Layouting Object-Centric Directly Follows Graphs

Deoksang Lee[1[0000-0001-9703-3172]], Minseok Song[1[0000-0002-6813-8853]], and Wil M.P. van der Aalst[2[0000-0002-0955-6940]]

[1] Department of Industrial and Management Engineering, Pohang University of Science and Technology (POSTECH), Pohang, South Korea
{duksang4834, mssong}@postech.ac.kr
[2] Process and Data Science, RWTH Aachen University, Aachen, Germany
{wvdaalst}@pads.rwth-aachen.de

Abstract. Process mining extracts insights from event logs to analyze and optimize business processes. Object-Centric Process Mining (OCPM) extends traditional process mining by utilizing Object-Centric Event Logs (OCEL), explicitly capturing interactions among multiple objects within business processes. Despite its advantages, visualizing object-centric process models remains challenging due to increased complexity, leading to denser graphs with more nodes and edges compared to traditional process mining. To address this challenge, this study proposes an advanced process layout generation method designed to improve the visualization clarity of Object-Centric Directly Follows Graph (OC-DFG), a widely used representation of object-centric process models. Our approach systematically assigns distinct axes to each object type, enhancing readability by clearly separating interactions. Additionally, the method incorporates edge crossminimization and spatial optimization techniques to further improve layout efficiency. We validate the proposed method quantitatively using both traditional layout evaluation metrics and newly developed metrics specifically designed for OCPM. Experimental results indicate notable improvements in layout metrics associated with readability for OC-DFG. This work contributes to process mining, modeling, and analytics by offering a structured visualization approach for complex object-centric process models, which may support stakeholders in analyzing processes and making informed decisions.

Keywords: OC-DFG, Object-Centric Directly Follows Graph, Layout, Visualization, OCPM, Object-Centric Process Mining.

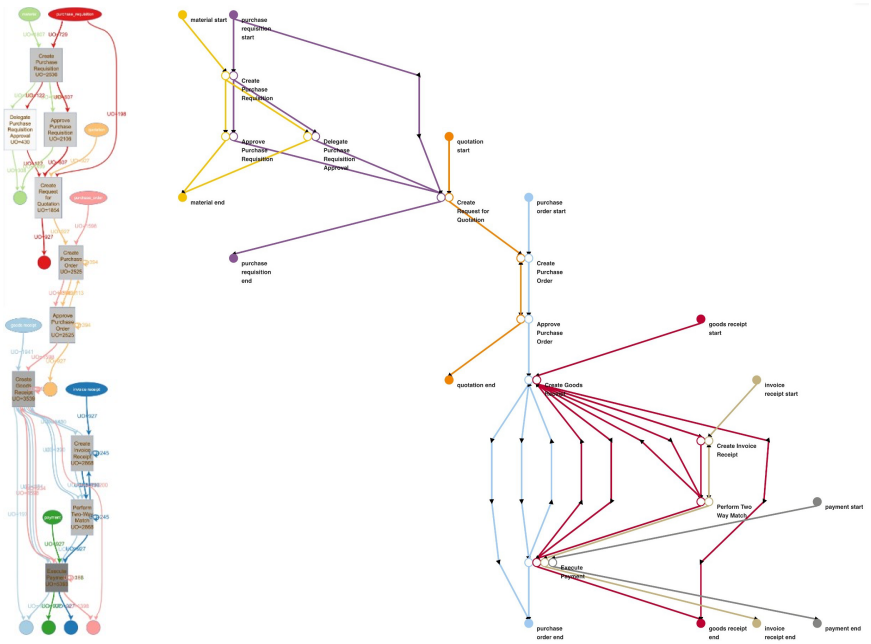
1 Introduction

In the era of data-driven process automation, business process management is critical in optimizing workflows, ensuring compliance, and supporting decision-making [1, 2]. Process mining, a key discipline within business process management, enables organizations to extract insights from event logs recorded in information systems [3]. Traditionally, process mining has relied on case-centric event logs, where a single case type represents an entire process. However, real-world business processes involve multiple interacting objects such as orders, customers, suppliers, and machines. The case-centric

paradigm fails to capture these interactions, leading to information loss and limiting its applicability to complex business environments.

To overcome this limitation, Object-Centric Process Mining (OCPM) emerged as a new paradigm that models business processes from a multi-object perspective [4, 5]. Unlike traditional process mining, OCPM explicitly represents interactions between multiple objects using Object-Centric Event Logs (OCEL), providing a more realistic and granular view of processes.

Following the rise of OCPM, various object-centric process model discovery techniques have been developed [6–8]. However, a significant challenge remains: the absence of effective layout generation methods for visualizing object-centric process models. Prior research established that effective visualizations enhance stakeholders' decision-making in business contexts [9–12]. Despite this, existing studies have relied on general graph layouts that prioritize aesthetic placement of nodes and edges rather than representing processes well. This approach hinders users' ability to understand business processes. Furthermore, they do not differentiate object types, preventing users from distinguishing object types and understanding interactions between object types. As depicted in Fig. 1(a), these limitations impede process analysts from using OCPM in real-world business environments. Consequently, a more advanced layout generation method is needed to differentiate object types and visualize interactions clearly, as shown in Fig. 1(b).



(a) (b)

Fig. 1. (a) A visual representation of an object-centric process model using a general graph layout. (b) The same object-centric process model visualized using the proposed method.

To address this challenge, we propose an advanced process layout generation method to enhance the visualization of object-centric process models. Specifically, we focus on Object-Centric Directly-Follows Graphs (OC-DFG), one of the most widely used object-centric process model representations [5]. Our approach assigns a distinct object type axis to each object type, positioning frequently interacting object types closer together while placing less related types farther apart. Additionally, we introduce cross-minimization and node positioning methods tailored to OC-DFG layouts. To evaluate the proposed method, we conduct a quantitative evaluation, introducing new metrics to measure the quality of OC-DFG layouts.

The proposed OC-DFG layout generation method may help users better interpret OC-DFG and extract process-related insights more efficiently, thereby accelerating the adoption of OCPM in practice. While this study focuses on OC-DFG, the method can also be applied to other object-centric process models such as Object-Centric Petri Nets (OC-PN) and Object-Centric Business Process Modeling Notations (OC-BPMN), broadening its applicability in process mining and business process management.

2 Background

This section introduces fundamental concepts for understanding the proposed method.

2.1 Universe

Let $\mathbb{U}_{ev}$ be the universe of events, $\mathbb{U}_{act}$ be the universe of activities, $\mathbb{U}_{o}$ be the universe of objects, $\mathbb{U}_{ot}$ be the universe of object types, $\mathbb{G}$ be the universe of the graph, $\mathbb{Z}$ be the universe of integers, and $\mathbb{F}$ be the universe of real numbers.

2.2 Backbone-based DFG layout, backbone, rank, and order

Backbone-based DFG layout is a visualization method designed to enhance DFG readability and clarity [13], as illustrated in Fig. 2.

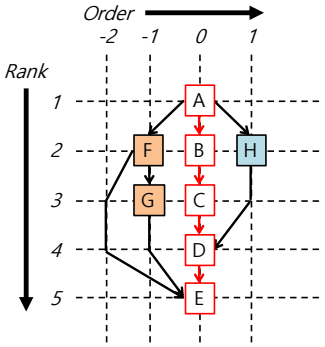


Fig. 2. An example of the backbone-based DFG layout. Red nodes and edges represent the backbone structure.

It identifies a backbone that represents the most frequent or significant process flow and serves as the central structure. For example, in Fig. 2, the sequence $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$, colored red, forms the backbone of the layout.

In the backbone-based DFG layout, node positions are determined by rank and order. Rank refers to vertical placement, where nodes with lower ranks appear higher in the layout. Conversely, order determines horizontal positioning, with nodes of lower order placed to the left of those with higher order.

2.3 Object-centric event log (OCEL)

An OCEL extends traditional event logs to capture multi-object interactions within processes [4, 5, 14], as shown in Table 1. According to prior studies [4, 5], the OCEL is defined as a tuple $OCEL = (EV, ET, O, OT, \pi_{act}, \pi_{time}, \pi_{omap}, \pi_{otype}, \pi_{atype} \leq)$, where $EV \subseteq \mathbb{U}_{ev}$ is a set of events, $ET \subseteq \mathbb{U}_{act}$ is a set of activities, $O \subseteq \mathbb{U}_{ot}$ is a set of objects, and $OT \subseteq \mathbb{U}_{ot}$ is a set of object types. $\pi_{act} : EV \rightarrow ET$ maps events to activities, $\pi_{time} : EV \rightarrow \mathbb{U}_{time}$ assigns timestamps to events, $\pi_{omap} : EV \rightarrow O$ relates events to objects, $\pi_{otype} : O \rightarrow OT$ assigns objects to object types, and $\pi_{atype} : ET \rightarrow 2^{OT}$ maps each activity to a set of object types, and $\leq$ defines a total order on events based on their timestamps.

In Table 1, $EV = \{ev_1, ev_2, ev_3, ev_4, ev_5\}$, $ET = \{po, ca, pi\}$, $O = \{o_1, o_2, i_1, i_2\}$, $OT = \{Order, Item\}$, $\pi_{act}(ev_1) = po$, $\pi_{act}(ev_2) = ca$, $\pi_{time}(ev_1) =$ 09-03-2025 15:34, $\pi_{omap}(ev_2) = \{o_1, i_1\}$, $\pi_{otype}(o_1) = Order$, $\pi_{otype}(i_1) = Item$, $\pi_{atype}(\text{po}) = \{Order\}$, and $\pi_{atype}(ca) = \{Order, Item\}$.

Table 1. Example of an OCEL

Event ID	Activity name	Timestamp	Objects involved	
			Order	Item
ev_1	*place_order (po)*	09-03-2025 15:34	$\{o_1\}$	-
ev_2	*check_availability (ca)*	09-03-2025 15:40	$\{o_1\}$	$\{i_1\}$
ev_3	*place_order (po)*	09-03-2025 16:00	$\{o_2\}$	-
ev_4	*check_availability (ca)*	09-03-2025 16:15	$\{o_2\}$	$\{i_2\}$
ev_5	*pick_item (pi)*	09-03-2025 16:45	$\{o_1\}$	$\{i_1\}$

2.4 Object-centric directly follows graph (OC-DFG)

An OC-DFG is a process model that intuitively represents interactions among multi-objects using nodes and edges [5]. To discover the OC-DFG, the OCEL is flattened for each object type, generating an individual DFG per object type. Then, these DFGs are merged to form the final OC-DFG, integrating interactions across object types.

3 OC-DFG layout generation method

In this section, we propose an OC-DFG layout generation method consisting of data

preparation, backbone determination, rank assignment, DFG layout generation for each object type, OC-DFG layout generation, cross-minimization, and node positioning.

3.1 Data preparation

In this step, we construct the sets of nodes N, edges E, and variants V from the given $OCEL = (EV, ET, O, OT, \pi_{act}, \pi_{time}, \pi_{omap}, \pi_{otype}, \pi_{atype}, \leq)$.

To ensure that each object type has clearly defined start and end points, we introduce synthetic start and end nodes, denoted as $N_{start} = \{start_{ot} | ot \in OT\}$ and $N_{end} = \{end_{ot} | ot \in OT\}$. We update π_{atype} as $\pi_{atype} = \pi_{atype} \cup \{(start_{ot}, ot) \mid ot \in OT\} \cup \{(end_{ot}, ot) \mid ot \in OT\}$. In Table 1, $N_{start} = \{start_{Order}, start_{Item}\}$, $N_{end} = \{end_{Order}, end_{Item}\}$, $\pi_{atype}(start_{Order}) = Order$, $\pi_{atype}(start_{Item}) = Item$, $\pi_{atype}(end_{Order}) = Order$, and $\pi_{atype}(end_{Item}) = Item$.

Each object $o \in O$ is associated with a trace τ_o, which is the ordered sequence of activities linked to that object. Let $ev_1, \dots, ev_n \in EV$ be events such that $o \in \pi_{omap}(ev_i)$ for all $1 \leq i \leq n$, and the sequence $ev_1 \leq ev_2, \dots \leq ev_n$ respects the total order $\leq$ defined over the OCEL. Then, the trace τ_o is constructed as $\tau_o = \langle start_{ot}, \pi_{act}(ev_1), \pi_{act}(ev_2), \dots, \pi_{act}(ev_n), end_{ot} \rangle$ where $ot = \pi_{otype}(o)$. For instance, in Table 1, the trace of o_1 is $\tau_{o_1} = \langle start_{Order}, po, ca, pi, end_{Order} \rangle$.

Based on the collection of traces, we define three sets. First, the node set includes all activities recorded in the event log, as well as the synthetic start and end nodes, i.e., $N = ET \cup N_{start} \cup N_{end}$. Second, the edge set E is constructed from all transitions between consecutive activities in each trace. Each edge is labeled with the corresponding object type. Formally, the edge set is defined as $E = \{(a_i, a_{i+1}, ot) \mid o \in O \wedge ot = \pi_{otype}(o) \wedge \tau_o = \langle start_{ot}, a_1, \dots, a_n, end_{ot} \rangle \wedge 0 \leq i \leq n\}$. In Table 1, the edges are $E = \{(po, ca, Order), (ca, pi, Order), (ca, pi, Item)\}$. Third, the variant set V is built by collecting all unique traces found in the event log, i.e., $V = \{\tau_o \mid o \in O\}$.

After constructing these sets, we organize the nodes, edges, and variants by object type. For each object type $ot \in OT$, we define N_{ot} as the set of nodes associated with ot, i.e., $N_{ot} = \{n \in N \mid ot \in \pi_{atype}(n)\}$, E_{ot} as the subset of edges labeled with ot, i.e., $E_{ot} = \{(a_1, a_2, ot') \in E \mid ot' = ot\}$, and V_{ot} as the set of traces for object type ot, i.e., $V_{ot} = \{\tau_o \mid o \in O, \pi_{otype}(o) = ot\}$.

3.2 Backbone determination

Given a set of nodes N, a set of edges E, and a set of variants V over an $OCEL = (EV, ET, O, OT, \pi_{act}, \pi_{time}, \pi_{omap}, \pi_{otype}, \pi_{atype}, \leq)$, we determine the backbone for each object type $ot \in OT$. The backbone is intended to represent the core sequence of activities within the process. In this study, we derived the backbone from the most frequent variant.

Let $mv_{ot} = \langle a_1, a_2, \dots, a_n \rangle$ be the most frequent variant of ot, where each $a_i \in N$. The sequence of backbone nodes is $BN_{ot} = \langle b_1, b_2, \dots, b_k \rangle$, where each $b_i \in N$, all elements b_i are distinct, and their relative order in mv_{ot} is preserved. In other words, BN_{ot} consists of the unique activities from mv_{ot}, preserving the order in which they first

appear. The set of backbone edges is defined as the set of directed transitions between consecutive backbone nodes that are also observed in the original variant. Formally, $BE_{ot} = \{(b_i, b_{i+1}, ot) \mid 1 \leq i < k \wedge j \in \{1, \dots, n-1\} \wedge a_j = b_i \wedge a_{j+1} = b_{i+1}\}$.

For example, consider the most frequent variant for an object type ot as $mv_{ot} = \langle A, B, A, C, D, E, D \rangle$. The backbone node sequence is $BN_{ot} = \langle A, B, C, D, E \rangle$ and the set of backbone edges is $BE_{ot} = \{(A, B, ot), (C, D, ot), (D, E, ot)\}$. The edge (B, C, ot) is excluded as it does not appear in the original variant.

For convenience, we denote the i-th element of a sequence S as $S[i]$. Accordingly, elements of the backbone node sequence can be accessed using the notation $BN_{ot}[i]$.

3.3 Rank assignment

In this step, node ranks are assigned using Integer Programming (IP) and node rank mapping function is obtained. The prior study [13] suggested a manual rank assignment method using traditional event logs. However, it has two limitations when applied to OCPM: it may introduce unnecessary back edges and result in longer edge lengths.

To address these issues, we introduce an IP-based rank assignment method. The model employs four decision variables: r_n, y_e, z_e, and α_e. Here, r_n is an integer variable representing the rank of node n ($r_n \geq 1$), and y_e is an integer variable capturing the length of edge e ($y_e \geq 0$). The binary variable z_e equals 1 if the start node's rank exceeds the end node's rank for edge e, and 0 otherwise. Precedence violation α_e is an integer variable ($\alpha \geq 0$) that accounts for cases where the start node's rank is higher than the end node's rank. Functions $snode : E \to N$ and $enode : E - N$ return the start and end nodes of an edge e, $freq : E \to \mathbb{Z}$ returns the frequency of the edge.

$$Min \qquad \sum_{e \in E} freq(e) \times \alpha_e + y_e \tag{1}$$

$$s.t. \qquad r_{snode(e)} + 1 \leq r_{enode(e)} + \alpha_e \; \forall e \in E \tag{2}$$

$$r_{start_{ot}} + 1 \leq r_n \; \forall n \in N_{ot}, n \neq start_{ot} \tag{3}$$

$$r_{end_{ot}} \geq r_n + 1 \; \forall n \in N_{ot}, n \neq end_{ot} \tag{4}$$

$$y_e \geq r_{snode(e)} - r_{enode(e)} \; \forall e \in E \tag{5}$$

$$y_e \geq -\left(r_{snode(e_i)} - r_{enode(e_i)}\right) \forall e \in E \tag{6}$$

$$r_{BN_{ot}[i]} + 1 \leq r_{BN_{ot}[i+1]} \; \forall i = 1,2, \dots, |BN_{ot}| - 1, \forall ot \in OT \tag{7}$$

$$x_{snode(e)} - x_{enode(e)} \leq -\varepsilon + M z_e \; \forall e \in E \tag{8}$$

$$x_{snode(e_i)} - x_{enode(e_i)} \geq \varepsilon - (1 - z_e) M \; \forall e \in E \tag{9}$$

The objective function (1) minimizes the precedence violations and edge length. Constraint (2) ensures that the end node is placed below its start node. However, in some cases, this may not be feasible. To handle such cases, we introduce α_e in constraint (2). Constraints (3) and (4) enforce that the start and end nodes of object types are positioned above and below all other nodes of the same type, respectively. Constraints (5) and (6) calculate the edge length. Constraint (7) ensures that the ranks of nodes in the backbone increase, placing backbone nodes linearly. Constraints (8) and (9) enforce that start and end nodes of an edge occupy different ranks to avoid horizontal edges (ε: a very small constant; M: a very large constant). After determining the node ranks, we define the set of node ranks $R \subseteq \mathbb{Z}^+$ as $R = \{r_n \mid n \in N\}$ and mapping function $\pi_{rank} : N \to \mathbb{Z}^+$ as $\pi_{rank} = \{(n, r_n) \mid n \in N\}$.

3.4 DFG layout generation for each object type

Definition 1 (DFG layout). *A DFG layout is a tuple $L = (N, E, \pi_{rank}, \pi_{order})$ where N is a set of nodes, $E \subseteq N \times N$ is a set of edges, $\pi_{rank} : N \to \mathbb{Z}^+$ is a node rank mapping function, and $\pi_{order} : N \to \mathbb{Z}$ is a node order mapping function.*

In this stage, we construct a DFG layout for each object type by determining the node orders. To achieve this, we adopt the method proposed in [13], which determines node orders based on nodes, edges, backbone, and node ranks.

Given an $OCEL = \big(EV, ET, O, OT, \pi_{act}, \pi_{time}, \pi_{omap}, \pi_{otype}, \pi_{atype} \leq\big)$, we have already extracted the required elements for each object type ot, as described in Section 3.1 and 3.2: a set of nodes N_{ot}, a set of edges E_{ot}, a sequence of backbone nodes BN_{ot}, and a set of backbone edges BE_{ot} for each object type. Furthermore, in Section 3.3, we determined the rank mapping function $\pi_{rank} : N \to \mathbb{Z}^+$. From this, we define the node mapping function for each object type as $\pi^{ot}_{rank} = \{(n, \pi_{rank}(n)) \mid n \in N_{ot}\}$. Using these elements, we compute the node orders $O_{ot} \subseteq \mathbb{Z}$ and obtain mapping function $\pi^{ot}_{order} : N_{ot} \to \mathbb{Z}$ following the method from [13]. As a result, the finalized DFG layout for each object type is constructed as $L_{ot} = (N_{ot}, E_{ot}, \pi^{ot}_{rank}, \pi^{ot}_{order})$.

3.5 OC-DFG layout generation

After generating DFG layouts for each object type, these individual layouts are incrementally merged to construct an OC-DFG layout. This merging process consists of four steps: (1) initializing the OC-DFG layout with the main object type, (2) selecting the target object type, (3) generating OC-DFG layout candidates, and (4) selecting the optimal layout. Steps (2) through (4) are repeated until no object types remain to be merged.

Recall that we have already obtained the set of nodes N, the set of nodes for each object type N_{ot}, the set of edges E, the set of edges for each object type E_{ot}, the node rank mapping function π_{rank}, the node rank mapping function for each object type π^{ot}_{rank}, the set of node order mapping function for each object type π^{ot}_{order}, and the DFG

layout for each object type L_{ot}. Before detailing the merging procedure, we introduce several concepts.

Definition 2 (Object type axis). *Let OT be a set of object types. An object type axis of an object type $ot \in OT$ is defined as a non-negative integer, i.e.,* $\mathrm{ota} \in \mathbb{Z}_{\geq 0}$.

The object type axis indicates the horizontal index of the object type within the OC-DFG layout. For example, suppose an OC-DFG layout assigns axis values of 1, 0, and 2 to ot_A, ot_B, and ot_C. This implies that ot_B appears on the leftmost side of the layout, ot_A is placed between ot_B and ot_C, and ot_C is positioned on the rightmost side.

Definition 3 (Node object type axis function). *Let N be a set of nodes, OT be a set of object types, $\pi_{atype} : N \rightarrow OT$ be a node object type mapping function, OTA be a set of object type axes, $\pi_{ota} : OT \rightarrow OTA$ be an object type axis mapping function. Node object type axis function $\pi_{nota} : N \rightarrow OTA$ is $\pi_{nota} = \{(n, \pi_{ota}(\pi_{atype}(n)))\}$.*

For example, suppose an OC-DFG layout includes nodes of object types ot_A, ot_B, and ot_C, assigned to axes 1, 0, and 2. If a node n is positioned along the axis of ot_B, then $\pi_{atype}(n) = ot_B$, $\pi_{ota}(ot_B) = 0$, and thus $\pi_{nota}(n) = 0$.

Definition 4 (OC-DFG layout). *An OC-DFG layout is defined as a tuple $OL = (N, E, OT, OTA, \pi_{rank}, \pi_{order}, \pi_{ota}, \pi_{nota})$ where N* is a set of nodes, $E \subseteq N \times N \times OT$ is a set of edges, *OT is a set of object types*, OTA is a set of object type axes, $\pi_{rank} : N \rightarrow \mathbb{Z}^+$ is a node rank mapping function, $\pi_{order} : N \rightarrow \mathbb{Z}$ is a node order mapping function, $\pi_{ota} : OT \rightarrow OTA$ is an object type axis mapping function, and $\pi_{nota} : N \rightarrow OTA$ is a node object type axis function.

1) Initializing the OC-DFG layout with the main object type. In this step, an OC-DFG layout OL is initialized using a main object type. The main object type is identified as the highest number of shared nodes among object types. Let $OCEL = (EV, ET, O, OT, \pi_{act}, \pi_{time}, \pi_{omap}, \pi_{otype}, \pi_{atype} \leq)$ be an OCEL and N_{ot} be a set of nodes of the object type $ot \in OT$. The main object type is determined as $ot = argmax_{ot' \in OT} |\{n \mid n \in N_{ot'} \wedge |\pi_{atype}(n)| > 1\}|$.

Then, the OC-DFG layout $OL = (N, E, OT, OTA, \pi_{rank}, \pi_{order}, \pi_{ota}, \pi_{nota})$ is initialized. Let ot be a main object type and $L_{ot} = (N_{ot}, E_{ot}, \pi^{ot}_{rank}, \pi^{ot}_{order})$ be the DFG layout of the main object type. Then, the OC-DFG OL is initialized as $N = N_{ot}$, $E = \{(a_1, a_2, ot) \mid (a_1, a_2) \in E_{ot}\}$, $OT = \{ot\}$, $OTA = \{0\}$, $\pi_{rank} = \pi^{ot}_{rank}$, $\pi_{order} = \{(n, -\min_{n \in N}(\pi^{ot}_{order}(n)) + \pi^{ot}_{order}(n)) \mid n \in N_{ot}\}$, $\pi_{ota} = \{(ot, 0)\}$, and $\pi_{nota} = \{(n, 0) \mid n \in N_{ot}\}$.

2) Selecting the target object type. Next, a target object type ot is selected. Before explaining how to select the target object type, we introduce the distance between two set of nodes.

Definition 5 (Distance between two set of nodes). *Let* N_A *and* N_B *be sets of nodes. The distance between two sets is defined as* $dist(N_A, N_B) = 1 - (2 \times |N_A \cap N_B|)/(|N_A| + |N_B|)$.

This measure reflects the dissimilarity between N_A and N_B with a value of 0 indicating identical sets and 1 indicating no overlap.

The target object type is the one whose DFG layout is the most like the current layout OL. Let $OCEL$ be an OCEL, OL be a current OC-DFG layout, and $L_{ot'}$ be a DFG layout of object type ot'. Then, the target object type ot is defined as $ot = argmin_{ot' \in OT(OCEL), ot' \notin OT(OL)} dist(N(OL), N(L_{ot'}))$.

3) Generating OC-DFG layout candidates. After determining the target object type ot, the target DFG layout $L_{ot} = (N_{ot}, E_{ot}, \pi^{ot}_{rank}, \pi^{ot}_{order})$ is merged with the current OC-DFG layout $OL = (N, E, OT, OTA, \pi_{rank}, \pi_{order}, \pi_{ota}, \pi_{nota})$. The target axis $ta \in \mathbb{Z}_{\geq 0}$ indicates the horizontal position where L_{ot} is to be inserted. There are $|OT(OL)| + 1$ possible values for ta. For example, if OL contains object types ot_A, ot_B, and ot_C assigned to axes 0, 1, and 2, then ta can take on values 0 (inserting to the left of ot_A), 1 (between ot_A and ot_B), 2 (between ot_B and ot_C), or 3 (to the right of ot_C).

For each possible target axis ta, we construct temporary layout. Then, we choose the layout with the lowest cost in the next step. The temporary OC-DFG layout $OL' = (N', E', OT', OTA', \pi'_{rank}, \pi'_{order}, \pi'_{ota}, \pi'_{nota})$ is constructed as $N' = N \cup N_{ot}$, $E' = E \cup \{(a_1, a_2, ot_{target}) \mid (a_1, a_2) \in E_{ot}\}$, and $\pi'_{rank} = \pi_{rank} \cup \pi^{ot}_{rank}$. The function π'_{order} is updated to maintain consistency in the layout. First, three parameters are computed: the left margin of OL at ta as $lm = \max(\{\pi_{order}(n) \mid n \in N \wedge \pi_{nota}(n) < ta\}) + 1$; the order of the leftmost node in L_{ot} as $lmo = \min(\{\pi^{ot}_{order}(n) \mid n \in N_{ot}\})$; and the width of L_{ot} as $w = \max(\{\pi^{ot}_{order}(n) \mid n \in N_{ot}\}) - \min(\{\pi^{ot}_{order}(n) \mid n \in N_{ot}\})$. Nodes with an object type axis less than ta retain their order. If a node's object type axis equals ta, its order is updated using lm, lmo, and its local order. Nodes with an axis greater than ta adjust their order based on its global order in OL and the width of $L_{ot_{target}}$: $\pi'_{order} = \{(n, \pi_{order}(n)) \mid n \in N \wedge \pi_{nota}(n) < ta\} \cup \{(n, \pi_{order}(n) + w - 1) \mid n \in N \wedge \pi_{nota}(n) \geq ta\} \cup \{(n, lm + \pi^{ot}_{order}(n) - lmo + 1) \mid n \in N_{ot} \wedge n \notin N\}$. The π'_{ota} is adjusted based on the position of the target object type. If an object type is the target, it is assigned the target axis ta. Object types positioned to the left of ta retain their positions, whereas those at or beyond ta shift to the right as $\pi'_{ota} = \{(ot', ta) \mid ot' \in OT \wedge ot' = ot\} \cup \{(ot', \pi_{ota}(ot')) \mid ot' \in OT \wedge \pi_{ota}(ot') < ta\} \cup \{(ot', \pi_{ota}(ot') + 1) \mid ot' \in OT \wedge \pi_{ota}(ot') > ta\}$. The function π'_{nota} is updated to reflect the inclusion of the new object type. If a node exists only in L_{ot}, it is assigned to ta. If a node was positioned to the left of ta, it retains its position. Otherwise, it is shifted to the right as $\pi'_{nota} = \{(n, ta) \mid n \in N_{ot} \wedge n \notin N\} \cup \{(n, \pi_{nota}(n)) \mid n \in N \wedge \pi_{nota}(n) < ta\} \cup \{(n, \pi_{nota}(n) + 1) \mid n \in N \wedge \pi_{nota}(n) \geq ta\}$.

As a result, the set of OC-DFG layouts $C = \{OL'_0, OL'_1, \ldots, OL'_{|OT|}\}$ is obtained. The example of merging OL with L_{ot} at the target axis zero is shown in Fig. 3.

4) Selecting the layout with the lowest cost. After generating OC-DFG layout candidates $C = \{OL'_0, OL'_1, \dots, OL'_{|OT|}\}$, the cost for each candidate is calculated. Then, the layout with the lowest cost is selected and assigned to new OL.

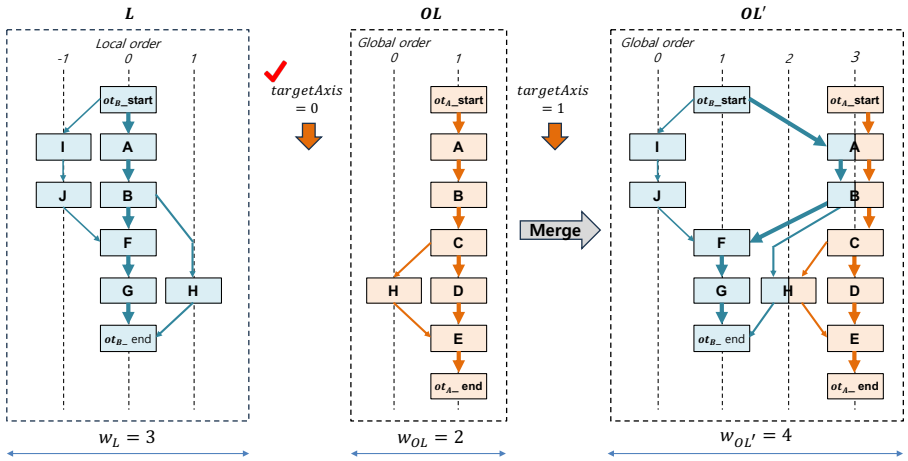


Fig. 3. A graphical explanation of merging the current OC-DFG layout OL with the target object type layout L at the target axis zero.

Definition 6 (Distance between two set of nodes in OC-DFG layout). *Let* $OL = (N, E, OT, OTA, \pi_{rank}, \pi_{order}, \pi_{ota}, \pi_{nota})$ *be an OC-DFG layout. Let* $N_A, N_B \subseteq N$ *be two sets of nodes. The layout-based distance between two sets of nodes is defined as* $dist_{lay}(N_A, N_B) = |avg(\{\pi_{order}(n) \mid n \in N_A\}) - avg(\{\pi_{order}(n) \mid n \in N_B\})|$.

Definition 7 (Distance rank function). *Let* OT *be a set of object types,* N *be a set of nodes,* N_{ot} *be a set of nodes of type* $ot \in OT$, $\pi_{atype} : N \to OT$ *be an object type mapping function, and* $d : N \times N \to \mathbb{F}$ *be a distance function. The distance rank function* $dr_d : OT \times OT \to \mathbb{Z}^+$ *is defined as* $dr_d(ot_A, ot_B) = |\{(ot_i, ot_j) \mid ot_i, ot_j \in OT \wedge ot_i \neq ot_j \wedge d(N_{ot_i}, N_{ot_j}) < d(N_{ot_A}, N_{ot_B})\}|$.

Definition 8 (Cost of an OC-DFG layout). *Let* $OL = (N, E, OT, OTA, \pi_{rank}, \pi_{order}, \pi_{ota}, \pi_{nota})$ *be an OC-DFG layout,* $N_{ot} \in N$ *be a set of nodes of type* $ot \in OT$, $\mathrm{d_{ocel}} : \mathrm{N} \times \mathrm{N} \to \mathbb{F}^+$ *be the distance function defined in Definition 5,* $d_{lay} : N \times N \to \mathbb{F}^+$ *be the distance function defined in Definition 6,* $dr_d : OT \times OT \to \mathbb{Z}^+$ *be the distance rank function over the distance function* d, $ec : \mathbb{G} \to \mathbb{Z}$ *a function that returns the number of edge crossings in graph, and* $\lambda_1, \lambda_2 \in [0,1]$ *are parameters. Cost of the given OC-DFG layout* OL *is defined as:*

$$cost(OL) = \lambda_1 \sqrt{\frac{\sum_{i=1}^{|OT|}\sum_{j=i+1}^{|OT|}\left(dr_{\mathrm{d}_{ocel}}(ot_i, ot_j) - dr_{d_{lay}}(ot_i, ot_j)\right)^2}{\sum_{i=1}^{|OT|}\sum_{j=i+1}^{|OT|} dr_{\mathrm{d}_{ocel}}(ot_i, ot_j)^2}} + \lambda_2 \frac{ec(OL)}{|\mathrm{E}|^2}$$

The first term measures the deviation between the ranked distances of object types in the OCEL and their corresponding ranked distances in the layout. This ensures that object types closely related in the OCEL are also positioned near each other in the layout, while those with larger ranked distances remain farther apart. The second term penalizes layouts with a high number of edge crossings.

Finally, the current layout OL is updated as $OL = argmin_{OL' \in C} cost(OL')$ where $C = \{OL'_0, OL'_1, \dots, OL'_{|OT|}\}$ is the set of OC-DFG layout candidates.

3.6 Cross-minimization

After deriving the OC-DFG layout $OL = (N, E, OT, OTA, \pi_{rank}, \pi_{order}, \pi_{ota}, \pi_{nota})$, the number of edge crossings is minimized as explained in Algorithm 1. The algorithm takes as input OL, a maximum number of iterations $maxIter$, and a function $ec : \mathbb{G} \to \mathbb{Z}_{\geq 0}$ that returns the number of edge crossings. Our method builds upon the edge-crossing minimization techniques proposed in [13, 15], which evaluate whether swapping adjacent nodes within the same rank reduces crossings. We refine this approach by restricting swaps to adjacent nodes of the same object type. This constraint preserves the alignment of nodes along their respective object type axes, enhancing interpretability while optimizing layout quality.

Algorithm 1 *crossMinimization*

Input: $OL, maxIter$
Output: OL_{best}

```
1  OL_best ← OL
2  maxRank ← max_{n∈N} π_order(n)
3  for i ← {1, ..., maxIter} do
4      for r ← {1, ..., maxRank − 1} do
5          currRankN ← sort_order({n | n ∈ N ∧ π_rank(n) = r})
6          postRankN ← sort_order({n | n ∈ N ∧ π_rank(n) = r + 1})
7          for j ← {1, ..., |postRankN| − 1} do
8              n_j, n_{j+1} ← postRankN_j, postRankN_{j+1}
9              if π_ota(n_j) = π_ota(n_{j+1}) do
10                 OL_swap ← swap(OL, n_1, n_2)
11                 if ec(OL_swap) < ec(OL_best) do
12                     OL_best ← OL_swap
13         end
14     end
15 end
16 return OL_best
```

3.7 Node positioning

In this step, the node coordinates are computed as described in Algorithm 2. The algorithm uses $OL = (N, E, OT, OTA, \pi_{rank}, \pi_{order}, \pi_{ota}, \pi_{nota})$, $maxIter$, the set of nodes N_{ot} and edges E_{ot} for each object type. First, the x-coordinates of the nodes are initialized using their orders. Then, the x-coordinates of the non-backbone nodes are updated by calculating the median of their adjacent nodes' x-coordinates. Next, the non-backbone nodes are left-packed within each object type to optimize spacing. Subsequently, backbone nodes undergo the same left-packing.

After computing the x-coordinates, the algorithm evaluates whether the total edge length along the x-axis is reduced. If an improvement is found, the updated x-coordinates are recorded as the optimal values. This process is iteratively repeated for a predefined number of iterations. Finally, the set of x-coordinates of nodes $X = \{x_n \mid n \in N\}$ is determined. From now, we define the function $x : N \to \mathbb{Z}_{\geq 0}$ to obtain x-coordinates of nodes, i.e., $x = \{(n, x_n) \mid n \in N\}$. The y-coordinates of nodes are computed using rank of nodes as $y_n = \pi_{rank}(n) \times k$ where k is a given gap between two adjacent ranks. We define the function $y : N \to \mathbb{Z}_{\geq 0}$ to obtain y-coordinates of nodes, i.e., $y = \{(n, y_n) \mid n \in N\}$.

Algorithm 2 *nodePositioning*

Input: $OL, maxIter, N_{ot}, BN_{ot}$
Output: $xbest$

```
1  xcoord ← initxcoord(OL)
2  xbest ← xcoord
3  for i = {1, ..., maxIter} do
4      for ot ∈ OT do
5          nonBN_ot ← N_ot − BN_ot
6          xcoord ← medianpos(xcoord, nonBN_ot)
7          xcoord ← packcutNonBN(xcoord, nonBN_ot)
8          xcoord ← packcutBN(xcoord, BN_ot)
9          if xlength(xcoord) < xlength(xbest) do
10             xbest ← xcoord
11 return xbest
```

4 Evaluation and result

4.1 Event logs

In the evaluation, we used seven OCELs, as detailed in Table 2.

Table 2. OCEL description

Event log	#event types	#object types	#events	#objects
Ethereum blockchain [16]	26	5	96	93
Hinge manufacturing [17]	11	12	38,528	23,771

Hiring [18]	24	6	18,119	3,768
Hospital [18]	9	8	14,642	3,688
Logistics [19]	14	7	35,372	13,882
Order-to-cash (O2C) [18]	22	9	28,278	8,819
Procure-to-pay (P2P) [18]	10	7	14,671	9,054

4.2 Benchmark

We selected the OC-DFG layout from the web-based tool developed by [6] as a benchmark for its well-organized structure, including node ranking, color-coded object types, and clearly visualized start and end nodes, which closely resembles our proposed layout. For evaluation, we parsed the benchmark layout from its SVG elements, calculating node positions and orders based on x/y coordinates.

4.3 Metrics

For evaluation, we employ seven metrics categorized into two aspects: general layout evaluation and object-centric process layout evaluation. For object-centric process layouts, we propose two new metrics object type compactness and object type interaction preservation. Let $OL = (N, E, OT, OTA, \pi_{rank}, \pi_{order}, \pi_{ota}, \pi_{nota})$ be a finalized OC-DFG layout, N_{ot} be the set of nodes of type $ot \in OT$, $x : N \to \mathbb{Z}_{\geq 0}$ be a node x-coordinate mapping function, and $y : N \to \mathbb{Z}_{\geq 0}$ be a node y-coordinate mapping function.

- Number of back edges (QM_{be}): Measures the flow consistency of the layout, defined as $|\{e \mid e \in E, \pi_{rank}(snode(e)) > \pi_{rank}(enode(e))\}|$
- Number of edge crossings (QM_{ec}): Measures the number of edge intersections.
- Total edge length (QM_{el}): Computes the cumulative length of all edges in OL as: $\sum_{e \in E} \sqrt{\left(x(snode(e)) - x(enode(e))\right)^2 + \left(y(snode(e)) - y(enode(e))\right)^2}$ where $snode : E \to N$ and $enode : E \to N$ return the start and end nodes of the edge.
- Edge orthogonality (QM_{eo}): Measures the alignment of edges to an imaginary Cartesian grid as $1 - \frac{1}{|N|}\sum_{i=1}^{|E|} \min(\theta_i, 90° - \theta_i, 180° - \theta_i)/45°$ where θ_i denotes the angular deviation of the edge e_i from the horizontal or vertical grid lines ($0 \leq QM_{eo} \leq 1$).
- Node orthogonality (QM_{no}): Evaluates the efficiency of rank and order space utilization as $|\{n \mid n \in N\}|/(w \times h)$ where $w = \max(\{\pi_x(n) \mid n \in N\})$ and $h = \max(\{\pi_y(n) \mid n \in N\})$ represent the layout width and height, respectively. ($0 \leq QM_{no} \leq 1$).
- Object type compactness (QM_{OTC}): Measures how well nodes of the same object type are clustered, extending the silhouette method proposed in [20]. The silhouette score is $S_i = (b_i - a_i)/\max(a_i, b_i)$ where a_i is the mean distance to nodes of the same object type, and b_i is the minimum mean distance to nodes of other types. $\alpha_i =$

$\frac{1}{|N_{ot_i}-1|}\sum_{n_a,n_b\in N_{ot_i},a\neq b}|x(n_a)-x(n_b)|$ and $b_i = \min_{ot_i\neq ot_j}\frac{1}{|N_{ot_j}|}\sum_{n_a,n_b\in N_{ot_j}}|x(n_a)-x(n_b)|$. The final score is the mean of S_i, ranging from -1 to 1, where higher values indicate better clustering.

- Object type interaction preservation (QM_{OTIP}): Assesses whether the object types that frequently interact in the event log are positioned close to each other in the layout. For this, we extend Non-Metric Multidimensional Scaling (NMDS) method [21] and quantified as:

$$\sqrt{\frac{\sum_{i=1}^{|OT|}\sum_{j=i+1}^{|OT|}\left(dr_{d_{ocel}}(ot_i,ot_j)-dr_{d_{lay}}(ot_i,ot_j)\right)^2}{\sum_{i=1}^{|OT|}\sum_{j=i+1}^{|OT|}dr_{d_{ocel}}(ot_i,ot_j)^2}}.$$

4.4 Result

The result highlights a trade-off between the general layout and object-centric process layout as summarized in Table 3. While our method has weakness in QM_{ec}, QM_{el}, and QM_{no}, it excels in QM_{be} and QM_{eo}. For OCPM-specific metrics, our method outperforms the benchmark, demonstrating better object type separation and representation of interactions. In other words, the benchmark provides a more compact and aesthetic layout, it does not account for object-centric processes.

Table 3. Overview of quantitative evaluation results

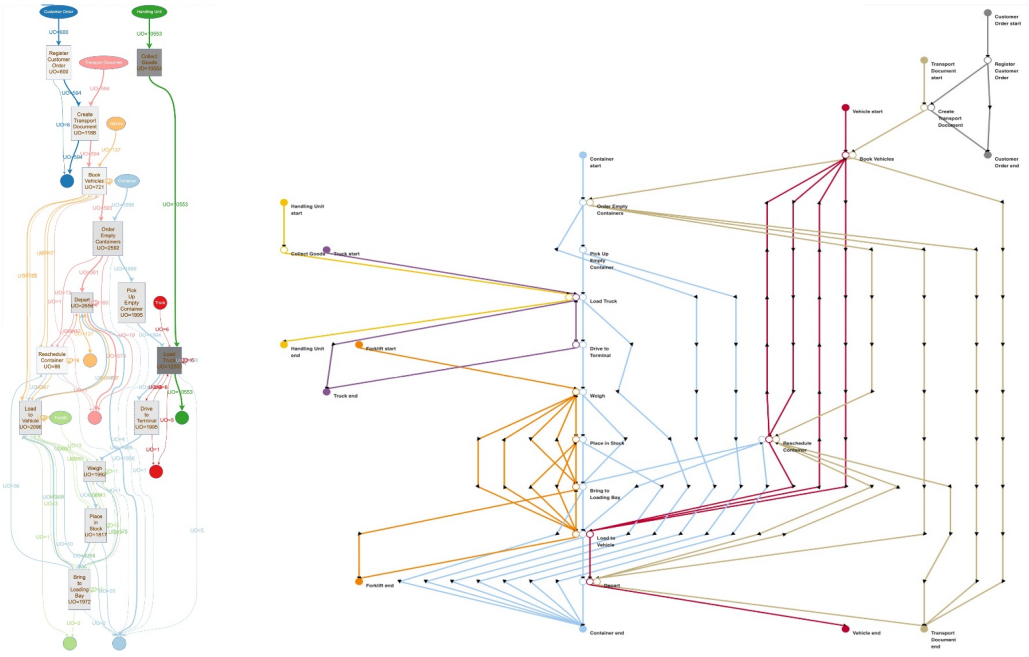
Method	Metric (Count of dominant metrics)						
	Metrics for general layout					Metrics for OCPM	
	QM_{be}	QM_{ec}	QM_{el}	QM_{eo}	QM_{no}	QM_{OTC}	QM_{OTIP}
Our	**7 (-66%)**	0	2	**6 (+0.08)**	0	**7 (+0.29)**	**5 (-0.10)**
Benchmark	0	**7 (-57%)**	**5 (-10%)**	1	**7 (+0.05)**	0	2

For general layout metrics, our method significantly reduces back edges (QM_{be}) by 66% due to its rank assignment, which explicitly minimizes back edges. In contrast, the benchmark prioritizes aesthetic node placement without considering flow consistency. Additionally, our approach improves edge orthogonality (QM_{eo}) by arranging backbones linearly. However, reducing back edges increases edge length (QM_{el}) and node orthogonality (QM_{no}) since additional ranks are needed to reduce back edges. Edge crossings (QM_{ec}) are higher in our method due to object type axis constraints, which require nodes of the same object type to stay together. Additionally, our cross-minimization restricts node swaps within the same object axis, preserving layout consistency but increasing crossings. The benchmark, which does not impose these constraints, achieves fewer crossings but sacrifices object type separation, which makes understanding object interactions harder.

For OCPM-specific metrics, our method outperforms the benchmark. It achieves 29% higher object type compactness (QM_{OTC}) by placing nodes of the same object type closer together. Additionally, it performs better in object type interaction preservation (QM_{OTIP}) by positioning frequently interacting object types closer together while keeping less related types farther apart, preserving the relationships observed in the event log.

4.5 OC-DFG Layouts visualized

We present the OC-DFG layouts generated by both the benchmark and our method for P2P and Logistics OCELs. Fig. 1 shows the layouts for the P2P OCEL, while Fig. 4 illustrates the layouts for the Logistics OCEL.



(a) (b)

Fig. 4. (a) An OC-DFG layout generated by the benchmark and (b) an OC-DFG layout generated by our method using P2P OCEL.

5 Conclusion

In this study, we proposed an advanced OC-DFG layout generation method, assigning a distinct object type axis to each object type to better distinguish object types and reveal their interactions in OC-DFG. While the method has some limitations in general

layout metrics such as edge crossings and node orthogonality, it outperforms the benchmark in other general and OCPM-specific metrics such as back edges, edge orthogonality, object type compactness, and object type interaction preservation.

Beyond academic relevance, the proposed approach may have practical value in domains such as manufacturing, logistics, healthcare, and finance, where understanding multi-object interactions is essential for process optimization and automation. While further validation is required, this structured visualization approach may assist practitioners in analyzing object interactions and interpreting complex processes more effectively in practical settings.

Nevertheless, some limitations remain. Our evaluation was conducted on relatively small OCEL datasets and limited to a comparison with a single benchmark method. Broader validation with large-scale logs and additional tools such as Celonis and PM4Py is necessary. Furthermore, while we introduced several quantitative evaluation metrics—including newly designed ones—we did not provide clear empirical evidence on how these metrics relate to users' actual understanding of the process model. The extent to which improvements in these metrics translate into better interpretability remains to be investigated through user studies. The method also currently focuses solely on OC-DFGs, without support for OC-Petri Nets or OC-BPMN. Lastly, the influence of hyperparameters used in the layout cost function (λ_1 and λ_2) has not yet been analyzed.

For future work, we plan to conduct qualitative user studies to evaluate how the proposed layout supports process comprehension, addressing the current lack of empirical validation. We also aim to extend our benchmark comparisons to include tools like Celonis and PM4Py, and to test our method on larger OCEL datasets to ensure scalability. Additionally, we will analyze the sensitivity of layout quality to cost function parameters (λ_1 and λ_2) and explore extending our approach to support OC-Petri Nets and OC-BPMN to increase applicability in diverse modeling contexts.

Acknowledgments. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT)(RS-2024-00357330).

References

1. Nofal, M.I., Yusof, Z.M.: Integration of Business Intelligence and Enterprise Resource Planning within Organizations. Procedia Technology. 11, 658-665 (2013).
2. Rinderle-Ma, S., Stertz, F., Mangler, J., Pauker, F.: Digital Transformation: Core Technologies and Emerging Topics from a Computer Science Perspective. 1st edn. Springer Vieweg, Berlin, Heidelberg (2023)
3. van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., Alves de Medeiros, A.K., Song, M., Verbeek, H.M.W.: Business process mining: An industrial application. Information Systems. 32, 713-732 (2007).
4. van der Aalst, W.: Object-Centric Process Mining: Dealing with Divergence and Convergence in Event Data. In: Ölveczky, P., Salaün, G. (eds) Software Engineering and Formal Methods. SEFM 2019, LNCS, vol. 11724, pp. 3-25. Springer, Cham (2019)

5. van der Aalst, W.: Object-Centric Process Mining: An Introduction. In: Cerone, A. (eds) Formal Methods for an Informal World. ICTAC 2021, LNCS, vol 13490, pp. 73-105. Springer, Cham (2023)
6. Berti, A., van der Aalst, W.: OC-PM: analyzing object-centric event logs and process models. International Journal on Software Tools for Technology Transfer. 25(1), 1-17 (2023).
7. van der Aalst, W.: Object-Centric Process Mining: Unraveling the Fabric of Real Processes. Mathematics. 11(12), 2691 (2023).
8. van der Aalst, W., Berti, A.: Discovering Object-centric Petri Nets. Fundamenta Informaticae. 175(1-4), 1-40 (2020).
9. Moore, J.: Data visualization in support of executive decision making. Interdisciplinary Journal of Information, Knowledge, and Management. 12, 125 (2017).
10. Park, S., Bekemeier, B., Flaxman, A., Schultz, M.: Impact of data visualization on decision-making and its implications for public health practice: a systematic literature review. Informatics for Health and Social Care. 47(2), 175-193 (2022).
11. Eberhard, K.: The effects of visualization on judgment and decision-making: a systematic literature review. Management Review Quarterly. 73(1), 167-214 (2023).
12. Kim, S.H.: Understanding the role of visualizations on decision making: A study on working memory. Informatics. 7(4), 53(2020).
13. Mennens, R.J.P., Scheepens, R., Westenberg, M.A.: A stable graph layout algorithm for processes. Computer Graphics Forum. 38(3), 725-737 (2019).
14. Object-Centric Event Log 2.0, https://ocel-standard.org, last accessed 2025/03/09
15. Gansner, E.R., Koutsofios, E., North, S.C., Vo, K.P.: A Technique for Drawing Directed Graphs. IEEE Transactions on Software Engineering. 19(3), 214-230 (1993).
16. Comprehensive Ethereum Execution Data for Object-Centric Process Mining of Decentralized Applications (DApps), https://zenodo.org/records/11065366, last accessed 2025/03/09
17. sOCEL 2.0: A Sustainability-Enriched OCEL of a Hinge Production Process, https://zenodo.org/records/13638681, last accessed 2025/03/09
18. Simulated Object-Centric Event Logs (OCEL 2.0) for Order-to-Cash, Procure-to-Pay, Hiring, and Hospital Patient Lifecycle Processes, https://zenodo.org/records/13879980, last accessed 2025/03/09
19. Container Logistics Object-centric Event Log, https://zenodo.org/records/8428084, last accessed 2025/03/09
20. Rousseeuw, P.J.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics. 20, 53-65 (1987).
21. Kruskal, J.B.: Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. Psychometrika. 29(1), 1-27 (1964).