Object-Centric Process Mining: An Introduction

Wil M.P. van der Aalst[0000−0002−0955−6940]

Process and Data Science (PADS), RWTH Aachen University, Germany
wvdaalst@pads.rwth-aachen.de www.vdaalst.com

Abstract. Initially, the focus of process mining was on processes evolving around a single type of objects, e.g., orders, order lines, payments, deliveries, or customers. In this simplified setting, each event refers to precisely one object and the automatically discovered process models describe the lifecycles of the selected objects. Dozens of process-discovery and conformance-checking techniques have been developed using this simplifying assumption. However, real-life processes are more complex and involve objects of multiple types interacting through shared activities. Object-centric process mining techniques start from event logs consisting of events and objects without imposing the classical constraints, i.e., an event may involve multiple objects of possibly different types. This paper introduces object-centric event logs and shows that many of the existing process-discovery and conformance-checking techniques can be adapted to this more holistic setting. This provides many opportunities, as demonstrated by examples and the tool support we developed.

Keywords: Object-Centric Process Mining · Process Discovery · Conformance Checking · Event Data · Process Mining

1 Introduction

More than half of the Fortune 500 companies are already using process mining to analyze and improve their business processes. Process mining starts from event logs extracted from data stored in information systems [1]. Some information systems record explicit audit trails ready to be used for process mining. However, this is the exception. Taking a closer look at contemporary information systems shows that most center around a database composed of dozens, hundreds, or even thousands of database tables. Looking at these tables, one will witness that most of these tables contain timestamps or dates, e.g., when the order was created, when the patient was admitted, when the data were entered into the system, etc. Hence, *event data are everywhere* [5].

According to Gartner, there are now over 40 process mining vendors, e.g., Celonis, Signavio (now part of SAP), ProcessGold (now part of UiPath), Fluxicon, Minit, LanaLabs (now part of Appian), MyInvenio (now part of IBM), QPR, Apromore, Everflow, etc. [30].[1] Most of these systems assume that *each event refers to a single case* and has a *timestamp* and *activity*. This is also the standard assumption made in literature [1]. Also the official IEEE standard for storing event data, called XES (eXtensible Event Stream) [28], makes this assumption.

[1] See the website www.processmining.org for an up-to-date overview of existing tools.

It is very natural to assume that an event has indeed a timestamp and refers to an activity: What happened and when did it happen? However, the assumption that each event refers to precisely one case may cause problems [3]. *Object-Centric Event Logs (OCEL)* aim to overcome this limitation [26]. In OCEL, an event may refer to any number of objects (of different types) rather than a single case. Object-centric process mining techniques may produce Petri nets with different types of objects [8] or artifact-centric process models [21, 22, 35].

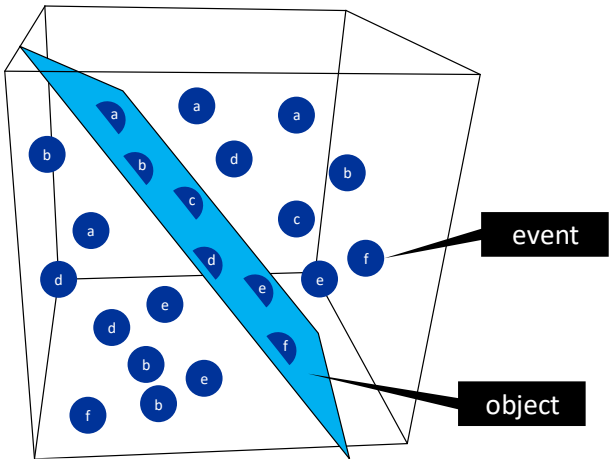

Fig. 1. An event may refer to any number of objects and the same object may be involved in multiple events. The classical case notion assumes that each event refers to a single case. Object-centric process mining drops this assumption to analyze and improve a much larger class of processes.

Large-scale process mining users, vendors, and researchers acknowledge that the case-notation is limiting the application of process mining. Also, the preprocessing of raw data to create event data satisfying the requirements may take up to 80% of the time. One can use the following metaphor to explain the problem. Existing techniques use *two-dimensional* (2D) event logs and models. In object-centric process mining, we use *three-dimensional* (3D) event logs and models (see Figure 1). The first two dimensions are activities and time while considering only one type of objects. Object-centric process mining adds a third dimension because multiple object types are considered and one event may refer to any number of objects.

For people familiar with *Colored Petri Nets* (CPNs) [9, 29], it is fairly easy to imagine a process model with multiple types of interacting objects. In a CPN tokens residing in places may represent different types of objects (products, people, machines, storage locations, etc.). Transitions in a CPN may connect places of different types and can thus represent activities involving multiple types of objects. However, it is far from trivial to discover such models. Traditional process mining approaches assume that each event

refers to a single case. As a result, cases can be considered *independent* of each other. This leads to process models that describe the lifecycle of a case *in isolation*.

A range of process discovery approaches using the "single-case assumption" have been proposed [4]. These can be grouped in "bottom-up" approaches like the Alpha algorithm [1, 10, 13, 14, 16, 38, 41] and 'top-down" approaches like the inductive mining approaches [31–33]. See [12] for a recent survey of process discovery techniques.

There are many techniques for conformance checking, but most of these also make the "single-case assumption". The two most frequently used approaches are *token-based replay* [37] and *alignments* [6, 17]. The approach in [24] is a notable exception using so-called *proclets* [7]. Proclets and most of the artifact-centric approaches [22] assume that process instances (case, artifact, proclet, etc.) are interacting. Object-centric process mining assumes that events share objects, i.e., objects are not interacting as autonomous entities, but may be involved in common events. This simplifies the problem as is illustrated by the recently developed object-centric process mining techniques and tools, the OCEL standard, and the simplicity of the definitions presented in this tutorial paper.

The advantages of using object-centric process mining are threefold:

- *Data extraction is only done once.* There is no longer a need to pick a case notion when extracting data. The extraction of objects and events does not depend on the viewpoint.
- *Relations between objects of possibly multiple types are captured and analyzed.* In traditional process mining (with a focus on individual cases), interactions between objects are abstracted away.
- *Object-centric process models provide a three-dimensional view of processes and organizations.* In one process model multiple object types and their relations are visualized. Moreover, by selecting subsets of object types and activities, it is possible to change viewpoint and get new insights.

The goal of this paper is not to explain a specific process-discovery or conformance-checking technique. Instead, we provide a *tutorial-style introduction* to the topic and will show that there are a few general principles that allow lifting process mining techniques from the "single-case" to the "multi-object" level. This paper is based on a tutorial and keynote given in the context of *18th International Colloquium on Theoretical Aspects of Computing* (ICTAC 2021) in September 2021.

The remainder is organized as follows. Section 2 introduces the input needed for object-centric process mining, followed by a brief introduction to process discovery and conformance checking for event logs assuming a single case notion (Section 3). Section 4 presents a baseline approach for object-centric process discovery building upon existing techniques. The topic of object-centric conformance checking is covered in Section 5. Process mining is not limited to process discovery and conformance checking. Hence, Section 6 briefly discusses the impact of object-centricity on other types of analysis. Section 7 discusses concerns related to the complexity caused by looking at many object types and activities. Section 8 concludes the paper.

2 Object-Centric Event Data

Traditional process mining approaches assume that each event is related to precisely one object (called case). This tutorial paper shows that *relaxing* this requirement allows for a major step forward in data-driven process analytics. For example, we can now look at processes and organizations *from any angle* using a *single source of truth*, i.e., there is no need to extract new data when changing the viewpoint of analysis. Looking at different types of objects at the same time allows us to discover novel and valuable insights that live at the intersection points of processes and departments. The transition from traditional process mining to object-centric process mining is similar to moving from 2D to 3D. Before we discuss novel 3D process mining techniques, this section first introduces *object-centric event logs*, i.e., collections of *related events and objects*.

An event e can have any number of attributes. $\#_x(e)$ is the value of attribute x for event e. We require the following two attributes to be present for any event: activity $\#_{act}(e)$, and timestamp $\#_{time}(e)$. An object o can also have any number of attributes, but these may change over time. For example, object o represents a specific patient that has a birth date that does not change, but also has an address, phone number, weight, and blood pressure that may change. $\#^t_x(o)$ is the value of attribute x for object o at time t. There is a mandatory object attribute $type$ with value $\#^t_{type}(o)$ representing the type of object. $\#^t_{type}(o)$ does not change over time, so we can also write $\#_{type}(o)$. There is a many-to-many relation R between events and objects. $(e, o) \in R$ if and only if object o is involved in event e. To formalize object-centric event logs, we first introduce several universes.

Definition 1 (Universes). *$\mathcal{U}_{ev}$ is the universe of events, $\mathcal{U}_{obj}$ is the universe of objects ($\mathcal{U}_{ev} \cap \mathcal{U}_{obj} = \emptyset$), $\mathcal{U}_{act}$ is the universe of activities, $\mathcal{U}_{time}$ is the universe of timestamps, $\mathcal{U}_{type}$ is the universe of object types, $\mathcal{U}_{att} = \{act, time, type, \ldots\}$ is the universe of attributes, $\mathcal{U}_{val}$ is the universe of values, and $\mathcal{U}_{map} = \mathcal{U}_{att} \not\to \mathcal{U}_{val}$ is the universe of attribute-value mappings. We assume that $\mathcal{U}_{act} \cup \mathcal{U}_{time} \cup \mathcal{U}_{type} \subseteq \mathcal{U}_{val}$ and $\bot \notin \mathcal{U}_{val}$. For any $f \in \mathcal{U}_{map}$, if $x \notin dom(f)$, we write $f(x) = \bot$ to denote that there is no attribute value.*

Using these universes, we define an event log as follows.

Definition 2 (Event Log). *An event log is a tuple $L = (E, O, \#, R)$ consisting of a set of events $E \subseteq \mathcal{U}_{ev}$, a set of objects $O \subseteq \mathcal{U}_{obj}$, a mapping $\# \in (E \to \mathcal{U}_{map}) \cup (O \to (\mathcal{U}_{time} \to \mathcal{U}_{map}))$, and a relation $R \subseteq E \times O$, such that for all $e \in E$, $\#(e)(act) \in \mathcal{U}_{act}$ and $\#(e)(time) \in \mathcal{U}_{time}$, and for all $o \in O$ and $t, t' \in \mathcal{U}_{time}$, $\#(o)(t)(type) = \#(o)(t')(type) \in \mathcal{U}_{type}$.*

An event log consists of a set of events E and a set of objects O connected through R. Events have an activity and a timestamp, and objects have a type. To make the notation more intuitive, we use $\#_x$ or $\#^t_x$ to refer to attribute values. Formally: $\#_x(e) = \#(e)(x)$ and $\#^t_x(o) = \#(o)(t)(x)$, e.g., $\#_{act}(e)$ is the activity of event e, $\#_{time}(e)$ is the timestamp of event e, $\#^t_{type}(o)$ is the type of object o at time t. Since the type of an object cannot change, we can drop t and write $\#_{type}(o)$.

Events are ordered by their timestamps. Without loss of generality, we can assume that there is a total order on events, i.e., we assume an arbitrary (but fixed) order for events that have the same timestamp.

Definition 3 (Event Order). *Let $L = (E, O, \#, R)$ be an event log. $\prec_L \subset E \times E$ is a total order such that for any pair of events $e_1, e_2 \in E$: $e_1 \prec_L e_2$ implies $e_1 \neq e_2$ and $\#_{time}(e_1) \leq \#_{time}(e_2)$.*

Figure 2 shows a visualization of an event log. The black circles represent events labeled with the corresponding activity. Objects flow through the graph from start ▶ to end ■ grouped and colored per object type. Exploiting the total order $\prec_L$ we can think of objects as a "line" visiting different events, i.e., pick an object o and the corresponding events visited by o form a sequence. Note that all objects entering an event are also exiting an event and vice versa.

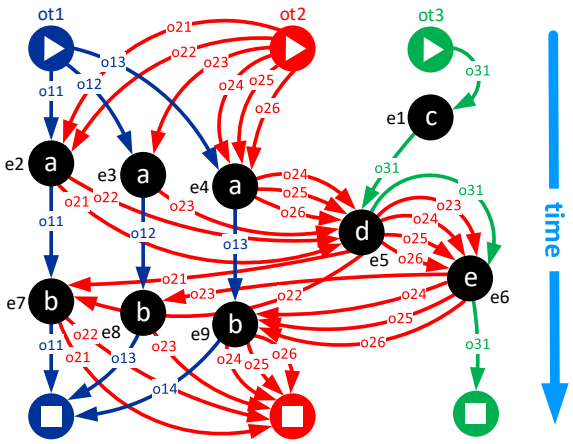

Fig. 2. A visualization of an event log $L = (E, O, \#, R)$ with nine events $E = \{e1, e2, \ldots, e9\}$, ten objects $O = \{o11, o12, \ldots, o31\}$, three object types (e.g., $\#_{type}(o11) = ot1$, $\#_{type}(o21) = ot2$, and $\#_{type}(o31) = ot3$), and five activities a, b, c, d, and e. The connections show the flow of objects. $R = \{(e1, o31), (e2, o11), (e2, o21), (e2, o22), (e3, o12), (e3, o23), (e4, o13), (e4, o24), (e4, o25), (e4, o26), \ldots, (e9, o26)\}$ relates events and objects.

The formalization $L = (E, O, \#, R)$ abstracts from the way the information is stored. A possible realization to store event data would be to have three database tables:

- An *event table* with one row per event and a column for each event attribute (e.g., activity and timestamp and optional attributes like costs), next to the event identifier as the primary key.
- An *object update table* with one row per object update and a column for each object attribute (including object type), next to the object identifier and timestamp (the combination of the latter two is the primary key).

– A *relation table* connecting event identifiers and object identifiers.

The *Object-Centric Event Log (OCEL)* standard [26] provides two storage formats for such data: *JSON-OCEL* and *XML-OCEL* (cf. `ocel-standard.org`). The only difference between our formalization and OCEL is that OCEL assumes that objects have fixed attributes. The idea is that a new object is created if its attributes change. Moreover, OCEL types attributes, e.g., string, integer, float, and timestamp. In this paper, we introduce object-centric process mining at a conceptual level, not assuming a particular syntax. However, our tools use OCEL as a concrete storage format.

In the remainder, we will use the following functions to query event logs and to extract the "path" of a particular object.

Definition 4 (Notations). *For an event log $L = (E, O, \#, R)$, we introduce the following notations.*

- $act(L) = \{\#_{act}(e) \mid e \in E\}$ *is the set of activities,*
- $types(L) = \{\#_{type}(o) \mid o \in O\}$ *is the set of object types,*
- $events(o) = \{e \in E \mid (e, o) \in R\}$ *are the events containing object $o \in O$,*
- $objects(e) = \{o \in O \mid (e, o) \in R\}$ *are the objects involved in event $e \in E$,*
- $seq(o) = \langle e_1, e_2, \ldots, e_n \rangle$ *such that* $events(o) = \{e_1, e_2, \ldots, e_n\}$ *and* $e_i \prec_L e_j$ *for any* $1 \leq i < j \leq n$ *is the sequence of events where object $o \in O$ was involved in,*
- $trace(o) = \langle a_1, a_2, \ldots, a_n \rangle$ *such that* $seq(o) = \langle e_1, e_2, \ldots, e_n \rangle$ *and* $a_i = \#_{act}(e_i)$ *for any* $1 \leq i \leq n$ *is the trace of object $o \in O$.*

Let $L = (E, O, \#, R)$ correspond to the visualization in Figure 2. $act(L) = \{a, b, c, d, e\}$, $types(L) = \{ot1, ot2, ot3\}$, $events(o11) = \{e2, e7\}$, $events(o21) = \{e2, e5, e7\}$, $events(o26) = \{e4, e5, e6, e9\}$, $objects(e1) = \{o11, o21, o22\}$, $objects(e4) = \{o13, o24, o25, o26\}$, $seq(o11) = \langle e2, e7 \rangle$, $seq(o21) = \langle e2, e5, e7 \rangle$, $seq(o26) = \langle e4, e5, e6, e9 \rangle$, $trace(o11) = \langle a, b \rangle$, $trace(o21) = \langle a, d, b \rangle$, and $trace(o26) = \langle a, d, e, b \rangle$. Functions seq and $trace$ determine the path of an object: the former returns a sequence of events and the latter a sequence of activities.

3 Traditional Process Mining

As mentioned in the introduction, most process mining techniques assume that each event corresponds to precisely one case. This implies that there is just one object type *case* and relation R is a function. In terms of the visualization in Figure 2 this means that every event has precisely one ingoing and one outgoing arc. This implies that an event log can be seen as a *multiset of traces* where each *trace is a sequence of activities* corresponding to a specific case.

Figure 3 visualizes the lifecycle of each object. There are ten objects $O = \{o11, o12, \ldots, o31\}$ distributed over three object types. When we are forced to pick a single case notion, we pick an object type and use the corresponding subset of sequences depicted in Figure 3. Note that the same event may appear in different lifecycles leading to the so-called convergence problem.

In this section, we show how to extract a classical event log from an object-centric event log and discuss filtering, basic process discovery, and conformance checking starting from a multiset of traces.

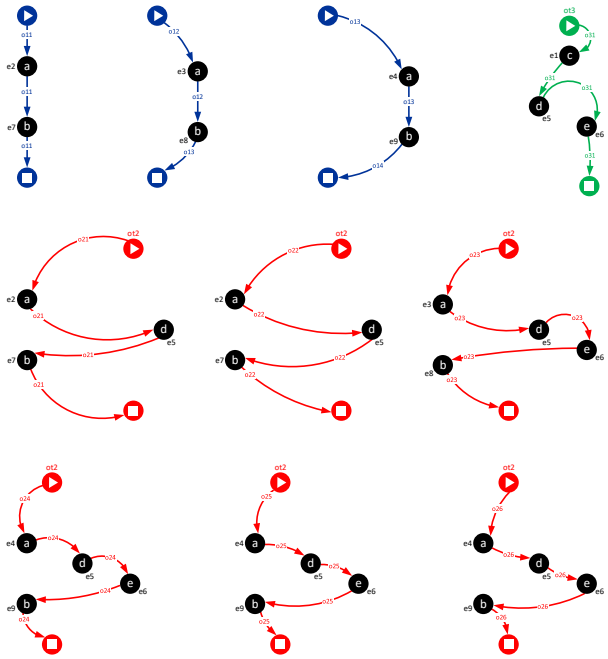

Fig. 3. A visualization of the lifecycle of each object based on Figure 2. Note that some events are replicated, e.g. $e4$ appears four times, because $objects(e4) = \{o13, o24, o25, o26\}$. Due to the total ordering of events, each object corresponds to a sequence of activities, e.g., $trace(o11) = \langle a, b \rangle$, $trace(o21) = \langle a, d, b \rangle$, and $trace(o26) = \langle a, d, e, b \rangle$.

3.1 Preliminaries

In the remainder, we use basic mathematical notations, including sequences, sets, multisets, and functions. $\sigma = \langle x_1, x_2, \ldots, x_n \rangle \in X^*$ is a sequence of length n over a set X. $\mathcal{P}(X) = \{Y \mid Y \subseteq X\}$ is the powerset of X, i.e., all subsets of X. $f \in X \not\to Y$ is a partial function with domain $dom(f)$ and range $rng(f) = \{f(x) \mid x \in dom(f)\}$. $f \in X \to Y$ is a total function with domain $dom(f) = X$. $\mathcal{B}(X) = X \to \mathbf{N}$ is the set of all multisets over X. We use square brackets to denote concrete multisets, e.g., $X = [a^2, b^7, c] \in \mathcal{B}(\{a, b, c\})$ is a multiset with 10 elements such that $X(a) = 2$, $X(b) = 7$, and $X(c) = 1$. As usual, multisets can be mapped to ordinary sets if needed. Some examples assuming $X = [a^2, b^7, c]$: $\{x \in X\} = \{a, b, c\}$, $X \cup [c^2, d^5] = [a^2, b^7, c^3, d^5]$, $X \cap [b^2, c^5, d^3] = [b^2, c]$, $[f(x) \mid x \in X] = [(f(a))^2, (f(b))^7, f(c)]$, $\sum_{x \in X} f(x) = 2f(a) + 7f(b) + f(c)$.

3.2 Creating a Classical Event Log for a Specific Object Type

To relate object-centric event logs to classical case-centric event logs, we can simply pick a single object type and then create a trace for each object using the function $trace$ introduced before. Note that multiple objects may have the same trace, therefore we get

a multiset of traces. Events need to be replicated if they refer to multiple objects of the selected type. Consider, for example, Figure 3. The bottom six object lifecycles define the simple event log $[\langle a,d,b\rangle^2, \langle a,d,e,b\rangle^4]$ when we consider only activities.

Definition 5 (Simple Event Log). *For an event log $L = (E, O, \#, R)$ and object type $ot \in types(L)$, we create a simple event log $L_{ot} = [trace(o) \mid o \in O \wedge \#_{type}(o) = ot] \in \mathcal{B}(\mathcal{U}_{act}{}^*)$.*

L_{ot} is obtained by considering the activity paths of all objects of the selected type. Again we use the event log L in Figure 2 to illustrate this. $types(L) = \{ot1, ot2, ot3\}$ $L_{ot1} = [trace(o11), trace(o12), trace(o13)] = [\langle a,b\rangle, \langle a,b\rangle, \langle a,b\rangle] = [\langle a,b\rangle^3]$. $L_{ot2} = [trace(o21), \ldots trace(o26)] = [\langle a,d,b\rangle, \ldots \langle a,d,e,b\rangle] = [\langle a,d,b\rangle^2, \langle a,d,e,b\rangle^4]$. $L_{ot3} = [trace(o31)] = [\langle c,d,e\rangle]$. Note that in L_{ot2} event $e5$ is replicated six times and $e6$ is replicated four times.

3.3 Activity and Variant-Based Filtering

A simple event log $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ may contain frequent and infrequent behaviors. Process mining may focus on the mainstream behavior or on exceptional behaviors. Therefore, we need to be able to filter the event log. We may want to remove infrequent activities or variants. To describe these two types of filtering, we introduce projection and ranking functions.

Definition 6 (Projection Functions). *Let $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ be an event log, $A \subseteq \mathcal{U}_{act}$ a set of activities, and $V \subseteq \mathcal{U}_{act}{}^*$ a set of variants (i.e. traces).*

- *$L{\Uparrow}V = [\sigma \in L \mid \sigma \in V]$ is the projection of L on V, and $L{\Uparrow}\overline{V} = [\sigma \in L \mid \sigma \notin V]$ removes all variants in V.*
- *$\sigma{\uparrow}A$ projects a trace $\sigma \in \mathcal{U}_{act}{}^*$ on A, i.e., $(\sigma \cdot \langle a\rangle){\uparrow}A = \sigma{\uparrow}A \cdot \langle a\rangle$ if $a \in A$, and $(\sigma \cdot \langle a\rangle){\uparrow}A = \sigma{\uparrow}A$ if $a \notin A$. $\sigma{\uparrow}\overline{A}$ removes all activities in A from trace $\sigma \in \mathcal{U}_{act}{}^*$, i.e., $\sigma{\uparrow}\overline{A} = \sigma{\uparrow}(\mathcal{U}_{act} \setminus A)$.*
- *$L{\uparrow}A = [\sigma{\uparrow}A \mid \sigma \in L]$ is the projection of L on A and $L{\uparrow}\overline{A} = [\sigma{\uparrow}\overline{A} \mid \sigma \in L]$ removes all A events.*

Consider the example log $L = [\langle a,b,c,d\rangle^{10}, \langle a,c,b,d\rangle^5, \langle a,e,d\rangle^3]$. $L{\Uparrow}\{\langle a,e,d\rangle\} = [\langle a,e,d\rangle^3]$. $L{\Uparrow}\overline{\{\langle a,e,d\rangle\}} = [\langle a,b,c,d\rangle^{10}, \langle a,c,b,d\rangle^5]$. $L{\uparrow}\{a,d\} = [\langle a,d\rangle^{18}]$. $L{\uparrow}\overline{\{a,d\}} = [\langle b,c\rangle^{10}, \langle c,b\rangle^5, \langle e\rangle^3]$. Next, we rank the activities and variants based on their frequency.

Definition 7 (Ranking Functions). *Let $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ be an event log.*

- *$afreq_L(A) = \sum_{\sigma \in L{\uparrow}A} |\sigma|$ counts the number of $A \subseteq \mathcal{U}_{act}$ events in event log L.*
- *$vfreq_L(V) = |L{\Uparrow}V|$ counts the number of $V \subseteq \mathcal{U}_{act}{}^*$ traces in event log L.*
- *$act(L) = \cup_{\sigma \in L}\{a \in \sigma\}$ are the activities in L.*
- *$var(L) = \{\sigma \in L\}$ are the variants in L.*
- *$arank(L) = \langle a_1, a_2, \ldots, a_n\rangle \in \mathcal{U}_{act}{}^*$ such that $afreq_L(\{a_i\}) \geq afreq_L(\{a_j\})$ and $a_i \neq a_j$, for any $1 \leq i < j \leq n$ and $\{a_1, a_2, \ldots, a_n\} = act(L)$.*

- $vrank(L) = \langle \sigma_1, \sigma_2, \ldots, \sigma_m \rangle \in ({\mathcal{U}_{act}}^*)^*$ *such that* $vfreq_L(\{\sigma_i\}) \geq vfreq_L(\{\sigma_j\})$ *and* $\sigma_i \neq \sigma_j$*, for any* $1 \leq i < j \leq m$ *and* $\{\sigma_1, \sigma_2, \ldots, \sigma_m\} = var(L)$.

There may be activities and variants that have the same frequency. In this case, we rank these in some fixed order, e.g., alphabetically. For event log $L = [\langle a, b, c, d \rangle^{10}, \langle a, c, b, d \rangle^5, \langle a, e, d \rangle^3]$, we have $arank(L) = \langle a, d, b, c, e \rangle$ and $vrank(L) = \langle \langle a, b, c, d \rangle, \langle a, c, b, d \rangle, \langle a, e, d \rangle \rangle$. The ranking of activities and variants can be used to select the most frequent activities and variants covering a given percentage $cov \in [0, 1]$ of the events or traces in the event log.

Definition 8 (Filtering Functions). *Let* $L \in \mathcal{B}({\mathcal{U}_{act}}^*)$ *be an event log with* $arank(L) = \langle a_1, a_2, \ldots, a_n \rangle$ *and* $vrank(L) = \langle \sigma_1, \sigma_2, \ldots, \sigma_m \rangle$*. Given a threshold* $cov \in [0, 1]$*, we can filter the log as follows.*

- $afilter_L(cov) = L{\uparrow}\{a_1, a_2, \ldots, a_k\}$ *where* $1 \leq k \leq n$ *is the smallest number such that* $afreq_L(\{a_1, a_2, \ldots, a_k\}) \geq cov \times afreq_L(act(L))$.
- $vfilter_L(cov) = L{\Uparrow}\{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ *where* $1 \leq k \leq m$ *is the smallest number such that* $vfreq_L(\{\sigma_1, \sigma_2, \ldots, \sigma_k\}) \geq cov \times vfreq_L(var(L))$.

Again we use event log $L = [\langle a, b, c, d \rangle^{10}, \langle a, c, b, d \rangle^5, \langle a, e, d \rangle^3]$ to illustrate the two filtering functions. $afilter_L(0.5) = [\langle a, d \rangle^{18}]$ (covering 36 of 69 events, i.e., 52%), $afilter_L(0.8) = [\langle a, b, c, d \rangle^{10}, \langle a, c, b, d \rangle^5, \langle a, d \rangle^3]$ (covering 66 of 69 events, i.e., 95%), $vfilter_L(0.5) = [\langle a, b, c, d \rangle^{10}]$ (covering 10 of 18 traces, i.e., 55%), and $vfilter_L(0.8) = [\langle a, b, c, d \rangle^{10}]$ (covering 15 of 18 traces, i.e., 83%).

Most process mining tools provide sliders to seamlessly simply models by leaving out infrequent activities and/or variants.

3.4 Discovering a Directly-Follows Graph (DFG)

Dozens of process discovery techniques are able to learn process models from event data based on the "multiset of traces" abstraction, i.e., event logs of the form $L \in \mathcal{B}({\mathcal{U}_{act}}^*)$. These process models provide a representation describing a set of traces, i.e., they describe behavior of the form $B \subseteq {\mathcal{U}_{act}}^*$. Examples include "bottom-up" approaches like the Alpha algorithm [1, 10] producing Petri nets and 'top-down" approaches like the inductive mining approaches [31–33] producing process trees. Both Petri nets and process trees are able to describe concurrency and can be translated into *Business Process Model and Notation* (BPMN) models [20, 36]. There are also approaches not able to uncover concurrency. These approaches typically produce *Directly-Follows Graphs* (DFGs).

DFGs simply visualize the "directly-follows" relation between activities. This typically leads to underfitting process models [2]. When activities appear out of sequence, loops are created, thus leading to Spaghetti-like diagrams suggesting repetitions that are not supported by the data. However, DFGs are easy to compute, also in a distributed manner, using a single pass through the event data [1]. This explains why they are widely used.

Definition 9 (Directly Follows Graph). *A Directly Follows Graph (DFG)* $G = (A, F)$ *is composed of a multiset of activities* $A \in \mathcal{B}(\mathcal{U}_{act})$ *and a set of weighted edges* $F \in \mathcal{B}((A \cup \{\blacktriangleright\}) \times (A \cup \{\blacksquare\}))$ *such that* $\{\blacktriangleright, \blacksquare\} \cap A = \emptyset$ *and for any* $a \in A$*:* $|[b \mid (b, a) \in F]| = |[b \mid (a, b) \in F]| = A(a)$ *and there exists a path* $\sigma = \langle \blacktriangleright, a_1, \ldots a_n, \blacksquare \rangle$ *such that* $(a_i, a_{i+1}) \in F$ *for* $1 \leq i < n$ *and* $a \in \{a_1, \ldots a_n\}$.

In a DFG $G = (A, F)$, A represents the multiset of activities, i.e., $A(a)$ denotes the frequency of activity $a \in A$. Recall that $\mathcal{B}(\mathcal{U}_{act}) = \mathcal{U}_{act} \rightarrow \mathbb{N}$. F is also a multiset. $F(a, b)$ is the frequency of the connection between activities a and b. The process starts with a symbolic activity $\blacktriangleright$ and ends with a symbolic activity $\blacksquare$. Therefore, each regular activity $a \in A$ needs to be on a path from $\blacktriangleright$ to $\blacksquare$. Finally, the frequency of activity a should match the sum of the frequencies on the input arcs and should also match the sum of the frequencies on the output arcs. This is expressed by the requirement $|[b \mid (b, a) \in F]| = |[b \mid (a, b) \in F]| = A(a)$ for any regular activity a. Note that this implies that $|[a \mid (\blacktriangleright, a) \in F]| = |[a \mid (a, \blacksquare) \in F]| > 0$, i.e., the frequencies of the symbolic start and end match, because the rest of the graph is balanced.

Given an event log L, we can create the corresponding DFG $G(L) = (A, F)$ as follows.

Definition 10 (Directly Follows Graph for an Event Log). *For a simple event log* $L \in \mathcal{B}({\mathcal{U}_{act}}^*)$*, the Directly Follows Graph (DFG)* $G(L) = (A, F)$ *is composed of a multiset of activities* $A = \bigcup_{\sigma \in L} [a \in \sigma]$*, a start activity* $\blacktriangleright \notin A$*, an end activity* $\blacksquare \notin A$*, a flow relation* $F = [(a_i, a_{i+1}) \mid \sigma = \langle a_1, a_2, \ldots, a_n \rangle \in L \wedge a_0 = \blacktriangleright \wedge a_{n+1} = \blacksquare \wedge 0 \leq i \leq n]$.

It is easy to see that the construction in Definition 10 leads to a DFG satisfying the requirements in Definition 9.

Figure 4 shows three DFGs based on the object-centric event log in Figure 2. There is one DFG for each of the object types. Using Definition 5 the following three simple event logs are created: $L_{ot1} = [\langle a, b \rangle^3]$, $L_{ot2} = [\langle a, d, b \rangle^2, \langle a, d, e, b \rangle^4]$, and $L_{ot3} = [\langle c, d, e \rangle]$. Using Definition 10, a DFG is created for each of these logs. $G_{ot1} = (A_{ot1}, F_{ot1})$ with $A_{ot1} = [a^3, b^3]$ and $F_{ot1} = [(\blacktriangleright, a)^3, (a, b)^3, (b, \blacksquare)^3]$ is the DFG computed for object type *ot1*. $G_{ot2} = (A_{ot2}, F_{ot2})$ with $A_{ot2} = [a^6, b^6, d^6, e^4]$ and $F_{ot2} = [(\blacktriangleright, a)^6, (a, d)^6, (d, b)^2, (d, e)^4, (e, b)^4, (b, \blacksquare)^6]$ is the DFG computed for object type *ot2*. $G_{ot3} = (A_{ot3}, F_{ot3})$ with $A_{ot3} = [c, d, e]$ and $F_{ot3} = [(\blacktriangleright, c), (c, d), (d, e), (e, \blacksquare)]$ is the DFG computed for object type *ot3*.

Definition 11 (Accepting Traces of a DFG). *Let* $G = (A, F)$ *be a DFG.* $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ *is an accepting trace of* G *if* $(\blacktriangleright, a_1) \in F$*,* $(a_n, \blacksquare) \in F$*, and* $(a_i, a_{i+1}) \in F$ *for* $1 \leq i < n$*.* $lang(G)$ *is the set of all accepting traces.*

The three DFGs in Figure 4 are acyclic and therefore have a finite number of accepting traces: $lang(G_{ot1}) = \{\langle a, b \rangle\}$, $lang(G_{ot2}) = \{\langle a, d, b \rangle, \langle a, d, e, b \rangle\}$, and $lang(G_{ot3}) = \{\langle c, d, e \rangle\}$. DFGs cannot capture concurrency. To illustrate this, we first introduce *accepting Petri nets*.

Note that activity-based filtering using $afilter_L(cov)$ and variant-based filtering using $vfilter_L(cov)$ can be applied to the event log before computing the DFG. In our

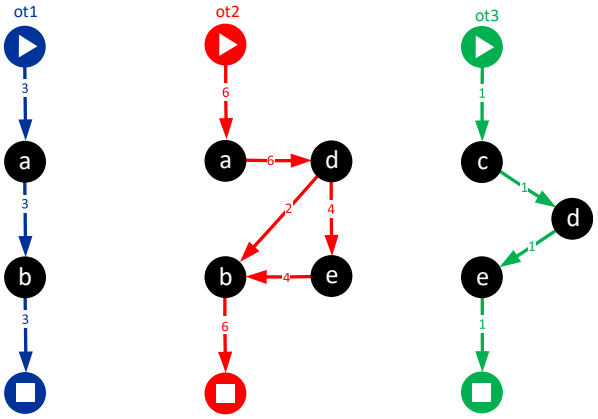

Fig. 4. Three Directly Follows Graphs (DFG) based on the simple event logs $L_{ot1} = [\langle a, b\rangle^3]$, $L_{ot2} = [\langle a, d, b\rangle^2, \langle a, d, e, b\rangle^4]$, and $L_{ot3} = [\langle c, d, e\rangle]$.

formalization $G = (A, F)$, the edges in a DFG have frequencies (F is a multiset). This can also be used for filtering. However, frequencies do not affect the set of accepting traces $lang(G)$. Hence, they are also ignored for conformance checking. It is assumed that the DFG only contains the relevant directly-follows relations.

3.5 Accepting Petri Net

There exist many different process modeling notations able to capture concurrency, e.g., UML activity diagram, UML statecharts, Petri nets, process algebras, BPMN (Business Process Model and Notation) models, etc. The Petri net formalism provides a graphical but also formal language and was the first formalism to adequately capture concurrency. See [18, 19] for a more extensive introduction. Other notations, like BPMN and UML activity diagrams, often have semantics involving "playing the token game" and build on Petri net concepts.

Because traces in an event log have a clear start and end, we use *accepting* Petri nets.

Definition 12 (Accepting Petri Net). *An accepting Petri net is a triplet $AN = (N, M_{init}, M_{final})$ where $N = (P, T, F, l)$ is a labeled Petri net, $M_{init} \in \mathcal{B}(P)$ is the initial marking, and $M_{final} \in \mathcal{B}(P)$ is the final marking. A labeled Petri net is a tuple $N = (P, T, F, l)$ with P the set of places, T the set of transitions, $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ the flow relation, and $l \in T \not\rightarrow \mathcal{U}_{act}$ a labeling function.*

The left-hand side of Figure 5 shows an accepting Petri net AN_1. There are eight places $P = \{p1, p2, p3, p4, p5, p6, p7, p8\}$ (represented by circles) and six transitions $T = \{t1, t2, t3, t4, t5, t6\}$ (represented by squares). The flow relation $F = \{(p1, t1), (t1, p2), (t1, p3), \ldots, (t6, p8)\}$ list the connections between places and transitions. The labeling function is a partial function, i.e., there may be transitions without a label. However, in AN_1 all transitions have a label: $l(t1) = a, l(t2) = b, l(t3) = c, l(t4) = d,$

$l(t5) = e$, and $l(t6) = f$. The accepting Petri net in Figure 5 has an initial marking $M_{init} = [p1]$ and a final marking $M_{final} = [p8]$.

States in Petri nets are called *markings* and mark certain places with *tokens* (represented by black dots). Formally, a marking M is a multiset of places, i.e., $M \in \mathcal{B}(P)$. Tokens in the initial marking are denoted using small triangles ▶ and tokens in the final marking are denoted using small squares ■. A transition is called *enabled* if each of the input places has a token. An enabled transition may *fire* (i.e., occur), thereby consuming a token from each input place and producing a token for each output place. In the initial marking $M_{init} = [p1]$, transition $t1$ can fire resulting in marking $[p2, p3, p4]$. In this marking, three transitions are enabled: $t2$, $t3$, and $t4$. Firing $t4$ results in marking $[p2, p3, p7]$ enabling $t2$, $t3$, and $t5$. After also firing $t2$ and $t3$, we reach marking $[p5, p6, p7]$. In this marking, there is a choice between $t5$ and $t6$. Firing $t6$ results in the final marking $M_{final} = [p8]$.

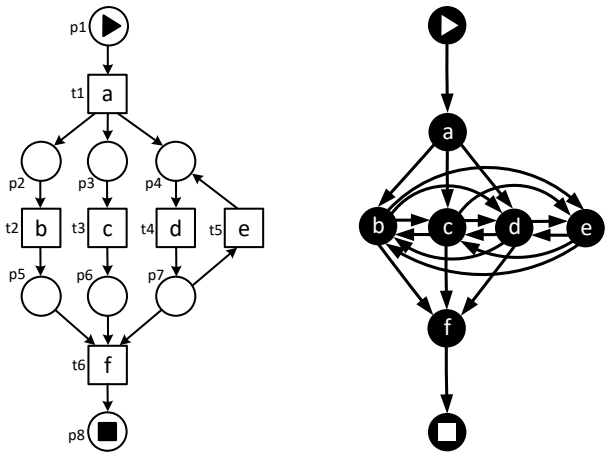

Fig. 5. Accepting Petri net AN_1 (left) and DFG G_1 (right).

A *firing sequence* is a sequence of transition occurrences obtained by firing enabled transitions and moving from one marking to the next. A *complete* firing sequence starts in the initial marking and ends in the final marking. AN_1 in Figure 5 has infinitely many complete firing sequences because of the loop involving $t5$ and $t4$. Examples of such complete firing sequences are $\langle t1, t2, t3, t4, t6\rangle$, $\langle t1, t4, t3, t2, t5, t4, t6\rangle$, and $\langle t1, t3, t2, t4, t5, t4, t5, t4, t6\rangle$. Note that when starting in the initial marking M_{init} it is possible to fire the transitions in the order indicated (i.e., the transition are enabled in the intermediate markings) ending in the final marking M_{final}.

Definition 13 (Accepting Traces of an Accepting Petri Net). *Let $AN = (N, M_{init}, M_{final})$ be an accepting Petri net with $N = (P, T, F, l)$ and $FS \subseteq T^*$ the set of all*

complete firing sequences. $lang(AN) = \{l(\sigma) \mid \sigma \in FS\}$ *is the set of all accepting traces.*[2]

For AN_1 in Figure 5 there are infinitely many accepting traces $lang(AN_1) = \{\langle a, b, c, d, f\rangle, \langle a, d, c, b, e, d, f\rangle, \langle a, c, b, d, e, d, e, d, f\rangle, \ldots\}$.

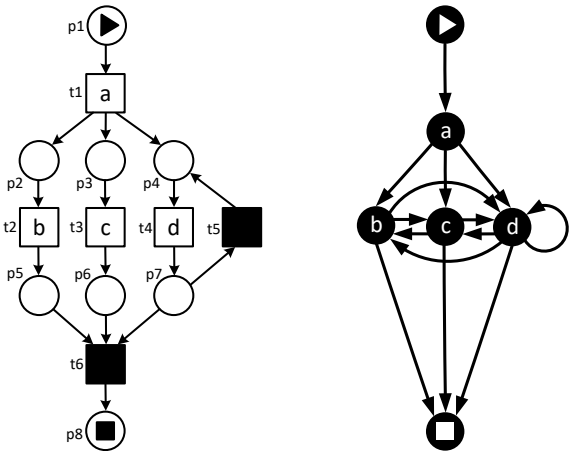

Fig. 6. Accepting Petri net AN_2 (left) and DFG G_2 (right). Note that the accepting Petri net has two silent transitions $t5$ and $t6$.

Accepting Petri net AN_2 in Figure 6 has the same network structure as AN_1 but a different labeling function l_2. $dom(l_2) = \{t1, t2, t3, t4\}$, i.e., $t5$ and $t6$ do not have a label and are called silent. This means that these transitions cannot be observed in the corresponding accepting traces. $lang(AN_2) = \{\langle a, b, c, d\rangle, \langle a, d, c, b, d\rangle, \langle a, c, b, d, d, d\rangle, \ldots\}$.

Figures 5 and 6 also show the corresponding DFGs: G_1 and G_2. Since DFGs cannot express concurrency, it is impossible to create DFGs that have the same set of accepting traces as the accepting Petri nets AN_1 and AN_2. Note that $lang(AN_1) \subset lang(G_1)$ and $lang(AN_2) \subset lang(G_2)$ and the differences are significant. In AN_1 and AN_2 there is always one b and one c in each accepting trace. However, in G_1 and G_2 there can be any number of b's and c's.

This explains why DFGs tend to be severely underfitting. Whenever activities do not happen in a fixed sequence, loops are introduced. However, DFGs, in combination with filtering, can be used to get a valuable first impression of the underlying process.

3.6 Conformance Checking

The goal of process discovery is to find a process model whose behavior is "close" to the behavior seen in the event log. Conformance-checking techniques aim to measure

[2] Note that labeling function l is applied to a sequence of transitions, i.e., $l(\langle\rangle) = \langle\rangle$, $l(\langle t\rangle) = \langle l(t)\rangle$ if $t \in dom(l)$, $l(\langle t\rangle) = \langle\rangle$ if $t \notin dom(l)$, and $l(\sigma \cdot \langle t\rangle) = l(\sigma) \cdot l(\langle t\rangle)$ for any $\sigma \in T^*$.

the distance between an event log and a process model. These techniques can be used to evaluate the quality of a discovered or hand-made process model. Here, we focus on the simplified setting where we have an event log $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ and a process model PM having $lang(PM) \in \mathcal{P}(\mathcal{U}_{act}{}^*)$ as accepting traces. We have seen DFGs and accepting Petri nets as examples. Despite the simplified setting, there are many challenges. Some examples:

- A process model may serve *different goals*. Should the model summarize past behavior, or is the model used for predictions and recommendations? Should the model show all behavior or just dominant behavior?
- Different notations provide different *representational biases* and may make it impossible to express certain behaviors. For example, Figures 5 and 6 illustrate that DFGs are unable to capture concurrent behavior.
- An event log contains just *example behavior*. Typically, a few trace variants are frequent and many trace variants are infrequent. Even when the process is stable, two event logs collected over different periods may have markedly different infrequent trace variants. The fact that something was not observed in a particular period does not mean it cannot happen or is not part of the process.
- Although event logs are multisets with *frequent* and *infrequent* traces, most process discovery techniques aim to discover process models that are "binary", i.e., a trace is possible or not.

These challenges show that it is impossible to restrict conformance to a single measure. In [1], the following four *quality dimensions* to evaluate a process model PM in the context of an event log L were identified.

- *Recall*, also called (replay) fitness, aims to quantify the fraction of observed behavior that is allowed by the model.
- *Precision* aims to quantify the fraction of behavior allowed by the model that was actually observed (i.e., avoid "underfitting" the event data).
- *Generalization* aims to quantify the probability that new unseen cases will fit the model (i.e., avoid "overfitting" the event data).
- *Simplicity* refers to Occam's Razor and can be made operational by quantifying the complexity of the model (number of nodes, number of arcs, understandability, etc.).

In this paper, we do not focus on a single measure and use the following generic definition.

Definition 14 (Conformance Measure). *A conformance measure is a function $conf \in \mathcal{B}(\mathcal{U}_{act}{}^*) \times \mathcal{P}(\mathcal{U}_{act}{}^*) \rightarrow [0,1]$. $conf(L, lang(PM))$ quantifies conformance for an event log $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ and a process model PM having $lang(PM) \in \mathcal{P}(\mathcal{U}_{act}{}^*)$ as accepting traces (higher is better).*

Let $L \in \mathcal{B}(\mathcal{U}_{act}{}^*)$ and $lang(PM) \in \mathcal{P}(\mathcal{U}_{act}{}^*)$. Some example measures are:

- Trace-based recall: $conf_1(L, lang(PM)) = \frac{|[\sigma \in L | \sigma \in lang(PM)]|}{|L|}$, i.e., the fraction of observed traces that fit into the model.

- Trace-based precision: $conf_2(L, lang(PM)) = \frac{|\{\sigma \in lang(PM) | \sigma \in L\}|}{|lang(PM)|}$, i.e., the fraction of modeled traces that were observed.
- DF-based recall: $conf_3(L, lang(PM)) = \frac{|[(a,b) \in DF_L | (a,b) \in DF_{PM}]|}{|DF_L|}$ with $DF_L = [(a_i, a_{i+1}) \in \mathcal{U}_{act} \times \mathcal{U}_{act} \mid \sigma = \langle a_1, a_2, \ldots, a_n \rangle \in L \wedge a_0 = \blacktriangleright \wedge a_{n+1} = \blacksquare \wedge 0 \leq i \leq n]$ and $DF_{PM} = \{(a_i, a_{i+1}) \in \mathcal{U}_{act} \times \mathcal{U}_{act} \mid \sigma = \langle a_1, a_2, \ldots, a_n \rangle \in lang(PM) \wedge a_0 = \blacktriangleright \wedge a_{n+1} = \blacksquare \wedge 0 \leq i \leq n\}$, i.e., the fraction of observed directly-follows relationships that also exist in the model.
- DF-based precision: $conf_4(L, lang(PM)) = \frac{|\{(a,b) \in DF_{PM} | (a,b) \in DF_L\}|}{|DF_{PM}|}$, i.e., the fraction of observed directly-follows relationships that also exist in the model.

The four conformance functions are just examples: $conf_1$ and $conf_3$ aim to measure recall (i.e., the fraction of observed behavior that is allowed by the model) and $conf_2$ and $conf_4$ aim to measure precision (i.e., the fraction of behavior allowed by the model that was actually observed). $conf_1$ and $conf_2$ consider complete traces and $conf_3$ and $conf_4$ compare direct successions in model and log. Note that $conf_2$ is problematic because $conf_2(L, lang(PM)) = 0$ if the model has a loop, no matter how large the event log is.

There are dozens (if not hundreds) of process-discovery and conformance-checking techniques, all assuming a simple event log based on a fixed case notion. A complete overview is out of scope. The goal of this tutorial is to show that these techniques can be lifted to object-centric process mining techniques. See [1] for a comprehensive introduction into process discovery and conformance checking.

3.7 Convergence and Divergence

In Section 3.2, we showed that any object-centric event log $L = (E, O, \#, R)$ can be converted to a simple event log L_{ot} by projecting events onto a single object type $ot \in types(L)$. We often call this the *flattening of an event log* into an event log where we again have *precisely one* case object per event. Figure 3 visualizes this process.

To better understand what happens with events during the flattening of an event log, consider an arbitrary event $e \in E$ with objects $O_e = \{o \in objects(e) \mid \#_{type}(o) = ot\}$ of type ot.

- If $|O_e| = 0$, then event e will not appear in L_{ot}. This is referred to as the *deficiency problem*. Potentially relevant events may disappear from the event log in this way.
- If $|O_e| = 1$, then event e appears precisely one in L_{ot}. This case is easy to interpret.
- If $|O_e| = k > 1$, then event e appears k times in L_{ot}, i.e., the same event is replicated. This may lead to confusing diagnostics and is known as the *convergence problem*. For example, based on Figure 4 one may think that a happened 6 times. However, in reality it happened only 3 times. This shows that many Key Performance Indicators (KPIs) computed based on the flatted event log will be misleading (e.g., costs, rework, mean duration, etc.)

The *divergence problem* is caused by the fact that after flattening the event log, we are unable to distinguish events referring to the same case and activity. Consider an object o of the selected object type ot and o_1 o_2, and o_3 three objects of another type

(not selected). Let $events(o) = \langle e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8 \rangle$, $events(o_1) = \langle e_2, e_3 \rangle$, $events(o_2) = \langle e_4, e_7 \rangle$, and $events(o_3) = \langle e_5, e_6 \rangle$, with $\#_{act}(e_1) = a$, $\#_{act}(e_2) = b$, $\#_{act}(e_3) = c$, $\#_{act}(e_4) = b$, $\#_{act}(e_5) = b$, $\#_{act}(e_6) = c$, $\#_{act}(e_7) = c$, and $\#_{act}(e_8) = d$. Hence, $trace(o) = \langle a, b, c, b, b, c, c, d \rangle$, $trace(o_1) = \langle b, c \rangle$, $trace(o_2) = \langle b, c \rangle$, and $trace(o_3) = \langle b, c \rangle$. When considering only o of the selected object type ot, it seems that activities b and c occur in random order (note that in $trace(o)$, b is followed by c and b and c is followed by b, c, and d). However, when considering o_1 o_2, and o_3 it becomes clear that b is always followed by c. This information gets lost on the flattened event log. The divergence problem leads to misleading process models that do *not* show the actual causalities. The resulting models often have loops because unrelated events are connected.

These problems are omnipresent when dealing with real-world processes and information systems. They illustrate the need for object-centric process mining.

4 Object-Centric Process Discovery

In Section 3, we showed that an object-centric event log $L = (E, O, \#, R)$ can be converted into a collection of simple event logs. For each object type $ot \in types(L)$, one can create a traditional simple event log $L_{ot} = [trace(o) \mid o \in O \wedge \#_{type}(o) = ot] \in \mathcal{B}({\mathcal{U}_{act}}^*)$. For each L_{ot} we can create a process model, e.g., a DFG $G = (A, F)$ or an accepting Petri net $AN = (N, M_{init}, M_{final})$. In this section, we show that these models can be folded into object-centric process models.

Note that in prior work, an abundance of event-logging formats and models have been proposed. Consider, for example, proclets [7], object-centric behavioral constraint models [11, 34], composite state machines [21], and various flavors of artifact-centric process models [23, 24, 35]. It is impossible to be complete here. Most of these models and approaches turned out to be too complex to be practically feasible. Therefore, we decided to resort to the simple representation proposed in [3] and standardized in the form of OCEL (`ocel-standard.org`). Unlike many of the artifact- and proclet-based approaches, we do not consider interacting entities (i.e., objects or artifact), but shared events only, i.e., one event may refer to multiple objects. It turns out that this approach helps to leverage existing techniques for process discovery and conformance checking. This is illustrated in the remainder.

Before introducing the folding operation, we first introduce some functions that help us to characterize the relationship between events, activities, objects, and object types. A central element is the "mix" of object types involved in the occurrence of an activity. Consider an activity $meeting$ and three related object types $person$, $room$ and $project$. If a $meeting$ activity *occurs* in *mode* $[person^5, room, project^2]$, then this refers to an event referring to activity $meeting$ and 8 objects: 5 objects of type $person$, 1 object of type $room$, and two objects of type $project$. We refer to such an event as *activity occurrence* $(meeting, [person^5, room, project^2])$. Other examples of activity occurrences are $(meeting, [person^{20}, room, project])$, $(clean\ room, [person^2, room])$, and $(approve\_project, [person^3, project])$. Note that there may be many events with the same activity occurrence. An event log can be transformed into a multiset of activ-

ity occurrences, e.g., $occ(L) = [(meeting, [person^5, room, project^2])^3, (clean\ room, [person^2, room])^2, \ldots]$.

Definition 15 (Activity Characteristics). *For an event log $L = (E, O, \#, R)$ with activities $A = act(L)$ and object types $OT = types(L)$ we define the following concepts and functions:*

- *An execution mode $m \in \mathcal{B}(OT)$ is a multiset of object types and characterizes the types of objects and their count involved in an event.*
- *An activity occurrence $(a, m) \in A \times \mathcal{B}(OT)$ refers to an activity a executed in mode m.*
- *$aot(L) = \{(\#_{act}(e), \#_{type}(o)) \mid e \in E \wedge o \in objects(e)\} \subseteq A \times OT$ is the set of of all activity-object-type combinations in event log L.*
- *$occ(L) = [(\#_{act}(e), [\#_{type}(o) \mid o \in objects(e)]) \mid e \in E] \in \mathcal{B}(A \times \mathcal{B}(OT))$ is the multiset of activity occurrences present in event log L.*
- *$mode_L(a) = [m \mid (a', m) \in occ(L) \wedge a' = a] \in \mathcal{B}(OT)$ is the multiset of execution modes of $a \in A$.*
- *$min_L(a, ot) = \min_{m \in mode_L(a)} m(ot) \in \mathbb{N}$ is the minimum number of $ot \in OT$ objects of involved when executing activity $a \in A$.*
- *$max_L(a, ot) = \max_{m \in mode_L(a)} m(ot) \in \mathbb{N}$ is the maximum number of $ot \in OT$ objects of involved when executing activity $a \in A$.*
- *$mean_L(a, ot) = \frac{\sum_{m \in mode_L(a)} m(ot)}{|mode_L(a)|}$ is the mean number of $ot \in OT$ objects involved when executing activity $a \in A$.*

To illustrate these measures, we use an example event log $L = (E, O, \#, R)$ with 22080 events and 11300 objects. A small fragment of the event log in tabular format is shown in Figure 7. $E = \{e_1, e_2, \ldots, e_{22080}\}$ are the 22080 events. Figure 7 shows the properties and related objects of events e_{13226} until e_{13244}, e.g., $\#_{act}(e_{13233}) = place\ order$ and $objects(e_{13233}) = \{991222, 884950, 884951, 884952\}$. There are three object types $order$, $item$, and $package$: order numbers start with "99", item numbers start with "88", and package numbers start with "66". Figure 7 also shows the timestamp of each event and two additional attributes: $price$ and $weight$.

Event e_{13233} represents an activity occurrence $(place\ order, [order, item^3])$, i.e., one order and three items are involved in the execution of activity $place\ order$. Some more examples: event e_{13226} represents an activity occurrence $(pick\ item, [item])$, event e_{13231} represents an activity occurrence $(package\ delivered, [item^6, package])$, and event e_{13239} represents an activity occurrence $(place\ order, [order, item^5])$. We can also compute $|mode_L(a)|$ (number of a events), $min_L(a, ot)$ (the minimum number of ot objects involved in a events), $mean_L(a, ot)$ (the average number of ot objects involved in a events), and $max_L(a, ot)$ (the maximum number of ot objects involved in a events). Table 1 shows the corresponding values for all activity/object-type combinations. For example, $min_L(place\ order, item) = 1$, $mean_L(place\ order, item) = 4.0$, $max_L(place\ order, item) = 13$, $min_L(create\ package, item) = 1$, $mean_L(create\ package, item) = 6.17$, and $max_L(create\ package, item) = 19$.

After showing some descriptive statistics, we create three events logs using Definition 5: L_{order}, L_{item}, and $L_{package}$. Recall that $L_{ot} = [trace(o) \mid o \in O \wedge \#_{type}(o) = ot] \in \mathcal{B}(\mathcal{U}_{act}{}^*)$.

e_i	activity	time	orders	items	packages	price	weight
13226	pick item	2022-09-27 17:08:09	{}	{884875}	{}	€ 129,99	0,98
13227	item out of stock	2022-09-27 17:18:12	{}	{884873}	{}	€ 2.200,00	1,25
13228	pick item	2022-09-27 17:26:02	{}	{884872}	{}	€ 129,99	0,98
13229	pick item	2022-09-27 17:29:36	{}	{884939}	{}	€ 495,00	0,48
13230	create package	2022-09-27 17:29:36	{}	{884826,884790,884827,884824,884914,884913,884915,884916,884822}	{660768}	€ 10.231,99	5,81
13231	package delivered	2022-09-27 17:34:30	{}	{884828,884834,884830,884832,884829,884833}	{660763}	€ 1.924,96	3,15
13232	pick item	2022-09-27 17:40:24	{}	{884858}	{}	€ 129,99	0,98
13233	place order	2022-09-27 17:53:09	{991222}	{884950,884951,884952}	{}	€ 5.534,00	2,95
13234	pay order	2022-09-27 17:55:19	{991094}	{}	{}	€ 3.463,98	2,90
13235	send package	2022-09-27 19:05:09	{}	{884788,884904,884889,884902,884903,884886,884887,884885,884787}	{660767}	€ 9.117,97	7,17
13236	place order	2022-09-27 20:15:42	{991223}	{884953,884954,884955}	{}	€ 2.674,98	2,55
13237	pick item	2022-09-27 20:24:50	{}	{884923}	{}	€ 99,99	0,78
13238	send package	2022-09-27 21:47:44	{}	{884826,884790,884827,884824,884914,884913,884915,884916,884822}	{660768}	€ 10.231,99	5,81
13239	place order	2022-09-27 23:07:38	{991224}	{884956}	{}	€ 134,00	0,50
13240	confirm order	2022-09-28 08:54:07	{991223}	{}	{}	€ 2.674,98	2,55
13241	place order	2022-09-28 08:57:24	{991225}	{884957,884958,884959,884960,884961}	{}	€ 2.431,98	2,37
13242	pick item	2022-09-28 09:13:17	{}	{884905}	{}	€ 29,99	0,38
13243	pick item	2022-09-28 09:17:41	{}	{884817}	{}	€ 699,00	0,17
13244	pick item	2022-09-28 09:25:34	{}	{884851}	{}	€ 495,00	0,48

Fig. 7. Small fragment of a larger object-centric event log with 22080 events and 11300 objects (200 orders, 8003 items, and 1297 packages).

$L_{order} = [\langle po, co, py\rangle^{1565}, \langle po, co, pr, py\rangle^{338}, \langle po, co, pr, pr, py\rangle^{80}, \langle po, co, pr, pr, pr, py\rangle^{15}, \langle po, co, pr, pr, pr, pr, py\rangle^{2}]$ when we use the short names introduced in Table 1, e.g., pr refers to $payment\ reminder$ and py refers to $pay\ order$. Figure 8 (left) shows the DFG created for L_{order} by applying Definition 10.

$L_{item} = [\langle po, pi, cp, sp, pd\rangle^{5231}, \langle po, is, ri, pi, cp, sp, pd\rangle^{1330}, \langle po, pi, cp, sp, fd, pd\rangle^{766}, \langle po, pi, cp, sp, fd, fd, pd\rangle^{271}, \langle po, is, ri, pi, cp, sp, fd, pd\rangle^{218}, \langle po, pi, cp, sp, fd, fd, fd, pd\rangle^{68}, \langle po, is, ri, pi, cp, sp, fd, fd, pd\rangle^{65}, \langle po, is, ri, pi, cp, sp, fd, fd, fd, pd\rangle^{24}, \langle po, pi, cp, sp, fd, fd, fd, fd, pd\rangle^{12}, \langle po, pi, cp, sp, fd, fd, fd, fd, pd\rangle^{10}, \langle po, is, ri, pi, cp, sp, fd, fd, fd, fd, pd\rangle^{7}, \langle po, is, ri, pi, cp, sp, fd, fd, fd, fd, fd, pd\rangle^{1}]$. The 8003 items are distributed over 12 variants. Figure 8 (middle) shows the DFG created for L_{item} by applying Definition 10.

$L_{package} = [\langle cp, sp, pd\rangle^{1060}, \langle cp, sp, fd, pd\rangle^{161}, \langle cp, sp, fd, fd, pd\rangle^{53}, \langle cp, sp, fd, fd, fd, pd\rangle^{16}, \langle cp, sp, fd, fd, fd, fd, pd\rangle^{5}, \langle cp, sp, fd, fd, fd, fd, fd, pd\rangle^{2}]$. The 1279 packages are distributed over 12 variants. Figure 8 (left) shows the DFG created for $L_{package}$.

Next, we combine the information in Table 1 and the three DFGs in Figure 8 into the *Object-Centric Directly Follows Graph* (OC-DFG) shown in Figure 9. The basic idea is as follows: An OC-DFG can be seen as a set of "stacked" or "folded" DFGs. The arcs and start and end activities always refer to a specific object type. The connections are realized through shared activities. The activity frequencies correspond to the *actual* frequencies. This helps to avoid the convergence and divergence problems mentioned before. Before we describe how to create the OC-DFG in Figure 9, we first formalize the concept of a OC-DFG.

Definition 16 (Object-Centric Directly Follows Graph). *An Object-Centric Directly Follows Graph (OC-DFG)* $OG = (A, OT, F, min, max)$ *is composed of the following elements:*

- $A \in \mathcal{B}(\mathcal{U}_{act})$ *is a multiset of activities,*
- $OT \subseteq \mathcal{U}_{type}$ *is a set of object types,*
- $A_{\blacktriangleright} = \{\blacktriangleright_{ot} \mid ot \in OT\}$ *and* $A_{\blacksquare} = \{\blacksquare_{ot} \mid ot \in OT\}$ *are artificial start and end activities (one per object type) such that* $(A_{\blacktriangleright} \cup A_{\blacksquare}) \cap A = \emptyset$*,*

Table 1. The characteristics of activities for the event log partially shown in Figure 7.

activity	frequency	object type								
		$ot = order$			$ot = item$			$ot = package$		
a	$\lvert mode_L(a)\rvert$	$min_L(a, ot)$	$mean_L(a, ot)$	$max_L(a, ot)$	$min_L(a, ot)$	$mean_L(a, ot)$	$max_L(a, ot)$	$min_L(a, ot)$	$mean_L(a, ot)$	$max_L(a, ot)$
confirm order (**co**)	2000	1	1	1	0	0	0	0	0	0
create package (**cp**)	1297	0	0	0	1	6.17	19	1	1	1
failed delivery (**fd**)	345	0	0	0	1	5.98	19	1	1	1
item out of stock (**is**)	1645	0	0	0	1	1	1	0	0	0
package delivered (**pd**)	1297	0	0	0	1	6.17	19	1	1	1
pay order (**py**)	2000	1	1	1	0	0	0	0	0	0
payment reminder (**pr**)	551	1	1	1	0	0	0	0	0	0
pick item (**pi**)	8003	0	0	0	1	1	1	0	0	0
place order (**po**)	2000	1	1	1	1	4.00	13	0	0	0
reorder item (**ri**)	1645	0	0	0	1	1	1	0	0	0
send package (**sp**)	1297	0	0	0	1	6.17	19	1	1	1

- $F \in \mathcal{B}(OT \times (A \cup A_{\blacktriangleright}) \times (A \cup A_{\blacksquare}))$ *is a weighted set of edges labeled with the corresponding object type,*
- *for any* $a \in A$*: there exists an object type* $ot \in OT$ *and path* $\sigma_{ot} = \langle a_1, \ldots, a_n \rangle$ *such that* $(ot, a_i, a_{i+1}) \in F$ *for* $1 \leq i < n$*,* $a_1 = \blacktriangleright_{ot}$*,* $a_n = \blacksquare_{ot}$*, and* $a \in \sigma_{ot}$*.*
- $|[b \mid (ot, b, a) \in F]| = |[b \mid (ot, a, b) \in F]|$ *for any* $a \in A$ *and* $ot \in OT$*,*
- $aot(OG) = \{(a, ot) \in A \times OT \mid \exists_{b \in A}\ (ot, b, a) \in F\}$ *is the set of activity-object-type combinations,*
- $min \in aot(OG) \rightarrow \mathbb{N}$ *and* $max \in aot(OG) \rightarrow \mathbb{N} \cup \{\infty\}$ *define cardinality constraints, i.e., for* $(a, ot) \in aot(OG)$*:* $min(a, ot)$ *is a lower bound for the number of* ot *objects involved in events corresponding to activity* a*,* $max(a, ot)$ *is an upper bound (of course* $min(a, ot) \leq max(a, ot)$*).*

Comparing Definition 16 (OC-DFG) with Definition 9 (DFG), shows the following. Both are composed of nodes corresponding to activities and weighted edges connecting these activities. However, in an OC-DFG there are separate edges and start and end activities *per object type*. For each object type ot, $\blacktriangleright_{ot}$ denotes the start of ot objects and $\blacksquare_{ot}$ the end. $A(a)$ denote the real frequency of activity a. $F(ot, a, b)$ denotes how often an object of type ot "moved" from a to b. Since each edge refers to an object type, we can talk about ot-paths $\sigma_{ot} = \langle a_1, \ldots, a_n \rangle$ with $(ot, a_i, a_{i+1}) \in F$ for $1 \leq i < n$ connecting $\blacktriangleright_{ot} = a_1$ to $\blacksquare_{ot} = a_n$ using only ot-edges. Each activity should be on one of such paths. The requirement $|[b \mid (ot, b, a) \in F]| = |[b \mid (ot, a, b) \in F]|$ for any $a \in A$ and $ot \in OT$ states that the sum of the frequencies of the input arcs of an activity a matches the sum of the frequencies of the output arcs of a for each individual object type ot. Note that these numbers may be different from $A(a)$ and that $|[b \mid (ot, b, a) \in F]| = |[b \mid (ot, a, b) \in F]| = 0$ if $(a, ot) \notin aot(OG)$ and $|[b \mid (ot, b, a) \in F]| = |[b \mid (ot, a, b) \in F]| > 0$ if $(a, ot) \in aot(OG)$.

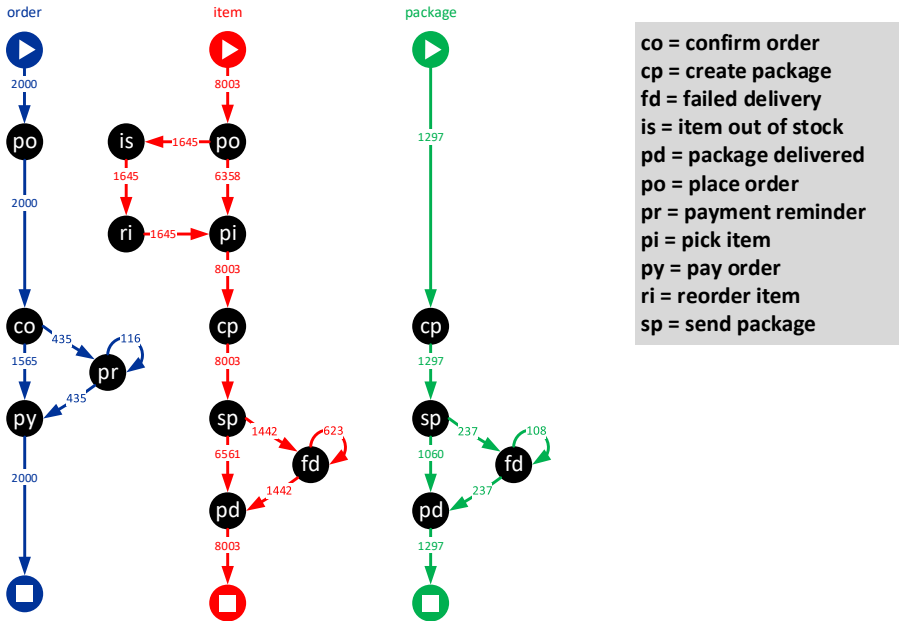

Fig. 8. Three DFGs created for the flattened event logs L_{order} (left), L_{item} (middle), and $L_{package}$ (right).

There is the special case that $min(a, ot) = max(a, ot) = 1$, i.e., the number of ot objects involved in events corresponding to activity a is precisely one. We call these *simple activity-object-type combinations*, denoted by $saot(OG) = \{(a, ot) \in aot(OG) \mid min((a, ot) = max((a, ot) = 1\}$. Note that for $(a, ot) \in saot(OG)$: $|[b \mid (ot, b, a) \in F]| = |[b \mid (ot, a, b) \in F]| = A(a)$. This is similar to the requirement in Definition 9 (DFG). In a normal DFG, the number of ot objects involved in events corresponding to activity a is precisely one. Figure 9 uses *single-headed* arcs for simple activity-object-type combinations. For other activity-object-type combinations, *double-headed* arcs are used. The property is always attached to the incoming arc. To simplify language, we refer to these as *normal arcs* (single-headed) and *variable arcs* (double-headed). However, it is a property of the activity-object-type combination.

Definition 17 (Object-Centric Directly Follows Graph Discovery). *Let $L = (E, O, \#, R)$ be an object-centric event log. $OG(L) = (A, OT, F, min, max)$ is the corresponding Object-Centric Directly Follows Graph (OC-DFG) and is defined as follows:*

- *$A = [\#_{act}(e) \mid e \in E]$ is the multiset of activities,*
- *$OT = types(L)$ is the set of object types,*
- *$F = [(ot, a_i, a_{i+1}) \mid o \in O \wedge ot = \#_{type}(o) \wedge trace(o) = \langle a_1, a_2, \ldots, a_n \rangle \wedge a_0 = \blacktriangleright_{ot} \wedge a_{n+1} = \blacksquare_{ot} \wedge 0 \leq i \leq n]$ is the weighted set of edges labeled with the corresponding object type,*
- *$min \in aot(L) \rightarrow \mathbb{N}$ and $max \in aot(L) \rightarrow \mathbb{N}$ such that for $(a, ot) \in aot(L)$: $min(a, ot) = min_L(a, ot)$ and $max(a, ot) = max_L(a, ot)$ (see Definition 15).*

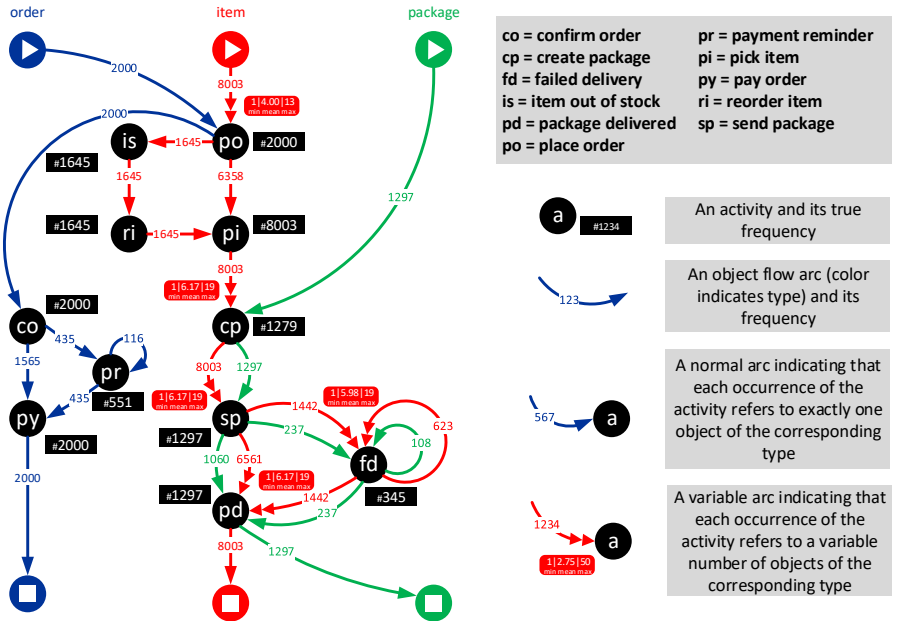

Fig. 9. An object-centric DFG based on the information in Table 1 and three DFGs in Figure 8. The three colors refer to the three object types.

Applying Definition 17 to the event log L partially shown in Figure 7 (i.e., the event log with 22080 events and 11300 objects) results in the OC-DFG shown in Figure 9. Using the short names, we obtain $OG(L) = (A, OT, F, min, max)$ with

- $A = \{(co, 2000), (cp, 1297), (fd, 345), (is, 1645), (pd, 1297), (po, 2000), (pr, 551), (pi, 8003), (py, 2000), (ri, 1645), (sp, 1297)\}$,
- $OT = \{order, item, package\}$,
- $F = [(order, \blacktriangleright_{order}, po)^{2000}, (item, \blacktriangleright_{item}, po)^{8003}, (package, \blacktriangleright_{package}, cp)^{1297}, (order, po, co)^{2000}, (item, po, is)^{1645}, (item, po, pi)^{6358}, (package, cp, sp)^{1297}, (item, cp, sp)^{8003}, \ldots, (item, pd, \blacksquare_{item})^{8003}, (package, pd, \blacksquare_{package})^{1297}]$
- $min = \{((po, order), 1), ((co, order), 1), ((pr, order), 1), ((py, order), 1), ((po, item), 1), ((is, item), 1), ((ri, item), 1), ((pi, item), 1), ((cp, item), 1), ((sp, item), 1), ((fd, item), 1), ((pd, item), 1), ((cp, package), 1), ((sp, package), 1), ((fd, package), 1), ((pd, package), 1)\}$
- $max = \{((po, order), 1), ((co, order), 1), ((pr, order), 1), ((py, order), 1), ((po, item), 16), ((is, item), 1), ((ri, item), 1), ((pi, item), 1), ((cp, item), 19), ((sp, item), 19), ((fd, item), 19), ((pd, item), 19), ((cp, package), 1), ((sp, package), 1), ((fd, package), 1), ((pd, package), 1)\}$

Note that Definition 17 only computes cardinality constraints for activity-object-type combinations. There are many ways to extend the model with more fine-grained information. For example, Figure 9 also shows the mean number of $ot \in OT$ objects of

involved when executing activity $a \in A$, i.e., the $mean_L(a, ot) = \frac{\sum_{m \in mode_L(a)} m(ot)}{|mode_L(a)|}$ value already introduced in Definition 15.

Instead of cardinality constraints, it is also possible to define an *allowed set of execution modes*. This allows us to relate the frequencies of different object types. For example, the number of objects of one type matches the number of objects of another type. In Definition 15, we already showed how such information can be extracted from event logs.

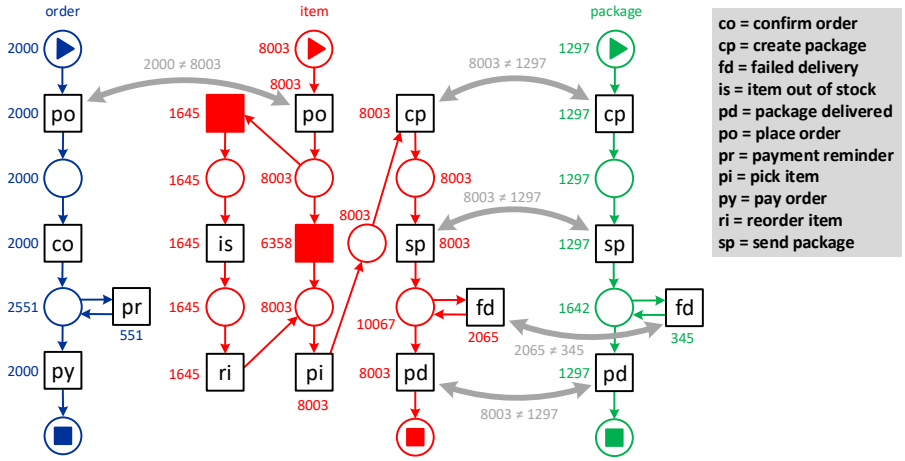

Fig. 10. Three accepting Petri nets created for the flattened event logs L_{order} (left), L_{item} (middle), and $L_{package}$ (right). Note that the accepting Petri net in the middle has two silent activities. The gray bidirectional arcs show disagreements between the different models and logs. For example, activity *po* (place order) occurs 2000 times in L_{order} (left) and 8003 times in L_{item} (middle), and activity *cp* (create package) occurs 8003 times in L_{item} (middle) and 1287 times in $L_{package}$ (right). When folding the accepting Petri nets, these differences need to be addressed.

The same principles can be applied to other representations and discovery algorithms. The only limitation is that activities have to be unique, i.e., it is not possible to have models where two nodes (e.g., transitions in a Petri net) refer to the same activity.

We use the same event log to illustrate the folding of accepting Petri nets. Figure 10 shows three accepting Petri nets discovered for the three flattened event logs created before: L_{order} (left), L_{item} (middle), and $L_{package}$ (right). Each transition is annotated with the frequency in the corresponding flattened event log. The places indicate how often a token was produced and later consumed in that place (i.e., the token-flow frequency). Using exactly the same principles as before, we can fold the three accepting Petri nets resulting in the *Object-Centric Accepting Petri Net* (OC-APN) shown in Figure 11.

The object-centric accepting Petri net is obtained as follows. The three accepting Petri nets are joined, assuming that all places and arcs are disjoint. The only thing shared are visible transitions with the same label. Silent transitions are never merged. Then the

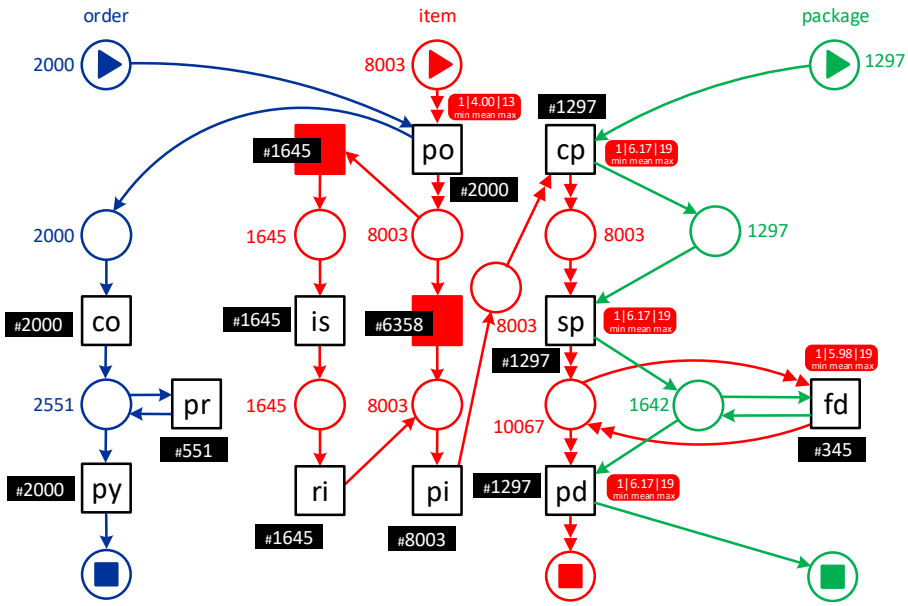

Fig. 11. An object-centric accepting Petri net based on the three accepting Petri nets in Figure 10. The transitions are labeled with the correct frequencies, e.g., activity *po* (place order) occurs 2000 times and activity *cp* (create package) occurs 1297 times. The silent transitions do not need to be merged and can therefore reuse the frequencies shown in Figure 10. All other transitions have the frequency found in the original object-centric event log. The double-headed arcs indicate that, when firing the transition, possibly a variable number of objects is consumed and produced (i.e., not always precisely one). For transitions having double-headed arcs, additional statistics are added, e.g., occurrences of *po* (place order) involve between 1 and 13 items with a mean of 4.00 and occurrences of *cp* (create package) involve between 1 and 19 items with a mean of 6.17.

actual frequencies are added. These are obtained from the original event log. Next all activity-object-type combinations (a, ot) are inspected (as before). If the number of ot objects is always precisely one for activity a, then we use normal arcs (single-headed). If the number of ot objects is not always precisely one for activity a, then we use variable arcs (double-headed) and also add additional information. Figure 11 shows the minimum, mean, and maximum number of objects involved.

In Section 3.3, we introduced *activity* and *variant-based filtering*. These can be easily combined with the approach in this section. Moreover, it is also possible to *filter for particular activity-object-type combinations*. For example, one could selectively remove $item$ objects from the activities sp (send package) and fd (failed delivery), but keep the $item$ objects for activity pd (package delivered) to measure flow times.

Note that object-centric DFGs and object-centric accepting Petri net are just examples illustrating the basic principles of object-centric process discovery. This shows that techniques for classical process discovery (using a single case notion) can be lifted to multiple object types and a variable number of objects involved in the execution

of activities. This leads to a more holistic view avoiding convergence and divergence problems discussed before.

5 Object-Centric Conformance Checking

Now we consider the situation where we have both an object-centric event log and an object-centric process model. Without going onto details, we argue that we can use a similar approach as for process discovery. We can project both the event log and the process model on each of the object types and apply classical conformance-checking techniques. On top of that, we need to check the cardinality constraints.

In Definition 14, we defined a conformance measure to be a function $conf \in \mathcal{B}({\mathcal{U}_{act}}^*) \times \mathcal{P}({\mathcal{U}_{act}}^*) \rightarrow [0,1]$. $conf(L, lang(PM))$ quantifies conformance for an event log $L \in \mathcal{B}({\mathcal{U}_{act}}^*)$ and a process model PM having $lang(PM) \in \mathcal{P}({\mathcal{U}_{act}}^*)$ as accepting traces (higher is better). We provided examples such as trace-based recall, trace-based precision, DF-based recall, and DF-based precision.

Given an object-centric process model PM, we assume it is possible to extract a process model PM_{ot} for each object type. This requires an unfolding of the object-centric process model. For example, the object-centric accepting Petri net shown in Figure 11 is unfolded into the three accepting Petri nets in Figure 10. The object-centric DFG shown in Figure 9 is unfolded into the three DFGs in Figure 8. Of course, we should ignore the frequencies added to the process models during discovery. Frequencies only come into play when a concrete event log is considered and are not part of the normative process model. However, cardinality constraints are part of the model. Figure 12 illustrates this. Note that the object-centric process model does not show frequencies. The only annotations that are kept are the upper and lower bounds.

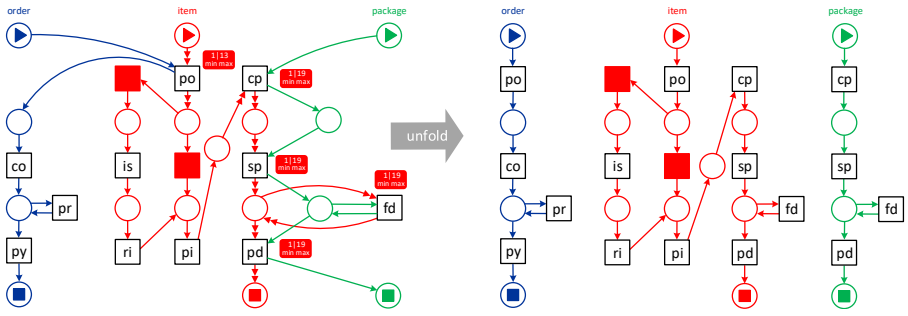

Fig. 12. Unfolding an object-centric accepting Petri net into one accepting Petri net per object type. Note the cardinality constraints, e.g., activity *po* always involves precisely one *order* object and between 1 and 13 *item* objects, activity *cp* always involves precisely one *package* object and between 1 and 19 *item* objects. These cannot be checked in the unfolded models, but can be checked on the original event log.

Using the flattened event logs (L_{ot}) and unfolded models (PM_{ot}), we can apply different conformance measures of the form $conf(L_{ot}, lang(PM_{ot}))$. These can be considered separately or aggregated in an overall measure.

By checking conformance using a collection of flattened event logs and unfolded models, we do not check the interactions between object types and also do not check cardinalities. However, these can be checked directly on the object-centric event log. Consider an event log $L = (E, O, \#, R)$ with activities $A = act(L)$ and object types $OT = types(L)$. In Definition 15, we introduced $occ(L) = [(\#_{act}(e), [\#_{type}(o) \mid o \in objects(e)]) \mid e \in E]$ as the multiset of activity occurrences present in event log L and $mode_L(a) = [m \mid (a', m) \in occ(L) \wedge a' = a]$ as the multiset of execution modes of $a \in A$.

Now assume that the process model PM defines a *set of allowed activity occurrences* $aocc(PM) \subseteq A \times \mathcal{B}(OT)$. If $(a, m) \in aocc(PM)$, then activity a can be executed in mode m. To illustrate this consider the cardinality constraints in Figure 12 (left). In this example: $aocc(PM) = \{(po, [order, item^k]) \mid 1 \leq k \leq 13\} \cup \{(cp, [item^k, package]) \mid 1 \leq k \leq 19\} \cup \{(sp, [item^k, package]) \mid 1 \leq k \leq 19\} \cup \{(fd, [item^k, package]) \mid 1 \leq k \leq 19\} \cup \{(pd, [item^k, package]) \mid 1 \leq k \leq 19\} \cup \{(co, [order]), (pr, [order]), (py, [order]), (is, [item]), (ri, [item]), (pi, [item])\}$. This shows that it is quite easy to add graphical annotations to object-centric process models describing the set of allowed activity occurrences. Activity-occurrence-based recall computes the fraction of observed activity occurrences allowed according to the process model.

Definition 18 (Checking Allowed Activity Occurrences). *Let $L = (E, O, \#, R)$ be an object-centric event log with an observed multiset of activity occurrences $occ(L)$ and PM an object-centric process model with $aocc(PM) \subseteq A \times \mathcal{B}(OT)$ as the set of allowed activity occurrences. $conf_{occ}(L, PM) = \frac{|[(a,m) \in occ(L) \mid (a,m) \in aocc(PM)]|}{|occ(L)|}$ is activity-occurrence-based recall.*

Note that $conf_{occ}(L, PM) \in [0, 1]$. For the running example, $conf_{occ}(L, PM) = 1$. However, if we make the cardinality constraints more strict (e.g., an order or package contains at most 5 items), then $conf_{occ}(L, PM) < 1$.

To summarize the above. Object-centric conformance checking takes as input an object-centric event log and an object-centric process model. Then two complementary checks are conducted: (1) for each object type, a flattened event log and unfolded model are created that are checked using traditional conformance-checking techniques (see Section 3.6) and (2) the observed multiset of activity occurrences $occ(L)$ is compared with the allowed set of activity occurrences $aocc(PM)$. These checks can be used to provide detailed diagnostics and can also be combined in an overall measure (e.g., a weighted average).

6 Other Forms of Object-Centric Process Mining

Process mining is not limited to process discovery and conformance checking and includes a wide variety of topics ranging from data quality problems specific for event data to action-oriented process mining where machine learning techniques are used to

predict problems and automatically take action. The broadness of the spectrum (including connections to neighboring fields like simulation, data management, artificial intelligence, and machine learning) also applies to object-centric process mining, as is shown in Figure 13.

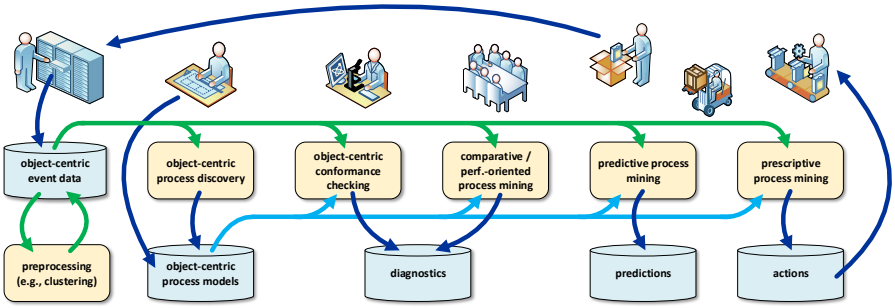

Fig. 13. Object-centric process mining is not limited to process discovery and conformance checking.

The process mining tasks described in Figure 13 have been researched during the past two decades. However, the focus was always on simple event logs, i.e., event logs assuming a single case notion. Like for process discovery and conformance checking, these tasks may become very challenging when dealing with multiple object types and a variable number of objects involved in events. Although it is often possible to lift existing approaches to the multi-object level (as was illustrated in the previous sections), many new challenges emerge when we drop the assumption of having a single case notion.

Consider, for example, the topic of *trace clustering* [27, 39, 40], which has been investigated in-depth for simple event logs. The goal of trace clustering is to find groups of cases that have similar characteristics. This can be used to partition the event log into multiple smaller, more homogeneous, event logs. These can be used to discover a collection of process models that are easier to analyze and interpret. These ideas cannot be easily transferred to object-centric event logs. Of course, one can try to cluster objects. However, objects share events and the log cannot be nicely partitioned based on clustered objects.

The same challenges can be seen when considering predictive process mining and prescriptive process mining. Often cases are assumed to be independent and this is no longer possible. This also shows that traditional process mining approaches fail to capture the interactions between different objects.

7 Handling Complexity

As mentioned in the introduction of this paper, the advantages of using object-centric process mining are threefold: (1) data extraction is only done once, (2) relations be-

tween objects of possibly multiple types are captured and analyzed, and (3) a three-dimensional view of processes and organizations comes into reach. However, because data exaction is only done once, the complexity of input data is typically far more complex than in traditional process mining. Traditionally, scoping is done before turning the data from source systems (e.g., SAP's 800.000 database tables) into an event log with a single case notion. Moreover, sliders and filters need to be reinvented for the new setting with multiple object types and events that refer to multiple objects.

An important tool to tackle complexity is the so-called *Event-Type-Object-Type* (ETOT) matrix. Recall that $act(L) = \{\#_{act}(e) \mid e \in E\}$ is the set of event types (i.e., activities) and $types(L) = \{\#_{type}(o) \mid o \in O\}$ is the set of object types. An ETOT matrix determines which *event-type* and *object-type combinations* are in focus. Given an event log and an ETOT matrix, we create an event log that retains only the event types, object types, and relations selected.

Definition 19 (Event-Type-Object-Type Matrix). *Let $L = (E, O, \#, R)$ be an event log. An Event-Type-Object-Type (ETOT) matrix is a relation $M \subseteq \{(act(e), \#_{type}(o)) \mid (e, o) \in R\}$. $L{\uparrow}M = (E', O', \#', R')$ is the projected event log based on M with $E' = \{e \in E \mid act(e) \in \{et \mid (et, ot) \in M\}\}$, $O' = \{o \in O \mid \#_{type}(o) \in \{ot \mid (et, ot) \in M\}\}$, $dom(\#') = E' \cup O'$, $\#'(e) = \#(e)$ for $e \in E'$, $\#'(o) = \#(o)$ for $o \in O'$, and $R' = \{(e, o) \in R \mid (act(e), \#_{type}(o)) \in M\}$.*

Table 2. The maximal Event-Type-Object-Type (ETOT) matrix based on the event log partially shown in Figure 7.

event type (activity)	object type		
	$ot = order$	$ot = item$	$ot = package$
confirm order (**co**)	✓		
create package (**cp**)		✓	✓
failed delivery (**fd**)		✓	✓
item out of stock (**is**)		✓	
package delivered (**pd**)		✓	✓
pay order (**py**)	✓		
payment reminder (**pr**)	✓		
pick item (**pi**)		✓	
place order (**po**)	✓	✓	
reorder item (**ri**)		✓	
send package (**sp**)		✓	✓

Note that an Event-Type-Object-Type (ETOT) matrix M can be visualized as a matrix. This is illustrated in Table 2. The example ETOT matrix is maximal, i.e., $M = \{(act(e), \#_{type}(o)) \mid (e, o) \in R\}$. Table 3 shows another ETOT matrix where many event-type and object-type combinations have been removed. The corresponding discovered object-centric accepting Petri net is shown in Figure 14. Note that the object type *package* was completely removed. Also, activities such as *payment reminder* were removed. The resulting process model is, therefore, much simpler.

Table 3. An Event-Type-Object-Type (ETOT) matrix M used to create an event log focusing on a subset of event types and object types.

event type (activity)	object type		
	$ot = order$	$ot = item$	$ot = package$
confirm order (**co**)	✓		
create package (**cp**)			
failed delivery (**fd**)			
item out of stock (**is**)		✓	
package delivered (**pd**)		✓	
pay order (**py**)	✓		
payment reminder (**pr**)			
pick item (**pi**)		✓	
place order (**po**)	✓	✓	
reorder item (**ri**)		✓	
send package (**sp**)			

Table 3 and Figure 14 show that it is easy to scope analysis; just select the event-type and object-type combinations that need to be included. An ETOT matrix M can be considered as an "analysis profile" or "view" on the processes and organization. These may be predefined and reusable. Depending on the questions one has, one pick such a predefined profile/view.

Selecting the ETOT matrix can be seen as a first step in the analysis process. However, one can consider further filter or projection steps based on picking subsets of objects. It is recommended to include an object completely or not at all, otherwise, results are difficult to interpret. This leads to another subsequent log projection based on a set of selected objects O_{sel}.

Definition 20 (Object Selection). *Let $L = L'{\uparrow}M = (E, O, \#, R)$ be an event log obtained by selecting event-type and object-type combinations as defined by ETOT matrix M starting from event log L'. Let $O_{sel} \subseteq O$ be a set of selected objects. Projecting L on these selected objects O_{sel} yields $L{\uparrow}O_{sel} = (E', O', \#', R')$ with $O' = O_{sel}$, $R' = \{(e, o) \in R \mid o \in O'\}$, $E' = \{e \mid (e, o) \in R'\}$, $dom(\#') = E' \cup O'$, $\#'(e) = \#(e)$ for $e \in E'$, and $\#'(o) = \#(o)$ for $o \in O'$.*

Note that events without selected objects are removed in Definition 20. Object selection can be used to implement *sliders that seamlessly simplify process models*. For example, objects are sorted per object type based on the frequency of the corresponding trace variant (note that each object defines a sequence of activities). If the slider is set to, for example, 80%, then one can pick 80% of the objects per object type (starting with the objects following the most frequent variant). Using $L{\uparrow}O_{sel}$ as defined in Definition 20, one obtains a new event log covering, by definition, 80% of all objects. Such a slider has a clear interpretation and predictable effects. However, relationships between objects are ignored. This can be problematic, as can be explained using our running example.

Assume we have orders, items, and packages as objects. We select orders, items, and packages based on the frequencies of the corresponding variants. This may result

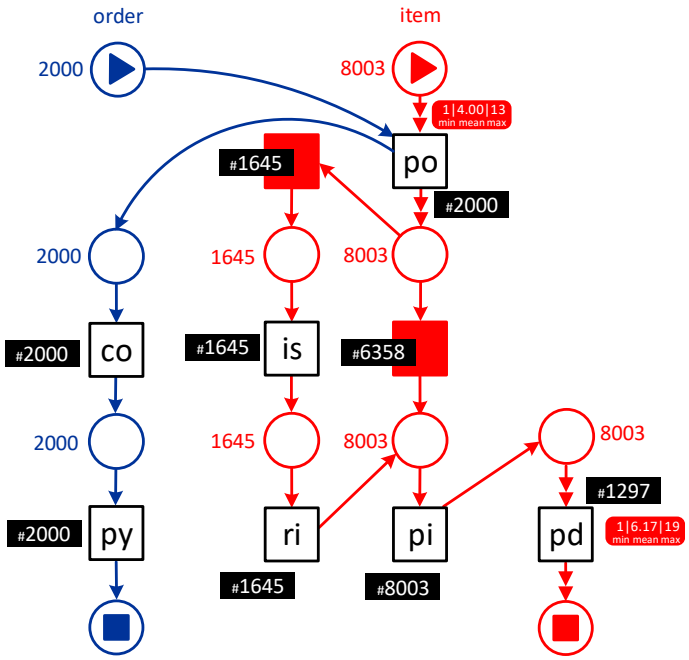

Fig. 14. An object-centric accepting Petri net based on event log $L{\uparrow}M$ created using the ETOT matrix M in Table 3.

in "orphan objects", e.g., orders without items, items without a corresponding order, packages without items, or items without a corresponding package. This is caused by the fact that dependencies between objects are ignored. One way to solve this is to use the notion of *leading objects*. One starts with a set of leading objects, e.g., all orders following the five most frequent variants, and all transitively related objects are included.

Definition 21 (Object Selection Based on Leading Objects). *Let* $L = L'{\uparrow}M = (E, O, \#, R)$ *be an event log obtained by selecting event-type and object-type combinations as defined by ETOT matrix* M *starting from event log* L'. $O2O = \{(o_1, o_2) \in O \times O \mid \exists_{e \in E} \{(e, o_1), (e, o_2)\} \subseteq R\}$. $O2O^*$ *is the transitive closure of relation* $O2O$. *Let* $O_{lead} \subseteq O$ *be the selected set of leading objects.* $O_{sel} = \{o \in O \mid \exists_{o' \in O_{lead}} (o', o) \in O2O^*\}$ *is the set of objects transitively connected to at least one leading object. Projecting* L *based on these objects connected to leading objects yields again* $L{\uparrow}O_{sel} = (E', O', \#', R')$ *with* $O' = O_{sel}$, $R' = \{(e, o) \in R \mid o \in O'\}$, $E' = \{e \mid (e, o) \in R'\}$, $dom(\#') = E' \cup O'$, $\#'(e) = \#(e)$ *for* $e \in E'$, *and* $\#'(o) = \#(o)$ *for* $o \in O'$.

Definition 21 allows us to pick a set of orders as leading objects and include all items included in these orders. It is also possible to select a set of packages and include all corresponding items and orders. Note that due to the transitive closure, too many objects may be included. Hence, further refinements or a well-chosen ETOT matrix

are needed to obtain the desired projected event log. However, Definition 20 and Definition 21, in combination with the ETOT matrix, provide the basic tools that can be used to implement sliders and filters. For example, we may want to create the process model based on all orders with a value of more than €8000, having items that are hazardous, shipped in packages with more than five items. Such questions require careful definitions to avoid misleading interpretations. However, the data set used as input is always the same. In traditional non-object-centric approaches, such selections are often done before loading the event data in a process mining tool. As a result, transparency is missing and analysts need to guess how the data was extracted. *Therefore, using object-centric process mining, we can handle complexity effectively. Moreover, we can also create transparency based on a single source of truth.*

8 Conclusion

Most process mining techniques and tools make the simplifying assumption that each event refers to precisely one case. This allows for the partitioning of events over different cases. Each case can be seen as a separate instance composed of a timed sequence of activities. It is remarkable that this simple view on operational processes has enabled so many organizations to improve their processes [25]. However, as the field of process mining is maturing and applications become more challenging, this simplifying assumption is no longer reasonable.

Object-centric process mining drops the "single case notion" assumption. An event may refer to any number of objects and different types of objects are considered in an integral manner. In this tutorial, we introduced object-centric event logs and two types of object-centric process models: Object-Centric Directly-Follows Graphs (OC-DFGs) and Object-Centric Accepting Petri Nets (OC-APNs). We presented a few baseline process-discovery and conformance-checking techniques. The goal was not to present specific techniques, but to introduce the challenges and show how existing techniques can be leveraged. For example, techniques and implementations, see [8, 15], the OCEL standard (`ocel-standard.org`), and the OCPM toolset. The insights provided are also valuable when using existing process mining software not supporting object-centric event logs. Also, practitioners should know about convergence and divergence problems, because these phenomena occur in almost any real-world process mining project.

Object-centric process mining is a new rapidly-growing subdiscipline with many exciting research challenges. Things like filtering, clustering, prediction, etc. become more challenging when considering object-centric event logs. However, the general principles used for object-centric process discovery and conformance checking presented in this tutorial can also be applied to most other process mining tasks.

Acknowledgments
The author thanks the Alexander von Humboldt (AvH) Stiftung for supporting his research. Funded by the Deutsche Forschungsgemeinschaft (DFG) under Germany's Excellence Strategy, Internet of Production (390621612).

References

1. W.M.P. van der Aalst. *Process Mining: Data Science in Action*. Springer-Verlag, Berlin, 2016.
2. W.M.P. van der Aalst. A Practitioner's Guide to Process Mining: Limitations of the Directly-Follows Graph. In *International Conference on Enterprise Information Systems (Centeris 2019)*, volume 164 of *Procedia Computer Science*, pages 321–328. Elsevier, 2019.
3. W.M.P. van der Aalst. Object-Centric Process Mining: Dealing With Divergence and Convergence in Event Data. In P.C. Ölveczky and G. Salaün, editors, *Software Engineering and Formal Methods (SEFM 2019)*, volume 11724 of *Lecture Notes in Computer Science*, pages 3–25. Springer-Verlag, Berlin, 2019.
4. W.M.P. van der Aalst. Foundations of Process Discovery. In W.M.P. van der Aalst and J. Carmona, editors, *Process Mining Handbook*, volume 448 of *Lecture Notes in Business Information Processing*, pages 37–75. Springer-Verlag, Berlin, 2022.
5. W.M.P. van der Aalst. Process Mining: A 360 Degrees Overview. In W.M.P. van der Aalst and J. Carmona, editors, *Process Mining Handbook*, volume 448 of *Lecture Notes in Business Information Processing*, pages 3–34. Springer-Verlag, Berlin, 2022.
6. W.M.P. van der Aalst, A. Adriansyah, and B. van Dongen. Replaying History on Process Models for Conformance Checking and Performance Analysis. *WIREs Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.
7. W.M.P. van der Aalst, P. Barthelmess, C.A. Ellis, and J. Wainer. Proclets: A Framework for Lightweight Interacting Workflow Processes. *International Journal of Cooperative Information Systems*, 10(4):443–482, 2001.
8. W.M.P. van der Aalst and A. Berti. Discovering Object-Centric Petri Nets. *Fundamenta Informaticae*, 175(1-4):1–40, 2020.
9. W.M.P. van der Aalst and C. Stahl. *Modeling Business Processes: A Petri Net Oriented Approach*. MIT Press, Cambridge, MA, 2011.
10. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
11. A. Artale, A. Kovtunova, M. Montali, and W.M.P. van der Aalst. Modeling and Reasoning over Declarative Data-Aware Processes with Object-Centric Behavioral Constraints. In T.T. Hildebrandt, B.F. van Dongen, M. Röglinger, and J. Mendling, editors, *International Conference on Business Process Management (BPM 2019)*, volume 11675 of *Lecture Notes in Computer Science*, pages 139–156. Springer-Verlag, Berlin, 2019.
12. A. Augusto, R. Conforti, M. Dumas, M. La Rosa, F.M. Maggi, A. Marrella, M. Mecella, and A. Soo. Automated Discovery of Process Models from Event Logs: Review and Benchmark. *IEEE Transactions on Knowledge and Data Engineering*, 31(4):686–705, 2019.
13. A. Augusto, R. Conforti, M. Marlon, M. La Rosa, and A. Polyvyanyy. Split Miner: Automated Discovery of Accurate and Simple Business Process Models from Event Logs. *Knowledge Information Systems*, 59(2):251–284, 2019.
14. R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process Mining Based on Regions of Languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *International Conference on Business Process Management (BPM 2007)*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383. Springer-Verlag, Berlin, 2007.
15. A. Berti and W.M.P. van der Aalst. Extracting Multiple Viewpoint Models from Relational Databases. In P. Ceravolo, M. van Keulen, and M.T. Gomez Lopez, editors, *Postproceedings International Symposium on Data-driven Process Discovery and Analysis*, volume 379 of *Lecture Notes in Business Information Processing*, pages 24–51. Springer-Verlag, Berlin, 2020.

16. J. Carmona, J. Cortadella, and M. Kishinevsky. A Region-Based Algorithm for Discovering Petri Nets from Event Logs. In *Business Process Management (BPM 2008)*, pages 358–373, 2008.
17. J. Carmona, B. van Dongen, A. Solti, and M. Weidlich. *Conformance Checking: Relating Processes and Models*. Springer-Verlag, Berlin, 2018.
18. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
19. J. Desel and W. Reisig. Place/Transition Nets. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 122–173. Springer-Verlag, Berlin, 1998.
20. M. Dumas, M. La Rosa, J. Mendling, and H. Reijers. *Fundamentals of Business Process Management*. Springer-Verlag, Berlin, 2018.
21. M.L. van Eck, N. Sidorova, and W.M.P. van der Aalst. Guided Interaction Exploration and Performance Analysis in Artifact-Centric Process Models. *Business and Information Systems Engineering*, 61(6):649–663, 2019.
22. D. Fahland. Describing Behavior of Processes with Many-to-Many Interactions. In S. Donatelli and S. Haar, editors, *Applications and Theory of Petri Nets 2019*, volume 11522 of *Lecture Notes in Computer Science*, pages 3–24. Springer-Verlag, Berlin, 2019.
23. D. Fahland, M. De Leoni, B. van Dongen, and W.M.P. van der Aalst. Behavioral Conformance of Artifact-Centric Process Models. In A. Abramowicz, editor, *Business Information Systems (BIS 2011)*, volume 87 of *Lecture Notes in Business Information Processing*, pages 37–49. Springer-Verlag, Berlin, 2011.
24. D. Fahland, M. de Leoni, B.F. van Dongen, and W.M.P. van der Aalst. Conformance Checking of Interacting Processes with Overlapping Instances. In S. Rinderle, F. Toumani, and K. Wolf, editors, *Business Process Management (BPM 2011)*, volume 6896 of *Lecture Notes in Computer Science*, pages 345–361. Springer-Verlag, Berlin, 2011.
25. G. Galic and M. Wolf. *Global Process Mining Survey 2021: Delivering Value with Process Analytics - Adoption and Success Factors of Process Mining*. Deloitte, 2021.
26. A.F. Ghahfarokhi, G. Park, A. Berti, and W.M.P. van der Aalst. OCEL Standard. www.ocel-standard.org, 2021.
27. G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Discovering Expressive Process Models by Clustering Log Traces. *IEEE Transaction on Knowledge and Data Engineering*, 18(8):1010–1027, 2006.
28. IEEE Task Force on Process Mining. XES Standard Definition. www.xes-standard.org, 2016.
29. K. Jensen and L.M. Kristensen. *Coloured Petri Nets*. Springer-Verlag, Berlin, 2009.
30. M. Kerremans, T. Srivastava, and F. Choudhary. Gartner Market Guide for Process Mining, Research Note G00737056. `www.gartner.com`, 2021.
31. S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering Block-structured Process Models from Event Logs: A Constructive Approach. In J.M. Colom and J. Desel, editors, *Applications and Theory of Petri Nets 2013*, volume 7927 of *Lecture Notes in Computer Science*, pages 311–329. Springer-Verlag, Berlin, 2013.
32. S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. In N. Lohmann, M. Song, and P. Wohed, editors, *Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2013)*, volume 171 of *Lecture Notes in Business Information Processing*, pages 66–78. Springer-Verlag, Berlin, 2014.
33. S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Scalable Process Discovery and Conformance Checking. *Software and Systems Modeling*, 17(2):599–631, 2018.

34. G. Li, R. Medeiros de Carvalho, and W.M.P. van der Aalst. Automatic Discovery of Object-Centric Behavioral Constraint Models. In W. Abramowicz, editor, *Business Information Systems (BIS 2017)*, volume 288 of *Lecture Notes in Business Information Processing*, pages 43–58. Springer-Verlag, Berlin, 2017.
35. X. Lu, M. Nagelkerke, D. van de Wiel, and D. Fahland. Discovering Interacting Artifacts from ERP Systems. *IEEE Transactions on Services Computing*, 8(6):861–873, 2015.
36. OMG. Business Process Model and Notation (BPMN), Version 2.0.2. Object Management Group, www.omg.org/spec/BPMN/, 2014.
37. A. Rozinat and W.M.P. van der Aalst. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 33(1):64–95, 2008.
38. M. Solé and J. Carmona. Process Mining from a Basis of State Regions. In J. Lilius and W. Penczek, editors, *Applications and Theory of Petri Nets 2010*, volume 6128 of *Lecture Notes in Computer Science*, pages 226–245. Springer-Verlag, Berlin, 2010.
39. M. Song, C.W. Günther, and W.M.P. van der Aalst. Trace Clustering in Process Mining. In D. Ardagna, editor, *BPM 2008 Workshops, Proceedings of the Fourth Workshop on Business Process Intelligence (BPI 2008)*, volume 17 of *Lecture Notes in Business Information Processing*, pages 109–120. Springer-Verlag, Berlin, 2009.
40. J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens. Leveraging Process Discovery With Trace Clustering and Text Mining for Intelligent Analysis of Incident Management Processes. In *IEEE Congress on Evolutionary Computation (CEC 2012)*, pages 1–8. IEEE Computer Society, 2012.
41. J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process Discovery using Integer Linear Programming. *Fundamenta Informaticae*, 94:387–412, 2010.