Defining Cases and Variants for Object-Centric Event Data

Jan Niklas Adams
Chair for Process and Data Science (PADS)
RWTH Aachen University, Aachen, Germany
niklas.adams@pads.rwth-aachen.de

Daniel Schuster
Institute for Applied Information Technology (FIT)
Fraunhofer, Sankt Augustin, Germany
daniel.schuster@fit.fraunhofer.de

Seth Schmitz, Günther Schuh
Laboratory for Machine Tools (WZL)
RWTH Aachen University, Aachen, Germany
{s.schmitz,g.schuh}@wzl.rwth-aachen.de

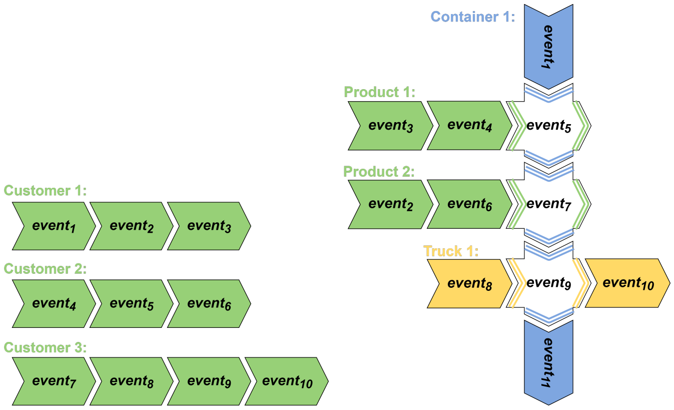
Wil M.P. van der Aalst
Chair for Process and Data Science (PADS)
RWTH Aachen University, Aachen, Germany
wvdaalst@pads.rwth-aachen.de

***Abstract*—The execution of processes leaves traces of event data in information systems. These event data can be analyzed through process mining techniques. For traditional process mining techniques, one has to associate each event with exactly one object, e.g., the company's customer. Events related to one object form an event sequence called a case. A case describes an end-to-end run through a process. The cases contained in event data can be used to discover a process model, detect frequent bottlenecks, or learn predictive models. However, events encountered in real-life information systems, e.g., ERP systems, can often be associated with multiple objects. The traditional sequential case concept falls short of these so-called object-centric event data since these data exhibit a graph structure. One might force object-centric event data into the traditional case concept by flattening it. However, flattening manipulates the data and removes information. Therefore, a concept analogous to the case concept of traditional event logs is necessary to enable the application of different process mining tasks on object-centric event data. In this paper, we introduce the case concept for object-centric process mining: process executions. These are graph-based generalizations of cases as considered in traditional process mining. Furthermore, we provide techniques to extract process executions. Based on these executions, we determine equivalent process behavior with respect to an attribute using graph isomorphism. Equivalent process executions with respect to the event's activity are object-centric variants, i.e., a generalization of variants in traditional process mining. We provide a visualization technique for object-centric variants. The contribution's scalability and efficiency are extensively evaluated. Furthermore, we provide a case study showing the most frequent object-centric variants of a real-life event log. Our contributions might be used as a basis to adapt traditional process mining techniques by researchers and to generate initial control-flow insights into object-centric event logs by practitioners.**

***Index Terms*—Object-centric process mining, variants**

I. Introduction

Process mining [1] is an umbrella term for techniques discovering knowledge about processes from *event data* that these processes generated. These event data come as an *event log*. Each event of an event log describes an activity executed in the process, together with its associated data.



(a) Traditional event logs. **(b)** Object-centric event logs.

Fig. 1: Traditional event logs take the form of (a): Each event is associated to exactly one object, called the case. Object-centric event logs (b) drop this restriction, events can be associated to multiple objects of different types.

One of the most fundamental concepts in process mining are *cases*. A case is an event sequence describing one observed end-to-end behavior in a process. Each event is associated to the *object* for which the event was conducted, e.g., the customer. This object is called the case of the event and each object can be associated to multiple events, forming the event sequence of end-to-end behavior. The event sequences of all objects are the fundamental starting point of many process mining algorithms: control-flow visualization [2], i.e., frequent sequences of conducted actions, bottleneck analysis [3], or outcome prediction [4]. An event log in traditional process mining exhibits the structure depicted in Fig. 1a: a collection of event sequences, one for each case.

Object-centric event data [5] is a generalized form of traditional event data. We drop the assumption that one event can only be related to one object. Each object is still associated to a sequence of events, but one event may belong to multiple

The authors would like to thank the Marga und Walter Boll-Stiftung for the kind support within the research project.

sequences. An example of the resulting structure of object-centric event data is depicted in Fig. 1b. Some events are shared between objects, e.g., $event_5$. The sequential structure of the event data is lost by dropping the assumption of single-object association. *The event data takes a graph structure.* Furthermore, objects are associated to different *object types*. An information system may record data for the conducted actions of objects of different types, e.g., products and delivery objects in an ordering process. In traditional process mining, all objects are assumed to be of the same type, e.g., all events describe an action conducted for a customer, leading to homogeneous events. In object-centric event logs, the homogeneity assumption is dropped, leading to differently typed objects and events. In conclusion, traditional event data describe homogeneously typed event sequences, while object-centric event data describe heterogeneously typed event graphs. Object-centric event data is closer to the reality experienced in many real-life information systems: It is encountered in production processes [6], high volume manufacturing data [7], and order-to-cash processes [8]. Object-centric event data exhibiting a graph structure was already observed in [9], [10]

Most process mining techniques rely on the existence of cases. To apply these techniques to object-centric event logs, the case concept and object-centric event logs must be connected. There are two ways to bridge the gap between the case concept and object-centric event logs: Either moving object-centric event logs to the traditional case concept or moving the traditional case concept to object-centric event logs. First, one can force object-centric event data into traditional event log format, enforcing homogeneity and sequentiality. This is called *flattening* [11]. Second, one can generalize the concept of cases from homogeneous sequences to heterogeneous graphs and adapt process mining techniques accordingly. When flattening an object-centric event log, one chooses an object type (also: case notion), removes events with no objects of this type, and duplicates events with multiple objects of that type. The output of flattening is a traditional event log. While flattening is fast and straightforward, it manipulates the event data: information about diversity in object types is discarded and a sequential structure removes dependencies contained in the event data [8], [11]. For these reasons, we aim to move the case concept towards object-centric event data in this paper. We provide the following contributions:

C1 We generalize the case concept of traditional process mining from homogeneous sequences to heterogeneous graphs for object-centric event logs. These are called process executions.

C2 We provide a general approach for extracting process executions from an object-centric event log. Furthermore, we provide two specific extraction techniques.

C3 We use graph isomorphism to determine equivalent process executions. Isomorphic graphs can be used to group equivalent behavior, e.g., object-centric variants for equivalent control-flow behavior.

C4 We propose an algorithm to visualize equivalent behavior with a focus on visualizing object-centric variants. This visualization is an extension of traditional variant visualization.

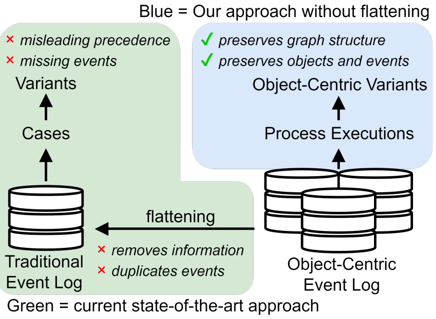


Fig. 2: Conceptual difference between our work and the current state-of-the-art for process mining on object-centric event logs. Current approaches flatten the event log leading to manipulated and removed information. The generated results might, therefore, also be misleading. We propose a graph-based case concept for object-centric event data called process executions. These can accurate represent object-centric event data.

These contributions aim to provide a foundation for moving process mining from traditional event data to object-centric event data. Using process executions, one may adapt existing algorithms and create new algorithms to discover new insights while leveraging the full, available information.

We discuss related work for this paper in Sec. II and continue with basic definitions on object-centric event data in Sec. III. These definitions build the basis for the introduction of process executions and their extraction in Sec. IV. We discuss equivalent process executions and variant visualization in Sec. V. The technical side of the contributions, i.e., scalability and efficiency, are evaluated in Sec. VI. We demonstrate the utility of our contributions in a case study in Sec. VII. We conclude the paper in Sec. VIII and provide directions for future contributions.

II. Related Work

Object-centric process mining deals with generating insights for event data with multiple objects. The problem of object multiplicity in real-life information systems was already formulated several years ago [12]. Early approaches dealt with the problem from a modeling perspective [13], [14] and with different object types being investigated separately. Object-centric process mining introduced a data-driven way to approach the problem and has recently gained attention [5], [8], [9], [15], [16]. So far, different tasks of process mining have been adapted to the object-centric setting. We group these into three categories corresponding to this paper's focus: Approaches operating case agnostically, approaches working with flattening [11] and approaches working with graphs as case concept. The techniques that were introduced for discovery [5] and conformance checking [10] work without the explicit use of a case concept. Galanti et al. [15] propose to flatten the event log to apply predictive process mining techniques to the flattened event log. The flattened event data is a traditional event log and can be used as input to all traditional process mining techniques. However, flattening is associated to the problems of *deficiency*, *convergence*. and *divergence* [11]. By not flattening the data and generalizing the case concept to a graph, these problems can be avoided. The conceptual differentiation between our work and previous work based on flattening is depicted in Fig. 2. To the best of our knowledge, we are the first paper defining a graph-based case concept and variants for object-centric event data.

III. Preliminaries and Event Data

Given a set X, a sequence $\sigma \in X^*$ of length $n \in \mathbb{N}$ assigns an enumeration to elements of the set $\sigma : \{1, \dots, n\} \to X$. We use the notation $\sigma{=}\langle \sigma_1, \dots, \sigma_n \rangle$. For an element $x \in X$ and a sequence $\sigma \in X^*$, we overload the notation $x \in \sigma$, expressing the occurrence of element x in the sequence $x \in range(\sigma)$.

We are dealing with event data of different object types. $\mathcal{T}$ defines the universe of types. There can be multiple instantiations of one type. We refer to each instantiation as an object. $\mathcal{O}$ defines the universe of objects. Each object is of one type $\pi_{type} : \mathcal{O} \to \mathcal{T}$. We define an event log containing events and objects of different types, assigning each object to an event sequence. $\mathcal{E}$ denotes the universe of event identifiers, $\mathcal{A}$ denotes the universe of event attributes and $\mathcal{V}$ denotes the universe of attribute values.

Definition 1 (Event Log)**.** *An event log $L{=}(E, O, trace, attr)$ is a tuple where*

- *$E \subseteq \mathcal{E}$ is a set of events,*
- *$O \subseteq \mathcal{O}$ is a set of objects,*
- *$trace : O {\to} E^*$ maps each object to an event sequence,*
- *$attr : E {\times} \mathcal{A} {\nrightarrow} \mathcal{V}$ maps event attributes onto values.*

We define some further notations used throughout the paper.

Definition 2 (Further Notations)**.** *Let $L{=}(E, O, trace, attr)$ be an event log. We define the following notations:*

- *$obj_L(e) = \{o \in O \mid e \in trace(o)\}$ for $e \in E$ denote the objects associated to an event.*
- *$con_L = \{(e, e') \in E \times E \mid \exists_{o \in O}\ trace(o) = \langle e_1, \dots, e_n \rangle\ \exists_{1 \leq i < n}\ e = e_i \wedge e' = e_{i+1}\}$ defines the directly-follows relationships for all events and all objects.*

An example of an event log is tabularly depicted in Fig. 3. Multiple objects are given: o1, o2, m1, m2, m3, and m4. Objects are of different types, e.g., o1 is of type Type1. Each object is associated to a sequence of events, e.g., $trace(o1) = \langle e_3, e_4, e_6 \rangle$.

The right-hand side of Fig. 3 depicts the events enriched with the information of obj_L and con_L. The result is a graph (also: event-object graph [10]), where the events form the nodes, labeled with the object information. The edges are formed by directly-follows relationships between events for an object. This graph describes the dependencies between events. Events with a path between them depend on each other, while event pairs with an absence of a path are independent. For example, e_3 is a prerequisite for e_4 and e_5. However, e_4 and e_5 have an arbitrary order. Since the object-centric event data exhibit a graph structure, we base our concept of a process execution on a graph rather than a sequence.

Event	Attr.	Object Types	
	. . .	Type1	Type2
e_1	. . .		m1
e_2	. . .		m2
e_3	. . .	o1	m1, m2
e_4	. . .	o1	
e_5	. . .		m1, m2
e_6	. . .	o1	m1, m2
e_7	. . .	o2	m3, m4
e_8	. . .	o2	
e_9	. . .		m3
e_{10}	. . .		m4
e_{11}	. . .		m3, m4
e_{12}	. . .	o2	m3, m4

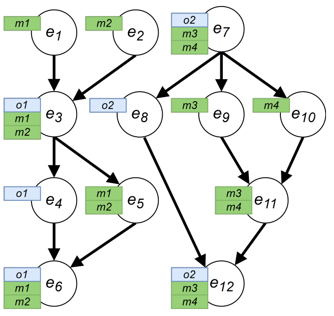


Fig. 3: Object-centric event log and the corresponding event-object graph. Events following each other for one object are connected.

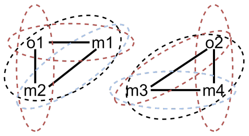


Fig. 4: Example of the object graph for Fig. 3. Each connected subgraph, indicated by the dashed lines, can extract a process execution.

IV. Process Execution Extraction

This section introduces the concept of process executions and techniques to extract process executions from an object-centric event log. The case concept in traditional process mining describes the event sequence for a single object. We generalize this concept to process executions describing the event graphs of multiple interdependent objects.

A. Process Executions

Before introducing the concept of a process execution, we first define the relationships between objects through the object graph. In this graph, objects form the nodes and are connected if they share an event.

Definition 3 (Object Graph)**.** *Let $L{=}(E, O, trace, attr)$ be an event log. The object graph is an undirected graph $OG_L = (O, C_O)$ with $C_O{=}\{\{o1, o2\}{\subseteq}O \mid \exists_{e \in E}\ o_1, o_2{\in}obj_L(e) \wedge o_1{\neq}o_2\}$. We denote the length of the shortest path between to objects $o, o'{\in}O'$ in OG_L with $dist : O' \times O' \to \mathbb{R} \cup \{\perp\}$ where $\perp$ denotes the absence of a path.*

Fig. 4 depicts an example of the object graph for the object-centric event log in Fig. 3. The graph tells us which objects co-appear in events and, therefore, which objects depend on each other, directly and transitively.

We now generalize the case concept used for traditional event data such that a process execution from an object-centric event log covers multiple objects instead of one. However, these objects must be co-dependent, i.e., they must form a connected subgraph in the object graph. A process execution is a graph formed from the events and their directly-follows relationships for a set of connected objects.

Definition 4 (Process Execution). *Let L=$(E, O, trace, attr)$ be an event log and $O' \subseteq O$ be a subset of objects that forms a connected subgraph in OG_L. The process execution of O' is a directed graph $p_{O'}$=(E', D) where*

- *$E' = \{e \in E \mid O' \cap obj_L(e) \neq \emptyset\}$ are the nodes, and*
- *$D = con_L \cap (E' \times E')$ are the edges.*

The dashed lines in Fig. 4 indicate the different connected subgraphs of the object graph that each may define a process execution. As an example we focus on the black dashed lines, i.e., the two connected components of the object graph. Using each of these two object sets, we retrieve two process executions, which are equal two the weakly connected components of the event-object graph in Fig. 3. A selection of different subgraphs leads to an extraction of different process executions. This is by design, as object graphs and their dependencies can grow very large. Splitting this graph apart or selecting only some relationships of interest can be done by only selecting specific subgraphs. In the following section, we introduce two specific extraction techniques.

a) Process Executions Through Connected Components: Our first technique retrieves the largest possible process executions, capturing all the dependencies and interactions contained in the event data. We do this by determining the connected components of the object graph. Each connected component is used to extract one process execution. The connected components of the object graph in Fig. 4 are indicated with black dashes.

Definition 5 (Connected Component Extraction). *Let $L = (E, O, trace, attr)$ be an event log and $OG_L = (O, C_O)$ its object graph. $ext_{comp}(L) = \{p_{O'} \mid O' \subseteq O \wedge (O', C_O \cap (O' \times O'))$ is a connected component of $OG_L\}$ extracts process executions by connected components.*

This technique has two main advantages. It is parameter-free, i.e., no interaction of the user is required, and it captures all dependencies between events and objects, forming a base for extensive exploration of interdependencies in the resulting process executions. However, this technique also comes with disadvantages. Connected components in the object graph can grow extremely big. In the worst case, the whole object graph is one single component leading to the extraction of only one single process execution. For these reasons, we introduce a second technique of extracting process executions that neglects some connections to retrieve smaller process executions.

b) Process Executions Through a Leading Type: Since every connected subgraph of the object graph can be used to extract process executions, we introduce a technique that identifies subgraphs of the object graph that revolve around a leading type. Such subgraphs have one node of an object of the leading type that is connected to nodes of objects of other types only by paths of the same length for each type. To find these subgraphs, the object graph is traversed in a breadth-first manner for each object of the leading type. Objects for which objects of the same type were not already traversed in an earlier level are added to the process execution and to the breadth-first search. Other objects are discarded.

Definition 6 (Leading Type Extraction). *Let L=$(E, O, trace, attr)$ be an event log, let OG_L=(O, C_O) be its object graph and ot∈$\mathcal{T}$ be an object type. $ext_{lead}(L, ot)$=$\{p_{O'} \mid o{\in}O \wedge \pi_{type}(o){=}ot \wedge O'{=}\{o'{\in}O \mid dist(o, o'){\neq}\bot \wedge \neg\exists_{o''\in O}\ \pi_{type}(o'){=}\pi_{type}(o'') \wedge dist(o, o''){<}dist(o, o')\}$ retrieves process execution through the leading type ot.*

The idea behind these subgraphs is the following: For any given object, the objects that are closest in the object graph are assumed to be the ones with the most dependencies. These close objects should be added to a process execution until objects of the same type have already been added with a shorter path length. In this way, the closest objects of each (reachable) type are added. In Fig. 4, the subgraphs obtained with the leading type technique for type Type1 are indicated with black dashes, for type Type2 with red dashes. The light blue dashed lines indicate other possible subgraphs that could be used to extract process executions but are retrievable with neither of the two introduced techniques.

V. Object-Centric Variants

In general, determining similar process executions is a form of clustering. Some general function $f_{dist}(p, p')$ can be used to determine a distance between two process executions p, p'. Based on these distances, clustering of process executions can be performed. When speaking of equivalent process executions we deal with a specific instance of this problem. Two process executions can be equivalent with respect to an event attribute $a{\in}A$ if one cannot distinguish between the executions under consideration of the event's associated object types and the event's chosen attribute. If the considered attribute is the event's activity, one typically speaks of control-flow variants. An f_{dist} is necessary that yields a value of zero if and only if process executions are equivalent. Process executions with distance zero can subsequently be grouped into equivalence classes. The problem of determining graph isomorphism fulfills these criteria.

A. Equivalence Class Mining

First, we boil a process execution down to type, order, and attribute information by projecting it onto the event attribute and types of the events.

Definition 7 (Projected Process Execution). *Let $L = (E, O, trace, attr)$ be an event log, $O' \subseteq O$ be a set of objects forming a connected subgraph in OG_L and $p_{O'} = (E', D)$ the corresponding process execution. For an attribute $a \in \mathcal{A}$, the projected process execution $p_{O'}{\downarrow}_a = (E', D, l_e, l_d)$ is defined as a graph with two label functions:*

- *$l_e(e) = (\pi_{attr}(e, a), \{(t, n) \in \mathcal{T} \times \mathbb{N}_0 \mid n = |\{o{\in}O' \mid o{\in}obj_L(e) \wedge \pi_{type}(o){=}t\}|\})$ for $e \in E'$,*
- *$l_d((e_1, e_2)) = \{(t, n) \in \mathcal{T} \times \mathbb{N}_0 \mid n = |\{o{\in}O' \mid o \in obj_L(e_1) \wedge o{\in}obj_L(e_2) \wedge \pi_{type}(o){=}t\}|\}$ for $(e_1, e_2) \in D$.*

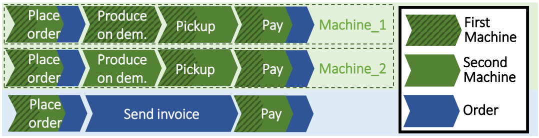
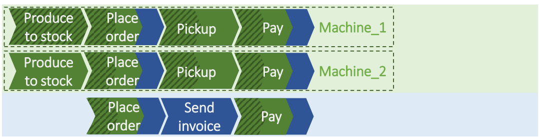
Class	Equivalence Class Visualization
1	 Place order Produce on dem. Pickup Pay Machine_1 Place order Produce on dem. Pickup Pay Machine_2 Place order Send invoice Pay First Machine Second Machine Order
2	 Produce to stock Place order Pickup Pay Machine_1 Produce to stock Place order Pickup Pay Machine_2 Place order Send invoice Pay

Table I: Example of the visualization of two variants for ordering machines using the activity attribute.

Two process executions p, p' that are equivalent with respect to an attribute $a{\in}A$ if $p{\downarrow}_a$ and $p'{\downarrow}_a$ are isomorphic under consideration of the node and edge labels.

Graph isomorphism is a well-studied problem in computer science for which, up to this point, no general polynomial-time algorithm is known [17]. Determining groups of isomorphic graphs from a set of graphs can naively be computed by conducting a one-to-one isomorphism test for each pair of graphs. We, however, employ the GROOVE framework introduced by Rensink [18]. This two-step technique first calculates an invariant hash code for each graph. Graphs with different hash codes can not be isomorphic. However, this does not mean graphs with the same hash code are isomorphic. Therefore, the initial equivalence classes of graphs with equal hash codes are refined through one-to-one comparisons. An additional benefit of employing this idea is retrieving an approximation of the equivalence classes by the initial, unrefined classes of the hash codes. The Weisfeiler-Lehman graph hashing [19] is used as a hashing function. We use the VF2 algorithm [20] for refinement of the initial classes, as it has shown superior performance for small, sparse graphs [21]. We verify the results by conducting a one-to-one comparison with the VF2 algorithm. Since calculating graph hashes is known to have scalability problems for some graphs [22], we compare the running times of the employed two-step technique with a one-to-one checking using the VF2 algorithm.

We denote this two-step function to derive equivalence classes by $iso_a{:}\{p_1, \ldots, p_n\}{\rightarrow}\{1, \ldots, m\}$, retrieving $m{\leq}n$ equivalence classes for process executions $\{p_1, \ldots, p_n\}$ and an attribute $a \in A$. An equivalence class is a set of process executions $EQ_a{=}\{p_1, \ldots, p_k\}$ such that $iso_a(p_1){=}\ldots{=}iso_a(p_k)$. The set of all equivalence classes is denoted by $\mathcal{EQ}_a$. The relative frequency of a class $EQ_a{\in}\mathcal{EQ}_a$ is given by

$$f_{EQ_a} = \frac{|EQ_a|}{\sum_{EQ'_a \in \mathcal{EQ}_a} |EQ'_a|}.$$

B. Variant Visualization

The techniques introduced before allow us to determine frequent process executions w.r.t. an event attribute. However, the mathematical object retrieved by these techniques is a graph with complex edge and node labels. These graphs are hard to interpret and understand, especially with increasing size. Therefore, we focus on delivering an improved visualization for control-flow variants using the activity attribute. To provide an accessible and understandable visualization, we introduce an extension of variant visualization used in process mining and business process management [2], [23] for control-flow variants. Traditionally, variants are visualized as a sequence of chevrons containing the activity labels.

Algorithm 1 Horizontal positioning of events

```
Require: $p_{O'}{\downarrow}_a$ - projected process execution, $e{\in}\mathcal{E}$ - event
Ensure: $x_{start}, x_{end}$ horizontal position for the event
  $x_{start} \leftarrow$ GET_X_START($p_{O'}{\downarrow}_a, e$)
  $x_{end} \leftarrow$ GET_X_END($p_{O'}{\downarrow}_a, e$)
  function GET_X_START($p_{O'}{\downarrow}_a, e$)
      pre $\leftarrow$ predecessors of $e$ in $p_{O'}{\downarrow}_a$
      if $|pre| = 0$ then
          return x=0
      end if
      return max(GET_X_START($p_{O'}{\downarrow}_a, e'$) for e' in pre)+1
  end function
  function GET_X_END($p_{O'}{\downarrow}_a, e$)
      suc $\leftarrow$ successors of $e$ in $p_{O'}{\downarrow}_a$
      if $|suc| = 0$ then
          return GET_X_START($p_{O'}{\downarrow}_a, e'$)
      end if
      return min(GET_X_START($p_{O'}{\downarrow}_a, e'$) for e' in suc)$-$1
  end function
```

An example of the result from the proposed visualization technique is depicted in Table I for two variants. There are two object types: orders (blue) and machines (green). Both variants have two objects of type machine, indicated by different shades of green. In the first variant, an order is placed for two machines. While an invoice for the order is sent, the machines are each produced and picked up concurrently, i.e., these events happen in an arbitrary order. For example, sending the invoice is not dependent on the production of any machine. To conclude, the order for both machines is paid. In the second variant, the machines are produced to stock in the beginning.

In our visualization, each type has a specific base color, and each object of such a type has a particular shade of the base color. Each object has a lane showing the event sequence for the object. Each event is depicted as a chevron with the activity label inside the chevron. If an event is shared between objects, it is placed on all corresponding object lanes and colored with the corresponding colors. The shared events are placed at the same horizontal position to respect the partial orders between events. Since the horizontal position of events depends on the previously occurring shared events and their predecessors, we introduce a layouting algorithm to determine the horizontal positions of events. The objects determine the vertical position, each object is associated to one unique vertical position grouped by type. Algorithm 1 describes retrieve the starting and ending horizontal position

Data Set	Number of Events	Number of Types	Number of Objects	Extraction Technique	Number of Executions	Events per Execution (max, min, avg)	Objects per Execution (max, min, avg)	Number of Variants
$\boldsymbol{DS_1}$ Loan Application Process	507553	2	67498	Connected components	28509	$(61, 7, 17.8)$	$(11, 2, 2.4)$	3420
				Leading type: Application	28509	$(61, 7, 17.8)$	$(11, 2, 2.4)$	3420
				Leading type: Offer	38989	$(56, 7, 18.7)$	$(2, 2, 2)$	7763
$\boldsymbol{DS_2}$ Order Management Process	22367	3	11484	Connected components	83	$(1382, 8, 269.5)$	$(717, 3, 138.4)$	83
				Leading type: Items	8159	$(155, 8, 57.9)$	$(11, 3, 6)$	8155
				Leading type: Orders	2000	$(179, 8, 51.3)$	$(68, 3, 19.4)$	1998
				Leading type: Packages	1325	$(155, 8, 49.7)$	$(32, 3, 10.5)$	1325
$\boldsymbol{DS_3}$ Customer Incident Management	119998	2	25598	Connected components	4825	$(259, 2, 24.3)$	$(53, 2, 5.1)$	3388
				Leading type: Incident	19966	$(144, 2, 25.2)$	$(3, 2, 2)$	16935
				Leading type: Customer	4826	$(259, 2, 24.3)$	$(53, 2, 5.1)$	3389
$\boldsymbol{DS_4}$ Farmer Subsidies Process	852610	6	58747	Connected components	14507	$(2973, 31, 58.8)$	$(22, 3, 4.2)$	7274
				Leading type: Payment Application	14507	$(2973, 31, 58.8)$	$(22, 3, 4.2)$	7274
				Leading type: Control summary	14507	$(2973, 31, 58.8)$	$(22, 3, 4.2)$	7274
				Leading type: Entitlement application	205	$(279, 43, 68.6)$	$(9, 5, 5.1)$	190
				Leading type: Geo parcel document	14507	$(2973, 31, 58.8)$	$(22, 3, 4.2)$	7274
				Leading type: Inspection	1999	$(2941, 48, 155.2)$	$(6, 5, 5)$	1955
				Leading type: Reference Alignment	14503	$(2973, 31, 58.8)$	$(22, 4, 4.2)$	7270

Table II: Overview of extracted process executions for different techniques, their properties, and variants.

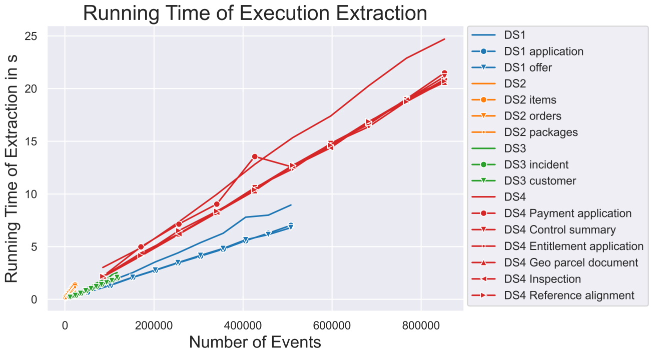


Fig. 5: Running times of execution extraction for different subset sizes, logs, and extraction techniques.

of an event. The starting position is determined recursively based on the predecessor events. The ending position is based on the starting position of the successor events.

VI. Algorithmic Evaluation

In this section, we evaluate the four contributions proposed in this paper. We use four event logs described in Table II. Three of them, $\boldsymbol{DS_1}$ [24], $\boldsymbol{DS_3}$ and $\boldsymbol{DS_4}$ [25], are real-life event logs, while $\boldsymbol{DS_2}$ is a synthetic event log, consisting of an especially high amount of connected objects and variability. First, we evaluate the results and running times of the different process execution extraction techniques. Then, we evaluate our employed two-step equivalence class calculation technique [18] and compare the running time to a one-to-one matching using the VF2-algorithm [20]. We use the event activity attribute for each data set to determine equivalence classes, i.e., we calculate variants. At last, we show the running times of our equivalence class layouting algorithm. Our algorithms are implemented in the OCPA library[1] and our experiments and data sets are publicly available on GitHub[2]. The tool OCπ [26][3] can be used to explore variants of the different event logs in and end-to-end application of our contributions.

A. Process Execution Extraction

Table II depicts the number of executions, the maximum, minimum, and average number of events and objects per execution for all data sets, and the two introduced execution extraction techniques. For leading type, the extraction is performed for each type. For some data sets, like $\boldsymbol{DS_1}$, the results are quite similar. However, for $\boldsymbol{DS_2}$ the raw data contains large connected components where many objects get entangled, visible by the low number of executions with connected components and their high average number of events. Using leading type, the executions increase in number but decrease in size, allowing for a tighter focus.

Fig. 5 shows the running times for each execution extraction technique for different sizes of each event log. Line plots without a marker refer to connected components extraction. A linear development of the running times can be observed for the given sublogs. A relationship of the slope and characteristics of the event logs, e.g., the average size of executions, seems likely. In general, the extraction by leading types is slightly faster than connected components. Both, connected components and leading type, show promising scalability for the application on real-life event data.

B. Variants

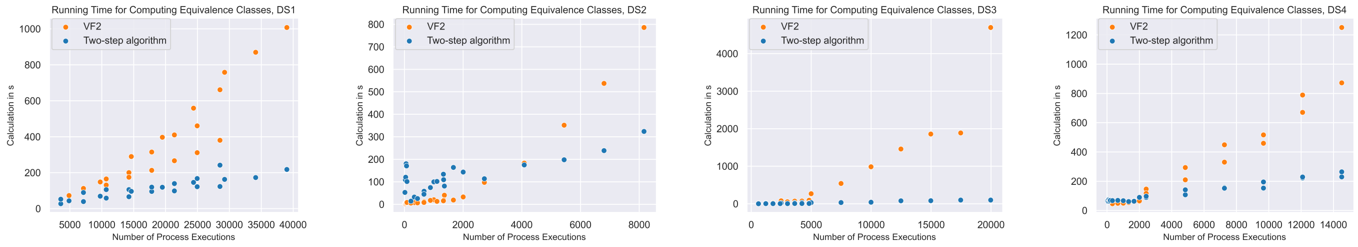
In this section, we first discuss the number of variants calculated for each event log. We then compare the results of our employed two-step technique with a baseline technique of determining isomorphic graphs from a set of graphs.

For the real-life logs, i.e., $\boldsymbol{DS_1}$, $\boldsymbol{DS_3}$, and $\boldsymbol{DS_4}$, the number of variants is often significantly smaller than the number of executions (cf. Table II). This observation shows that these types of event data contain several equivalent process executions. Only for the synthetic data set $\boldsymbol{DS_2}$ the number of equivalence classes is almost the number of process executions. Through this event log's highly entangled and interconnected nature, almost no process execution is

[1] https://github.com/ocpm/ocpa

[2] https://github.com/niklasadams/OCCasesAndVariants.git

[3] https://www.ocpi.ai



(a) DS_1

(b) DS_2

(c) DS_3

(d) DS_4

Fig. 6: Running times of the two-step algorithm compared to the VF2-algorithm for determining isomorphism.

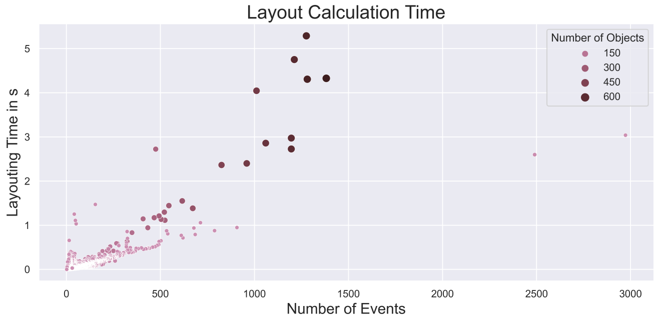


Fig. 7: Computation time of the layouting algorithm depending on the number of events and the number of objects, indicated by the size and color of the points.

equivalent to any other execution. A clustering or subgraph mining approach might be more suited for such extreme cases.

We evaluate our employed technique's correctness and running time against a baseline. This baseline works similarly to the two-step technique. However, it starts with all executions in one equivalence class and then refines it. While doing so, it performs a matching under consideration of edge and node labels using the VF2-algorithm.

Fig. 6 depicts the running times for our employed technique and the baseline, i.e., VF2, for each event log. The measures are collected by calculating equivalence classes for different subset sizes of the process executions from different extraction techniques. Generally, the two-step approach shows good scalability and efficiency compared to the baseline technique.

C. Variant Visualization

In this section, we evaluate the scalability of our variant layouting algorithm. We perform the layouting for all executions retrieved for all logs by extracting connected components. We sort the running times for each equivalence class based on the number of events in a corresponding process execution and plot the results in Fig. 7. Furthermore, each point is colored and sized according to the number of objects associated with the equivalence class. We observe increasing running times with an increase of events and objects. In general, the results show promising scalability of the layouting.

VII. End-to-End Application

This section showcases the end-to-end application of our contributions to an event log to retrieve insights about the most frequent variants. We use the event log provided by $\boldsymbol{DS_1}$. This data set describes customers applying for loans in a financial institution. After assessing an application, a loan offer is made. This offer can be accepted, refused, or canceled. Subsequently, new offers for the same application can be made. We use connected components/leading type application (both lead to the same result) to extract process executions. We extract the object-centric variants and visualize them using our layouting algorithm. Table III depicts the results.

11%, which amounts to more than 3000 process executions, show equivalent behavior: After some application steps are performed, an offer is created and sent to the customer. After a phone call, the application and the offer are canceled. However, in the second most frequent variant, the offer is returned and accepted after the phone call. Executions with multiple offers are also frequent. The fourth and fifth depicted variants show executions where the second offer is canceled or accepted. The variant visualization provides intuitive insights into the creation and concurrency of objects.

VIII. Conclusion

This paper presented four contributions to translate the concept of cases and variants to object-centric event data. The case concept is generalized to process executions: event graphs of multiple, dependent objects. We use connected subgraphs of the object graph to extract process executions and provide two specific algorithms. We determine equivalent process executions, i.e. variants, through graph isomorphism. Furthermore, we propose a visualization extending traditional variant visualization. In Sec. VII, we provided an end-to-end application of these contributions, visualizing object-centric variants of a loan application process.

The work presented in this paper can be extended in two major directions: Additional extraction techniques and different clustering techniques. The two extraction techniques introduced are just two of many possible ones that could extract patterns of interest from the object graph. Our equivalence class calculation is one specific type of clustering. Process executions could be clustered in other ways based

Activity abbreviations: ACA = Create Application, AS = Application Submitted, ACon = Application Concept, WCA = Application Workflow Completed, AA = Application Accepted, OCO = Create Offer, OCre = Offer Created, OSMO = Offer Sent Mail & Online, WCAO = Calling After Offer, ACom = Application Complete, ACan = Application Cancelled, OCan = Offer Cancelled, OR = Offer returned, WVA = Validate Application, AV = Validating, WCIF = Calling for Incomplete Files, AI = Application Incomplete, OA = Offer Accepted, AP = Pending

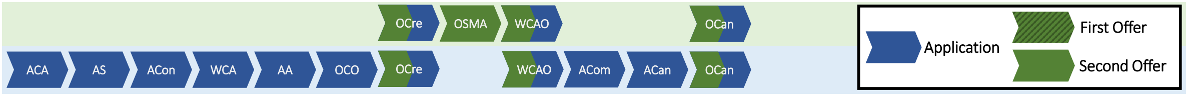
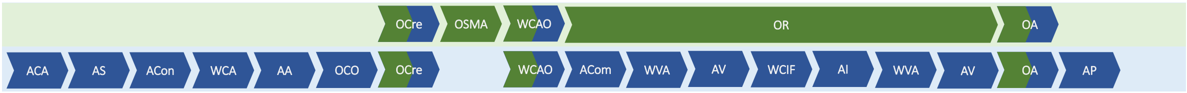
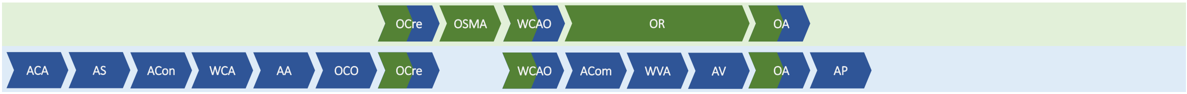
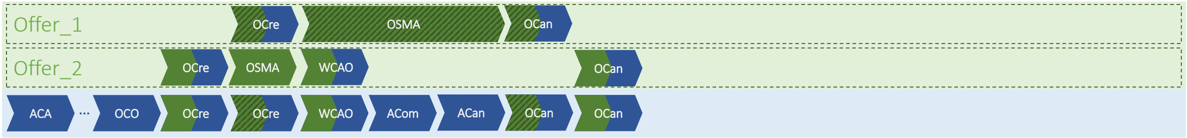
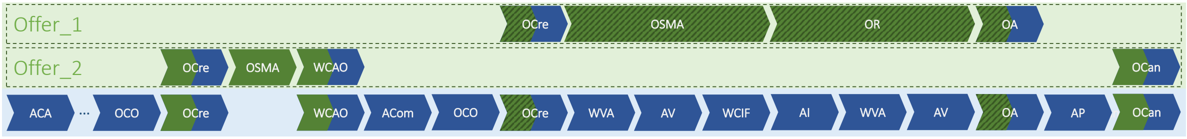
Variant number (corresponding frequency)	& visualized **object-centric variant**
1 (11%)	 OCre OSMA WCAO OCan ACA AS ACon WCA AA OCO OCre WCAO ACom ACan OCan Application First Offer Second Offer
2 (5%)	 OCre OSMA WCAO OR OA ACA AS ACon WCA AA OCO OCre WCAO ACom WVA AV WCIF AI WVA AV OA AP
3 (4%)	 OCre OSMA WCAO OR OA ACA AS ACon WCA AA OCO OCre WCAO ACom WVA AV OA AP
14 (2%)	 Offer_1 OCre OSMA OCan Offer_2 OCre OSMA WCAO OCan ACA ··· OCO OCre OCre WCAO ACom ACan OCan OCan
35 (0.4%)	 Offer_1 OCre OSMA OR OA Offer_2 OCre OSMA WCAO OCan ACA ··· OCO OCre WCAO ACom OCO OCre WVA AV WCIF AI WVA AV OA AP OCan

Table III: The three most frequent object-centric variants in the loan application process. Additionally, we depict two variants with multiple offers for one application. Activities are abbreviated using the first letters.

on other distance measures. Clustering would also help with generating insights for highly entangled event sequences.

Cases are essential for traditional process mining techniques. With this paper, we provide a technique to extract the object-centric equivalent of cases. These can be used to adapt existing process mining techniques to the object-centric setting and develop new techniques providing novel insights.

References

[1] W. M. P. van der Aalst, *Process mining: Data science in action*. Springer, 2016.

[2] D. Schuster, S. J. van Zelst, and W. M. P. van der Aalst, "Cortado - an interactive tool for data-driven process discovery and modeling," in *PETRI NETS*. Springer, 2021, pp. 465–475.

[3] A. Senderovich, M. Weidlich, L. Yedidsion, A. Gal, A. Mandelbaum, S. Kadish, and C. A. Bunnell, "Conformance checking and performance improvement in scheduled processes: A queueing-network perspective," *Inf. Syst.*, vol. 62, pp. 185–206, 2016.

[4] N. Tax, I. Verenich, M. L. Rosa, and M. Dumas, "Predictive business process monitoring with LSTM neural networks," in *CAiSE*. Springer, 2017, pp. 477–492.

[5] W. M. P. van der Aalst and A. Berti, "Discovering object-centric Petri nets," *Fundam. Informaticae*, vol. 175, no. 1-4, pp. 1–40, 2020.

[6] G. Schuh, A. Gützlaff, S. Cremer, S. Schmitz, and A. Ayati, "A data model to apply process mining in end-to-end order processing processes of manufacturing companies," in *IEEM*. IEEE, 2020, pp. 151–155.

[7] Q. Qi and F. Tao, "Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison," *IEEE Access*, vol. 6, pp. 3585–3593, 2018.

[8] P. Waibel, L. Pfahlsberger, K. Revoredo, and J. Mendling, "Causal process mining from relational databases with domain knowledge," *CoRR*, vol. abs/2202.08314, 2022.

[9] S. Esser and D. Fahland, "Multi-dimensional event data in graph databases," *J. Data Semant.*, vol. 10, no. 1, pp. 109–141, 2021.

[10] J. N. Adams and W. M. P. van der Aalst, "Precision and fitness in object-centric process mining," in *ICPM*. IEEE, 2021, pp. 128–135.

[11] W. M. P. van der Aalst, "Object-centric process mining: Dealing with divergence and convergence in event data," in *SEFM*. Springer, 2019, pp. 3–25.

[12] W. M. P. van der Aalst et al., "Process mining manifesto," in *BPM Workshops*. Springer, 2011, pp. 169–194.

[13] D. Fahland, "Describing behavior of processes with many-to-many interactions," in *PETRI NETS*. Springer, 2019, pp. 3–24.

[14] E. H. J. Nooijen, B. F. van Dongen, and D. Fahland, "Automatic discovery of data-centric and artifact-centric processes," in *BPM Workshops*, M. L. Rosa and P. Soffer, Eds. Springer, 2012, pp. 316–327.

[15] R. Galanti, M. de Leoni, N. Navarin, and A. Marazzi, "Object-centric process predictive analytics," *CoRR*, vol. abs/2203.02801, 2022.

[16] G. Park, J. N. Adams, and W. M. P. van der Aalst, "OPerA: Object-centric performance analysis," *CoRR*, vol. abs/2204.10662, 2022.

[17] M. Grohe and D. Neuen, "Recent advances on the graph isomorphism problem," *CoRR*, vol. abs/2011.01366, 2020.

[18] A. Rensink, "Isomorphism checking in GROOVE," *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, vol. 1, 2006.

[19] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, pp. 2539–2561, 2011.

[20] L. P. Cordella *et al.*, "An improved algorithm for matching large graphs," in *IAPR-TC15*, 2001, pp. 149–159.

[21] P. Foggia, C. Sansone, and M. Vento, "A performance comparison of five algorithms for graph isomorphism," in *IAPR TC-15*, 2001, pp. 188–199.

[22] D. Conte *et al.*, "Thirty years of graph matching in pattern recognition," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 18, no. 3, pp. 265–298, 2004.

[23] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.

[24] B. van Dongen, "BPI challenge 2017," https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b.

[25] ——, "BPI challenge 2018," https://doi.org/10.4121/uuid:3301445f-95e8-4ff0-98a4-901f1f204972.

[26] J. N. Adams and W. M. P. van der Aalst, "Ocπ: Object-centric process insights," in *PETRI NETS*. Springer, 2022, pp. 139–150.