier work, this algorithm can also be used to transform flat XES [15] logs into DOCEL logs.

The remainder of this paper is structured as follows: Section 2 motivates the need for data sets with dynamic object attributes. Section 3 presents our log generators to create DOCEL logs. Section 4 then describes our algorithm to transform OCEL into DOCEL logs. Section 5 uses data sets created using our log generators to evaluate the algorithm and shows how even real-life event data captured in XES format can be transformed into DOCEL and thus be used for research on object-centric process mining. Finally, Section 6 reflects on related work and Section 7 concludes the paper.

2 Motivation

Over the course of the execution of a business process, the objects involved in its execution may undergo changes, such as creating, updating or deleting, which results in attribute values changes throughout an object's lifecycle. The eXtensible Event Stream (XES) event log format, with its traditional case-based view of processes, addresses this issue by directly linking attributes to the events that manipulate them [15]. Because

[1] https://www.tf-pm.org/resources/oced-standard

each event belongs to exactly one case, it is unambiguously clear which attribute was manipulated by a specific event as well as to which trace it belongs.

However, in object-centric processes, events may refer to any number of objects [4]. In OCEL, this is implemented using an event table storing all attributes that are manipulated by events and an objects table storing all objects and their (static) attributes. Unfortunately, with OCEL it is not possible to uniquely identify to which object a dynamic attribute belongs, neither through its events table nor its objects table [14]. To illustrate this, consider the event table of an OCEL log shown in Table 1 and its objects table in Table 2, which cover the (simplified) handling of two orders. First, an order is created, then the ordered items are picked, before they are sent. Between creating an order and sending items, orders can be updated, i.e., items may be added or removed. Besides the common `EventID`, `Activity`, and `Timestamp` attributes and the references to the instances of different object types associated with each event, the log contains an additional event attribute, `Value`. While this attribute is associated with events, it is clear that it actually refers to objects. Specifically, it refers to the current value of the order the event is associated with. This is left implicit by the OCEL format making it impossible for automated process analysis techniques to properly leverage this information. For instance, by looking at event e_6, it is unclear whether the *Value* refers to the current value of the order or the value of the item that is removed. This makes analyzing attribute changes in object-centric business processes difficult using OCEL.

ID	Activity	Timestamp	Orders	Items	Value
e_1	Create order	05-20 09:07	$\{o_1\}$	$\{i_1, i_2\}$	*100*
e_2	Pick items	05-23 14:20	$\{o_1\}$	$\{i_1, i_2\}$	*100*
e_3	Create order	06-03 19:17	$\{o_2\}$	$\{i_3\}$	*60*
e_4	Pick items	06-04 15:20	$\{o_2\}$	$\{i_3\}$	*60*
e_5	Update order	06-04 18:11	$\{o_1\}$	$\{i_1\}$	*70*
e_6	Remove item	06-05 11:48	$\{o_1\}$	$\{i_2\}$	*70*

Table 1: Events of an OCEL log.

Type	Instances
Orders	$\{o_1, o_2\}$
Items	$\{i_1$(`Weight: 24`), i_2(`Weight: 99`), i_3(`Weight: 10`)$\}$

Table 2: Objects of the log

The DOCEL format addresses this issue with the notion of static and dynamic attributes. Static attributes do not change over the course of a business process, whereas dynamic object attributes can change value over the course of a process and are linked to both an event and an object. For instance, consider the running example, where orders have a `Value` attribute. If an order changes during its lifecycle, i.e., between its first and last occurrence in the execution of a business process, e.g., because items are added later on, the value attribute changes. DOCEL explicitly links this change to the order itself and the event that caused it. Unfortunately, despite the increased interest in object-centric processes, there are currently no event logs available describing processes with dynamic attributes nor are there tools available to generate such object-centric logs.

3 DOCEL Dataset Generators

This section introduces the DOCEL log generators for two artificial, yet realistic processes. First, we introduce the two processes that are simulated by the log generators.

Table 3: Object types order-to-delivery process

Object Type	Static Attributes	Dynamic Attributes
Customer	Name, Bank Account	Customer Address
Order	/	Weight, Order Price
Product Type	Product Name, Price, Weight	/
Item	Price, Weight	/
Packages	Price, Weight	/

Table 4: Object types shipping-method process

Object Type	Static Attributes	Dynamic Attributes
Customers	Name, Bank Account	/
Product Type	Value, Fragile	/
Orders	Quantity	Value, Refund, Shipping Method

These processes contain dynamic attributes as well as various AND and OR-gateways, and loops, which are also found in real-life processes. Second, we explain the log generators and their tunable parameters.

3.1 Process descriptions

In this section, we describe the processes that serve as a basis for our log generators. The process models can be found alongside the Python notebooks that implement the generators[2]. In this paper, we limit ourselves to a textual description for space reasons.

Order-to-delivery process The order-to-delivery process[3] contains 5 object types and various attributes that we summarize in Table 3.

The process starts with the customer adding items to an order which increases the value and the weight of an order. Each item is of a certain product type and inherits its weight and price. Once the order is placed, the items are picked. Before paying the order, the customer is still allowed to remove items from an order. If that is the case, the items are removed and the value and weight of an order are updated. Once the order is paid, the package is created with a weight and price and sent out. However, a customer might change their delivery address in which case the delivery fails and has to be re-executed. Moreover, every delivery has a very small chance of failing due to unforeseen circumstances. Once the package is successfully delivered, the process ends.

Shipping-method process The shipping-method process covers orders that have different shipping modes depending on various factors. It is based on the process described in [8] and contains three object types whose attributes are summarized in Table 4.

A customer places an order with a quantity for each product type. This is then received by the company which has to manually confirm the purchase at which point the value of the order is also determined. Once the purchase is confirmed, the products are retrieved from the warehouse and added to the package. In case, the product type

[2] Available at `https://github.com/a-rebmann/ocel_to_docel/tree/main/notebooks`

[3] This process is based on the running example OCEL log available on `ocel-standard.org`

is fragile (indicated with a binary value), the product is first wrapped with some protection. Next, the customer has to confirm the shipping information, after which the shipping method is determined. If the package contains a fragile product or its value exceeds a certain amount, the package is shipped with a courier. Otherwise the package is shipped by mail. Simultaneously, an invoice is sent to the customer. Once the package has arrived, the customer determines their satisfaction. If the customer is satisfied the process is ended after the order has been filed. In case, the customer is dissatisfied, the customer requests a refund, setting the binary refund value to 1. Next, the customer confirms the shipping information, which will be handled using an express courier. The company sends a recollect letter after which the customer returns the package. Once the package is returned, the company refunds the customer. Once the package has arrived, the customer determines their satisfaction (assumed to be positive here) after which the order is filed and the process ends.

3.2 Log Generators

To allow the research community to generate as many different logs as desired for different research needs, we propose log generators that allow users to change various parameters that influence the generated logs, as summarized in Tables 5 and 6. In both log generators, it is possible to change the amount of object instances of all the object types involved in the process as well as changing the time between events and initial time frame of a process. Beyond that, it is possible to change various probabilities of events happening in the process such as removing an item or asking for a refund. Both log generators can generate DOCEL logs in a spreadsheet format, which is intuitive to understand because every table (events, objects and dynamic attribute tables) can be stored in a separate sheet. Note that, because we provide the generators as Python notebooks, it is possible to change the logs beyond the options described in the tables with some minimal Python coding.

Finally, to be able to evaluate our automated OCEL-to-DOCEL transformation algorithm (cf .Section 4), we also included the option to create an OCEL log for a generated

Table 5: Summary of functionality in the order-to-delivery Log Generator

Tunable parameter	**Description of Functionality**
Customer Addresses and Names	Randomly generated for each log entry along with randomly generated bank account details.
Products	Taken from a fixed list directly obtained from the OCEL log of X.
Start Timeframe	Allows defining the start time for the process.
Time Between Events	Allows adjusting the time interval between consecutive events.
Number of Orders	Can be changed to generate logs with varying numbers of orders.
Max Number of Products	Can be adjusted to set a maximum limit for the number of products in an order.
Max Number of Items	Can be modified to set a maximum limit for the total number of items in an order.
Probability of Removing an Item	Can be adjusted to control the likelihood of an item being removed from an order. The higher the value, the more likely an item is removed.
Probability of Changing an Address	Allows changing the probability of an address being modified in the log entries. The higher the probability, the more likely the address will be changed.
Probability of Failing a Delivery	Can be altered to control the likelihood of a delivery failure occurrence. The higher the probability, the more likely the probability will be changed.

Table 6: Summary of functionality in the shipping method log generator

Tunable parameters	**Description of Functionality**
Number of Products	Allows adding new products with custom values for the price and the possibility of fragility.
Number of Customers	Enables creating new customers with randomly generated names and bank accounts.
Lists of People Executing Activities	Allows adapting the lists of people who execute the company's activities.
Start Timeframe	Allows defining the start time for the process.
Time interval Between Events	Allows adjusting the time interval between consecutive events.
Order Value Threshold	Can be changed to determine the shipping method based on the order's value.
Number of Orders	Can be changed to generate logs with varying numbers of orders.
Probability of Refund	Can be adapted to determine the likelihood of a refund, indicating customer satisfaction with the purchase. The higher the value the more satisfied the customer is.

DOCEL log. If OCEL logs are created, the dynamic attributes are directly linked to the events, therefore, losing the clear information on object-attribute allocations.

4 Transforming OCEL to DOCEL

This section presents our proposed algorithm to transform an OCEL event log into a DOCEL formatted one, achieved by detecting dynamic object attributes and assigning them to the appropriate object instances. Our algorithm takes as input an OCEL formatted event log L, which comprises a set of events recorded by an information system. Each event $e \in L$ is a tuple $e = (eid, act, ts, OI, AV)$, with eid the event's id, act its activity, ts its timestamp, OI a set of object instances, and AI a set of attribute-value pairs. Each object instance $oi \in OI$ is a tuple (oid, $type$), where oid is the instance's identifier and $type$ its type, whereas each attribute-value pair $(a, v) \in AI$ relates an attribute value v to an attribute name a. We denote the set of object types that occur in L as $\mathcal{O}_T^L$ and the set of event attribute names as $\mathcal{A}_T^L$.

As visualized in Fig. 1, our algorithm applies two steps to transform an OCEL log L into a DOCEL formatted log L'. Step 1, *Dynamic object attribute detection*, aims to detect dynamic object attributes in the event attributes of the input log and match them with the object type they refer to. Based on that matching, Step 2, *Dynamic-object-attribute-to-object assignment*, creates object attributes and associates these with the individual object instances they relate to, establishing a DOCEL log that makes these relationships explicit. In the remainder of this section, we describe these steps in detail.

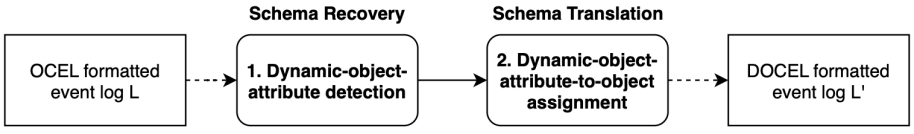

Fig. 1: Algorithm overview.

4.1 Dynamic object attribute detection

This step aims to determine whether an attribute $a \in \mathcal{A}_T^L$ is a dynamic object attribute of an object type $t \in \mathcal{O}_T^L$, resulting in a set of attribute-object-type matches M.

For this step, recall that dynamic object attributes are attached to events in the OCEL format, even though they capture information about an object associated with an event rather than relate to the event itself. For instance, although an *Update order* event has a `Value` attribute in Table 1, this attribute actually captures the (new) value of the order, not of the event.

Identifying candidate object types. For each attribute $a \in \mathcal{A}_T^L$, our algorithm first identifies a set of candidate object types $O_a \subseteq \mathcal{O}_T^L$. To determine if an object type $t \in \mathcal{O}_T^L$ is a candidate type for attribute a, our algorithm checks if it meets two requirements:

- *Object-attribute co-occurrence:* First, our algorithm checks if every occurrence of attribute a can be related to exactly one instance of object type t. This means that L cannot contain an event e for which $(a, v) \in e.AV$ and for which $e.OI$ either does not contain any object instance of type t or contains multiple of them. For instance, for the running example type *item* is not a candidate for attribute `Value`, since events e_1 and e_2 each refer to two items, but only contain a single `Value`.
- *Observed attribute changes:* Second, our algorithm checks if attribute a is actually dynamic (with respect to type t), i.e., if its value is observed to change during an object's lifecycle. For this, our algorithm checks if L contains (at least) two events, e and e', which both contain the same object instance of type t, but are associated with different values for attribute a.

Any object type t that passes both checks is added to the set of candidate object types O_a. If, after checking all object types in $\mathcal{O}_T^L$, O_a contains exactly one candidate type t, the match (a, t) is added to M. Instead, if O_a does not contain any candidates, then a is not a dynamic attribute, whereas, if O_a contains more than one candidate, our algorithm turns to the disambiguation procedure described next.

Disambiguating candidate types. An attribute a can have multiple candidate object types in O_a if certain object types always occur together for events. For example, if every event in a log L is associated with both a *customer* and an *order*, an attribute such as `Value` would have both of these types as a candidate, since each occurrence of `Value` can be associated with exactly one *customer* and one *order*.

To be able to match a to a single object type $t \in O_A$ in such cases, our algorithm employs two disambiguation strategies:

- *Relation-based selection:* Our algorithm first checks for 1:N relationships between object types in O_a, aiming to assign a to a more fine-granular candidate object type. For illustration, consider a `Refund` attribute and two object types, *customer* and *order* in $O_{\texttt{Refund}}$. All events with a value for `Refund` refer to one customer and one order. Matching `Refund` to *customer* would obfuscate the relation between single orders and their `Refund` if a customer places multiple orders. Therefore, the algorithm tries to identify cases for each $t \in O_a$, where for two events $e_1, e_2 \in L$ an instance oi_t of t occurs together with two different instances $oi_{t'_1}$ and $oi_{t'_2}$ of another type $t' \in O_a$ and, if so, removes t from the candidates. If a single candidate t remains, the match (a, t) is added to M.
- *Name-based selection:* If, after the relation-based selection, O_a still contains more than one candidate, our algorithm finds the most similar object type among the candidates based on a's name. For this, our algorithm quantifies the semantic similarity $\texttt{sim}(a, t) \in [0, 1]$ between a and each type $t \in O_a$ by computing the cosine sim-

ilarity between sentence embeddings, obtained from a pretrained *Sentence Transformer* [18]. These embeddings are specifically designed to capture semantically meaningful representations on the level of (short) sentences rather than individual words, making them highly suitable for our purpose.
Given these similarity scores, our algorithm determines if there is a type $t \in O_a$ for which the similarity score is distinctly higher (according to a threshold τ, set to 0.1 by default) than the scores of the other types, i.e., if $\texttt{sim}(a, t) > \texttt{sim}(a, t') + \tau$ for each $t' \in O_a \setminus \{t\}$. If such a distinctively similar type t is found, the match (a, t) is added to M, otherwise a is not matched to any object type.

The set of attribute-object-type matches M then serves as input to the next step.

4.2 Dynamic-object-attribute-to-object assignment

Based on each match $(a, t) \in M$, Step 2 creates an object attribute $a_{(e,oi)} = (vid, eid, oid, a)$ for each event e that changes oi's value for a. The algorithm first assign $a_{(e,oi)}$ a unique identifier $a_{(e,oi)}.vid$. Then it sets its attribute-value pair, $a_{(e,oi)}.a = (a, v)$ with $(a, v) \in e.AI$, its event identifier, $a_{(e,oi)}.eid = e.eid$, and its object instance identifier, $a_{(e,oi)}.oid = oi.id$ with $oi \in e.OI \wedge oi.type = t$. For the attribute `Value` of our running example, this creates the object attribute table visualized in Table 7.

Finally, the algorithm removes the attributes for which a match was found from the events' set of attribute-value pairs and returns the established DOCEL log. For the example, it would, therefore, remove the `Value` attribute from the events in Table 1 and return the resulting event table, the unchanged objects table, and the newly created object attribute table (Table 7).

Table 7: `Value` attribute table.

ValueID	OrderID	EventID	Value
v1	o1	e1	100
v2	o2	e3	60
v3	o1	e5	70

5 Evaluation

We implemented our algorithm in Python and performed evaluation experiments to assess our algorithm's capability to accurately transform OCEL logs into DOCEL logs (Section 5.1). Afterwards, we show how it can be used in combination with an existing approach to also transform XES event logs into DOCEL logs (Section 5.2). The implementation, evaluation data, and generated DOCEL logs are available in our repository[4].

5.1 Experiments

We assess whether our algorithm is able to correctly detect the dynamic attributes in an OCEL log and transform it into a DOCEL log.

Datasets. We use OCEL and DOCEL event logs generated using our log generators (cf. Section 3) by simulating the process execution for 100 orders per scenario. The OCEL logs serve as input to our algorithm, whereas the DOCEL logs serve as a gold standard,

Table 8: Characteristics of the OCEL logs used for the evaluation

ID	# Events	Objects	# Event att.	# Dyn. att.
Order to Delivery	6,014	Customer (44), Order (100), Item (3,559), Packages (100), Product Type (20)	7	3
Shipping Method	2,036	Customer (50), Product Type (3), Order (100)	5	3

i.e., the correct transformation result. The OCEL logs obtained in this manner differ in their number of events, objects, object types, and attributes as shown in Table 8.

Setup. To assess the ability of our algorithm to correctly detect dynamic object attributes in the OCEL event logs, we conduct experiments using two settings:
(1) Original attribute names. In this setting, we use all information from the event log as input to our approach.
(2) Hidden attribute names. To assess the robustness of our algorithm, we reduce the available information by hiding event attribute names in the OCEL logs. This allows us to assess the dependency of our algorithm on its name-based check (Section 4.1).

We measure the performance in terms of precision, recall, and F_1-score with respect to the dynamic object attributes in the original DOCEL logs. Using tp to denote the dynamic attributes correctly matched to an object type fp for the dynamic attributes incorrectly matched to an object type, and fn for the dynamic attributes that were wrongly not matched to an object type, we then quantify the precision as $tp/(tp+fp)$, the recall as $tp/(tp+fn)$, and F_1-score as the harmonic mean of precision and recall.

Results. We first report on the results of attribute-to-object-type matching, which is the most challenging part, before we report on the attribute value to object assignment.
Attribute-to-object-type matching. Table 9 reports on the results of the attribute-to-object-type matching per event log.

Table 9: Results of the attribute-to-object-type matching per OCEL log.

Log	Original Attribute Names				Hidden Attribute Names			
	Count	Precision	Recall	F_1	Count	Precision	Recall	F_1
Order to delivery	3	1.00	1.00	1.00	3	1.00	0.75	0.86
Shipping method	4	0.80	1.00	0.89	4	0.80	1.00	0.89
Average	3.5	0.90	1.00	0.95	3.5	0.90	0.88	0.88

We find that our approach achieves a perfect recall and good precision (0.9) in detecting dynamic object attributes and associating these with the correct object type, when all original information from the input log is available. An in-depth look shows that the only error made is the assignment of the *Resource* attribute of shipping-method process to the *order* type. While unconventional, this assignment is not necessarily problematic, since indeed each time a resource executes a process step, that step relates to a specific order. Such incorrect assignments could be easily avoided by specifying a set of names reserved for event attributes, e.g., *org:resource* or *org:role* as done in XES

[4] `https://github.com/a-rebmann/ocel_to_docel`

logs [15]. The importance of the duplicate-resolution strategy based on object lifecycles becomes clear for the shipping-method process, where *customer* and *order* are in 1:N relation. Without it, e.g., the `Refund` attribute would be matched to both *customer* and *order*. This strategy resolves this, correctly matching `Refund` only to the *order*. Note that the name-based matching strategy could not resolve this, because *refund* is semantically similar to both *customer* and *order*.

When hiding the original attribute names, we find that for shipping-method process the performance remains the same, whereas for order to delivery it drops achieving an F_1-score of 0.86 compared to 1.00 with original attribute names. The reason for this is the missing match of the `Customer Address` attribute to the *customer* object type, because the candidates *order* and *customer* could not be resolved solely using the relation-based disambiguation in this case.

Object-attribute-to-object assignment. We also report on the results of Step 2, i.e., the assignment of object-attributes to objects, for the setting with original attribute names. We provide results for (1) when propagating false positives from Step 1 and (2) when only including object attributes that were correctly matched to object types in Table 10.

Table 10: Results of the dynamic-object-attribute-to-object assignment per OCEL log for the setting with original attribute names, with and without propagation of false positives (*fp*s) from Step 1.

Log	**When propagating *fp*s**				**Without propagating *fp*s**			
	Count	**Precision**	**Recall**	**F_1**	**Count**	**Precision**	**Recall**	**F_1**
Order to delivery	3,710	1.00	1.00	1.00	3,710	1.00	1.00	1.00
Shipping method	1,886	0.23	1.00	0.37	442	1.00	1.00	1.00
Average	2,798	0.62	1.00	0.69	1,875	1.00	1.00	1.00

We find that when propagating false positives (*fp*s) from Step 1, we achieve an average F_1-score of 0.69, a precision of 0.37, and a perfect recall (1.00). The lower precision is caused only by the assignment of the `Resource` attribute to the *order* object type for the shipping-method event log. Given that there are many more resource handovers involved than value changes to the actual object attributes this has a considerable impact on the overall performance scores. Apart from this, we achieve perfect scores, though. This can also be seen when disregarding false positives from Step 1. In this case, we achieve perfect scores for the assignment of object attributes to objects.

5.2 Application: From XES to DOCEL

Recently an approach was proposed to uncover object-centric data in a flat event log, e.g., in XES format, to automatically transform it into a log OCEL format [17]. Given the limitations of OCEL, we aim to combine our algorithm with this approach to investigate whether XES logs can be transformed into DOCEL.

To this end, we use the BPI Challenge 2017 log [9], which captures the loan application process at a financial institute and involves two main types of objects: applications and offers. For a single application, multiple offers can be made, where at some point an

offer may be accepted. To make sure we have a dynamic object attribute in this log, we add an `OfferAccepted` attribute to each event, which is set to `false` when a new application is created and changes to `true`, when an offer associated with the application is accepted. The goal is to check if our algorithm can identify this attribute correctly as a dynamic object attribute associated with the *application* object type creating a DOCEL log from the output of the XES-to-OCEL approach.

When applying our algorithm to the transformed XES log, we find that it indeed correctly matched the `OfferAccepted` attribute to the application. Based on that it removed the corresponding event attributes and created correct object attributes linked to both the correct application and the event that writes the value. Beyond detecting the derived attribute correctly, the algorithm also detected the `EventOrigin` and the `Action` attributes as dynamic object attributes of the *application* object type. After inspecting the attribute values, the latter of these matches makes sense, because `Action` captures the status of the application, which changes throughout its lifecycle. `EventOrigin`, as its name indicates, captures the origin of the event and was, therefore, falsely associated with *application*. Nevertheless, this outcome shows the potential of our algorithm to help generate a broader variety of object-centric event logs that consider evolving objects based on available event data. The DOCEL log our algorithm created can be found in our repository linked on Page 8.

6 Related Work

The first proposed object-centric event log formats are XOC logs [16] together with OCBC models [2]. These proposals have scalability issues because of the duplication of attributes and object relations with each executed event. Next, OCEL logs were introduced which do not have such scalability issues and are currently the most used object-centric event log format [13] with a lot of dedicated research and tools such as a visualization tool [11], fitness and precision metrics [3], clustering analysis [12], or predictive object-centric process analysis [10].

The conversion of XES logs to OCEL logs has been investigated in [17]. Next to that a generic approach to extract OCEL logs from SAP systems and relational databases has also been researched in respectively [7] and [6]. Finally, the extraction of OCEL logs from virtual knowledge graphs was researched in [19].

7 Conclusion

This paper offers two problem-specific log generators and a transformation algorithm to the research community. The two paramaterizable log generators support both the widely used OCEL format and the more recent DOCEL format, which allows for dynamic attributes and consistent object-attribute allocation. To further support the creation of DOCEL logs, we also proposed an algorithm to convert OCEL logs to DOCEL logs. This algorithm not only identifies the presence of dynamic attributes, which are better represented in the DOCEL format, but also links the attributes to the correct object types. Our evaluation shows that the algorithm can accurately transform OCEL into

DOCEL logs and that, in combination with previous work, it can even be used to convert XES logs to DOCEL logs. However, it cannot deal with all situations. For instance, if multiple instances of the same object type are changed by one event (batching), this cannot be handled if these instance have not previously been changed individually.

In the future, we aim to address this limitation. We also plan to develop a comprehensive user interface tool to enhance the user experience. Furthermore, we aim to transform additional XES logs that would benefit from a conversion to DOCEL logs.

References

1. van der Aalst, W.M., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A framework for lightweight interacting workflow processes. International Journal of Cooperative Information Systems **10**(04), 443–481 (2001)
2. van der Aalst, W.M., Li, G., Montali, M.: Object-centric behavioral constraints. arXiv preprint arXiv:1703.05740 (2017)
3. Adams, J.N., van der Aalst, W.: Precision and fitness in object-centric process mining. In: 2021 3rd International Conference on Process Mining (ICPM). pp. 128–135. IEEE (2021)
4. Adams, J.N., Schuster, D., Schmitz, S., Schuh, G., van der Aalst, W.M.: Defining cases and variants for object-centric event data. arXiv preprint arXiv:2208.03235 (2022)
5. Berti, A.: Filtering and sampling object-centric event logs. arXiv preprint arXiv:2205.01428 (2022)
6. Berti, A., Park, G., Rafiei, M., van der Aalst, W.: A generic approach to extract object-centric event data from relational databases (2023)
7. Berti, A., Park, G., Rafiei, M., van der Aalst, W.M.: A generic approach to extract object-centric event data from databases supporting SAP ERP. Journal of Intelligent Information Systems pp. 1–23 (2023)
8. De Smedt, J., Hasić, F., Vanthienen, J.: Towards a holistic discovery of decisions in process-aware information systems. In: Business Process Management. LNBIP, Springer (2017)
9. van Dongen, B.: BPI Challenge (2017). https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b
10. Galanti, R., de Leoni, M., Navarin, N., Marazzi, A.: Object-centric process predictive analytics. arXiv preprint arXiv:2203.02801 (2022)
11. Ghahfarokhi, A.F., van der Aalst, W.: A python tool for object-centric process mining comparison. arXiv preprint arXiv:2202.05709 (2022)
12. Ghahfarokhi, A.F., Akoochekian, F., Zandkarimi, F., van der Aalst, W.M.: Clustering object-centric event logs. arXiv preprint arXiv:2207.12764 (2022)
13. Ghahfarokhi, A.F., Park, G., Berti, A., van der Aalst, W.M.: OCEL: A standard for object-centric event logs. In: ADBIS. pp. 169–175. Springer (2021)
14. Goossens, A., De Smedt, J., Vanthienen, J., van der Aalst, W.: Enhancing data-awareness of object-centric event logs. In: ICPM Workshops (2022)
15. Günther, C.W., Verbeek, H.M.W.: XES standard definition. IEEE Std (2014)
16. Li, G., Murillas, E.G.L.d., Carvalho, R.M.d., van der Aalst, W.: Extracting object-centric event logs to support process mining on databases. In: CAiSE. pp. 182–199. Springer (2018)
17. Rebmann, A., Rehse, J.R., van der Aa, H.: Uncovering object-centric data in classical event logs for the automated transformation from XES to OCEL. In: BPM. pp. 11–16 (2022)
18. Reimers, N., Gurevych, I.: Sentence-bert: Sentence embeddings using siamese bert-networks. In: EMNLP. ACL (11 2019)
19. Xiong, J., Xiao, G., Kalayci, T.E., Montali, M., Gu, Z., Calvanese, D.: Extraction of object-centric event logs through virtual knowledge graphs. In: 35th International Workshop on Description Logics, DL 2022, Haifa, Israel, August 7-10, 2022 (2022)