Object-Centric Process Mining: Dealing With Divergence and Convergence in Event Data

Wil M.P. van der Aalst[1,2][0000−0002−0955−6940]

[1] Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany
[2] Fraunhofer Institute for Applied Information Technology, Sankt Augustin, Germany
wvdaalst@pads.rwth-aachen.de

Abstract. Process mining techniques use event data to answer a variety of process-related questions. Process discovery, conformance checking, model enhancement, and operational support are used to improve performance and compliance. Process mining starts from recorded events that are each characterized by a case identifier, an activity name, a timestamp, and optional attributes like resource or costs. In many applications, there are multiple candidate identifiers leading to different views on the same process. Moreover, one event may be related to different cases (convergence) and, for a given case, there may be multiple instances of the same activity within a case (divergence). To create a traditional process model, the event data need to be "flattened". There are typically multiple choices possible, leading to different views that are disconnected. Therefore, one quickly loses the overview and event data need to be exacted multiple times (for the different views). Different approaches have been proposed to tackle the problem. This paper discusses the gap between real event data and the event logs required by traditional process mining techniques. The main purpose is to create awareness and to provide ways to characterize event data. A specific logging format is proposed where events can be related to objects of different types. Moreover, basic notations and a baseline discovery approach are presented to facilitate discussion and understanding.

Keywords: Process Mining · Process Discovery · Divergence · Convergence · Artifact-Centric Modeling.

1 Introduction

Operational processes are often characterized by the 80/20 rule, also known as the Pareto principle. Often, 80% of the observed process executions (cases) can be described by less than 20% of the observed process variants. This implies that the remaining 20% of the observed process executions account for 80% of the observed process variants. Therefore, it is often relatively easy to create a precise and simple process model describing 80% of the cases. However, to add the remaining 20% of the cases, discovery techniques create models that are either complex and overfitting or severely underfitting. Standard processes such as the *Purchase-to-Pay (P2P)* and *Order-to-Cash (O2C)* seem simple at first: Just a handful of activities executed in a well-defined order. Although the majority of P2P and O2C process instances can be described by a simple process model, the number of process variants may be enormous. There may be thousands of ways to execute the P2P and O2C process due to exceptions, rework, deviations, and errors. In some organizations, one can observe close to one million different ways to perform the O2C process in a single year. Unfortunately, often the 20% least frequent behavior may cause most of the compliance and performance problems. This is called *organizational friction*. *Process mining* aims to identify and remove such organizational friction.

The author started to develop the first process mining techniques in the late 1990-ties [2]. Input for process mining is an *event log*. An event log 'views' a process from a particular angle. Each event in the log refers to (1) a particular process *instance* (called *case*), (2) an *activity*, and (3) a *timestamp*. There may be additional event attributes referring to resources, people, costs, etc., but these are optional. Events logs are related to process models (discovered or hand-made). Process models can be expressed using different formalisms ranging from *Directly-Follows Graphs* (DFGs) and *accepting automata* to *Petri nets*, *BPMN diagrams*, and *UML activity diagrams*. Typically, four types of process mining are identified:

– *Process discovery*: Learning process models from event data. A discovery technique takes an event log and produces a process model without using additional information [2]. An example is the well-known Alpha-algorithm [12], which takes an event log as input and produces a Petri net explaining the behavior recorded in the log. Most of the commercial process mining tools first discover DFGs before conducting further analysis.
– *Conformance checking*: Detecting and diagnosing both differences and commonalities between an event log and a process model [15]. Conformance checking can be used to check if reality, as recorded in the log, conforms to the model and vice versa [2]. The process model used as input may be descriptive or normative. Moreover, the process model may have been made by hand or learned using process discovery.
– *Process reengineering*: Improving or extending the model based on event data. Like for conformance checking, both an event log and a process model are used as input. However, now, the goal is not to diagnose differences. The goal is to change the process model. For example, it is possible to repair the model to better reflect reality. It is also possible to enrich an existing process model with additional perspectives. For example, replay techniques can be used to show bottlenecks or resource usage. Process reengineering yields updated models. These models can be used to improve the actual processes.
– *Operational support*: Directly influencing the process by providing warnings, predictions, or recommendations [2]. Conformance checking can be done "on-the-fly" allowing people to act the moment processes deviate. Based on the model and event data related to running process instances, one can predict the remaining flow time, the likelihood of meeting the legal deadline, the

associated costs, the probability that a case will be rejected, etc. The process is not improved by changing the model, but by directly providing data-driven support in the form of warnings, predictions, and/or recommendations.

Process mining aims to provide *actionable* results, e.g., automated alerts, interventions, reconfigurations, policy changes, and redesign. The uptake of process mining is industry is accelerating in recent years. Currently, there are more than 30 commercial offerings of process mining software (e.g., Celonis, Disco, ProcessGold, myInvenio, PAFnow, Minit, QPR, Mehrwerk, Puzzledata, LanaLabs, StereoLogic, Everflow, TimelinePI, Signavio, and Logpickr).

In this paper, we challenge the following two commonly used assumptions:

– There is a *single* case notion.
– Each event refers to *precisely one* case.

We assume that there are multiple case notions (called object types) and that an event may refer to any number of objects corresponding to different object types. This idea is not new (see Section 2) and was already elaborated in [2]. However, existing process mining tools and techniques still assume that there is a single case notion and precisely one case per event.

When extracting an event log from some information system (e.g., the thousands of tables of SAP), the resulting log may suffer from *convergence* (one event is related to multiple cases) and *divergence* (independent, repeated executions of a group of activities within a single case). This may lead to the replication of events and thus misleading results (e.g., duplicated events are counted twice). It may also lead to loops in process models which are not really loops (but concurrency at the sub-instance level). These problems are partly unavoidable. However, it is good to be aware of these phenomena and to demand process mining tools supporting *object-centric process mining*.

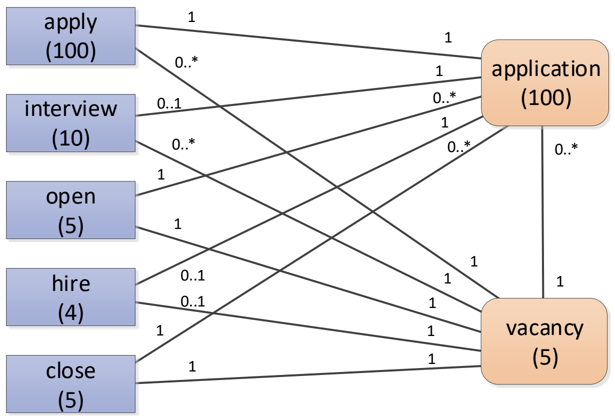

Fig. 1. A simple example explaining convergence and divergence problems. There are five activities (left-hand side) and two possible case notions (right-hand side).

To explain convergence and divergence, consider the example shown in Figure 1. In a hiring process, we deal with two types of objects: *application* and *vacancy* (see right-hand side). Moreover, assume that there are five different activities: *apply* (new application for an open vacancy), *interview* (interview with an applicant for an open vacancy), *open* (create a vacancy after which people can apply), *hire* (hire an applicant for a specific vacancy), and *close* (close the vacancy). These activities are shown on the left-hand side of Figure 1. The figure also shows cardinality constraints that need to hold at the end.

There are two possible case notions when applying traditional process mining approaches: *application* or *vacancy*. Assume that we have 100 applications and 5 vacancies. Each application refers to precisely one vacancy. Ten applications resulted in an interview and four persons were hired. Figure 1 shows the frequencies of activities and object types.

Suppose that we want to use *application* as a case notion and want to include the opening and closing of the corresponding position in the process model. This means that, when applying traditional process mining approaches, we need 100 *open* and *close* events rather than just five. This is called *convergence*. One *open* or *close* event is related to multiple cases. The problem is that events are replicated and process mining results are no longer showing to the actual number of events.

Suppose that we want to use *vacancy* as a case notion and want to include the applications and interviews of the corresponding applicants in the process model. This means that within a single case there many be many *apply* and *interview* events. Of course each *interview* event is preceded by precisely one *apply* event. However, because we cannot distinguish between the different applicants within a case, we see seemingly random interleavings of the two activities. However, there is a clear precedence at the level of individual applications (an interview never precedes an application). This is called *divergence*. Ordering information at the sub-instance level is lost, thus leading to loops in the process model that do not really exist in the real process.

Later we will show a more elaborate example, but Figure 1 already shows the problems we encounter when there is just a single one-to-many relationship. In real-life processes, we often have multiple one-to-many and many-to-many relationships, thus making process mining challenging. Object-centric process mining techniques aim to address such convergence and divergence problems.

The remainder is organized as follows. Section 2 briefly discusses related work. Section 3 introduces the problem using a simple example. Section 4 formalizes event data in a way that makes the problem explicit: An event may refer to any number of objects corresponding to different object types. A baseline discovery approach is presented in Section 5. The baseline approach uses only the directly-follows relation. Section 6 discusses approaches that go beyond the baseline approach and that aim to discover concurrency. Section 7 concludes the paper.

2 Related Work on Object-Centric Process Mining

For a basic introduction to *process mining*, we refer to [2]. Chapter 5 of the process mining book focuses on the input side of process mining. Specifically, Section 5.5 discusses the need to "flatten" event data to produce traditional process models.

Traditional process models ranging from *workflow nets* [1, 8] and *process trees* [24] to *Business Process Modeling Notation (BPMN) models* [31] and *Event-driven Process Chains (EPCs)* [34] assume a single case notion. This means that cases are considered in isolation. This is consistent with the standard notion of event logs where events refer to an activity, a timestamp, and precisely one case identifier [2].

The problem that many processes cannot be captured using a single case notion was identified early on. IBM's *FlowMark* system already supported the so-called "bundle" concept to handle cases composed of subcases [22]. This is related to the *multiple instance patterns*, i.e., a category of *workflow patterns* identified around the turn of the century [9]. One of the first process modeling notations trying to address the problem were the so-called *proclets* [6, 7]. Proclets are lightweight interacting workflow processes. By promoting interactions to first-class citizens, it is possible to model complex workflows in a more natural manner using proclets.

This was followed by other approaches such as the *artifact-centric modeling notations* [14, 16, 27, 30]. See [19] for an up-to-date overview of the challenges that arise when instances of processes may interact with each other in a one-to-many or many-to-many fashion.

Most of the work done on interacting processes with converging and diverging instances has focused on developing novel modeling notations and supporting the implementation of such processes. Only a few approaches focused on the problem in a process mining context. This is surprising since one quickly encounters the problem when applying process mining to ERP systems from SAP, Oracle, Microsoft, and other vendors of enterprise software.

In [17] techniques are described to extract "non-flat" event data from source systems and prepare these for traditional process mining. The *eXtensible Event Stream* (XES) format [23] is the current standard which requires a case notion to correlate events. XES is the official IEEE standard for storing event, supported by many process mining vendors. Next to the standard IEEE XES format [23], new storage formats such as *eXtensible Object-Centric* (XOC) [25] have been proposed to deal with object-centric data (e.g., database tables) having one-to-many and many-to-many relations. The XOC format does not require a case notion to avoid flattening multi-dimensional data. An XOC log can precisely store the evolution of the database along with corresponding events. An obvious drawback is that XOC logs tend to be very large.

The approaches described in [20, 21, 28] focus on interacting processes where each process uses its own case identifiers. In [28] interacting artifacts are discovered from ERP systems. In [20] traditional conformance checking was adapted to check compliance for interacting artifacts.

One of the main challenges is that artifact models tend to become complex and difficult to understand. In an attempt to tackle this problem, Van Eck et al. use a simpler setting with multiple perspectives, each modeled by a simple transition system [18, 35]. These are also called *artifact-centric process models* but are simpler than the models used in [14, 16, 20, 21, 27, 30, 28]. The state of a case is decomposed onto one state per perspective, thus simplifying the overall model. Relations between sub-states are viewed as correlations rather than hard causality constraints. Concurrency only exists between the different perspectives and not within an individual perspective. In a recent extension, each perspective can be instantiated multiple times, i.e., many-to-many relations between artifact types can be visualized [35].

The above techniques have the drawback that the overall process is not visualized in a single diagram, but shown as a collection of interconnected diagrams using different (sub-)case notions. The so-called *Object-Centric Behavioral Constraint* (OCBC) models address this problem and also incorporate the data perspective in a single diagram [5, 10, 13, 26]. OCBC models extend data models with a behavioral perspective. Data models can easily deal with many-to-many and one-to-many relationships. This is exploited to create process models that can also model complex interactions between different types of instances. Classical multiple-instance problems are circumvented by using the data model for event correlation. Activities are related to the data perspective and have ordering constraints inspired by declarative languages like *Declare* [11]. Instead of LTL-based constraints, simpler cardinality constraints are used. Several discovery techniques have been developed for OCBC models [26]. It is also possible to check conformance and project performance information on such models. OCBC models are appealing because they faithfully describe the relationship between behavior and data and are able to capture all information in a single integrated diagram. However, OCBC models tend to be too complex and the corresponding discovery and conformance checking techniques are not very scalable.

The complexity and scalability problems of OCBC models led to the development of the so-called *Multiple Viewpoint (MVP) models*, earlier named StarStar models [4]. MVP models are learned from data stored in relational databases. Based on the relations and timestamps in a traditional database, first, a so-called E2O graph is built that relates events and objects. Based on the E2O graph, an E2E multigraph is learned that relates events through objects. Finally, an A2A multigraph is learned to relate activities. The A2A graph shows relations between activities and each relation is based on one of the object classes used as input. This is a very promising approach because it is simple and scalable. Although this paper does not present a concrete discovery approach, the ideas are consistent with the MVP models and discovery techniques developed by Berti et al. [4].

Although commercial vendors have recognized the problems related to convergence and divergence of event data, there is no real support for concepts comparable to artifact-centric models, Object-Centric Behavioral Constraint (OCBC) models, and Multiple Viewpoint (MVP) models. Yet, there are a few initial attempts by some vendors. An example is *Celonis*, which supports the use of a secondary case identifier to avoid "Spaghetti-like" models where concurrency between sub-instances is translated into loops. The directly-follows graphs in Celonis do not consider interactions between sub-instances, thus producing simpler models. Another example is the multi-level discovery technique supported by *myInvenio*. The resulting models can be seen as simplified MVP models where different activities may correspond to different case notions (but one case notion per activity). The problem of this approach is that in reality the same event may refer to multiple case notions and choosing one is often misleading, especially since it influences the frequencies shown in the diagram.

In spite of the recent progress in process mining, problems related to multiple interacting process instances have not been solved adequately. One of the problems is the lack of standardized event data that goes beyond the "flattened" event data found in XES. Hence, process mining competitions tend to focus on classical event logs. This paper aims to shift the focus towards object-centric process mining.

3 The Problem

Event data can be found in any domain, e.g., logistics, manufacturing, finance, healthcare, customer relationship management, e-learning, and e-government. The events found in these domains typically refer to activities executed by resources at particular times and for a particular case (i.e., process instances). Process mining techniques are able to exploit such data. In this paper, we focus on process discovery. However, conformance checking, performance analysis, decision mining, organizational mining, predictions, and recommendations are also valuable forms of process mining that can benefit from the insights provided by this paper.

In a traditional event log, each event refers to a case (process instance), activity, a timestamp, and any number of additional attributes (e.g., cost, resources, etc.). The timestamp is used to order events. Since each event refers to precisely one case, each case can be represented by a *sequence of activities* (i.e., a *trace*). An example trace is $\langle a, d, d, d, e \rangle$, i.e., activity a followed by three activities d, followed by activity e. Different cases may have the same trace. Hence, an event log is a multiset of traces.[3] For example $L = [\langle a, b, c, e \rangle^{40}, \langle a, c, b, e \rangle^{30}, \langle a, d, e \rangle^{20}, \langle a, d, d, e \rangle^{5}, \langle a, d, d, d, e \rangle^{3}, \langle a, d, d, d, d, e \rangle^{2}]$ is an event log describing the traces of 100 cases. Traditional process mining techniques use such event data.

Figure 2 shows different process mining results obtained using *ProM* for an event log extracted from SAP. *ProM* provides a range of discovery techniques able to show the underlying process. Discovered process models may also show

[3] Multisets are represented using square brackets, e.g., $M = [x^2, y^3, z]$ has six elements. Unlike sets the same element can appear multiple times: $M(x) = 2$, $M(y) = 3$, and $M(z) = 1$. $[f(x) \mid x \in X]$ creates a multiset, i.e., if multiple elements x map onto the same value $f(x)$, these are counted multiple times.

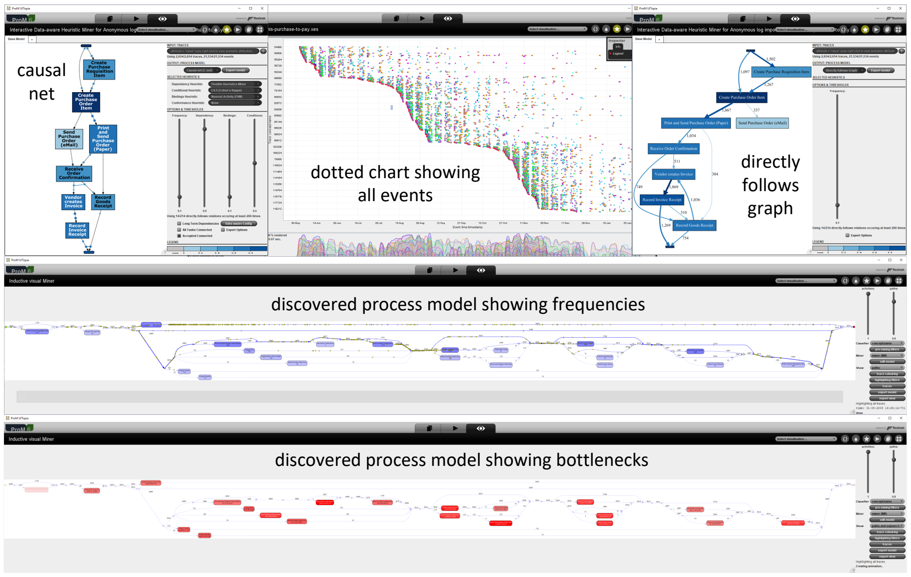
causal net
dotted chart showing all events
directly follows graph
discovered process model showing frequencies
discovered process model showing bottlenecks

Fig. 2. Various screenshots of *ProM* showing discovered process models that can be used to address performance and compliance problems.

frequencies and bottlenecks. Moreover, it is possible to perform root-cause analysis for compliance and performance problems and one can drill-down to individual cases and events. In Figure 2, we used a specific case notion allowing us to apply conventional process mining techniques working on "flattened" event logs.

The assumption that there is just one case notion and that each event refers to precisely one case is problematic in real-life processes. To illustrate this, consider the simplified order handling process from an online shop like Amazon, Alibaba, Bol, Otto, or Walmart. We are interested in the process that starts with a customer ordering products and ends with the actual delivery of all items. Figure 3 shows the activities (left-hand side) and object types (right-hand side). The four main object types are *order*, *item*, *package*, and *route*. Each order consists of one or more order lines, called items. A customer can first place an order with two items followed by an order consisting of three items. Depending on availability, items ordered for one customer are wrapped into packages. Note that one package may contain items from multiple orders. Moreover, items from a single order may be split over multiple packages. Packages are loaded into a truck that drives a route to deliver the packages in a particular order. Customers may not be home resulting in a failed delivery. The undelivered packages are stored and part of a later route. Hence, a route may involve multiple packages and the same package can be part of multiple routes.

The right-hand side of Figure 3 shows the cardinalities of the relations between the four object types. Each item is part of one order and one package.

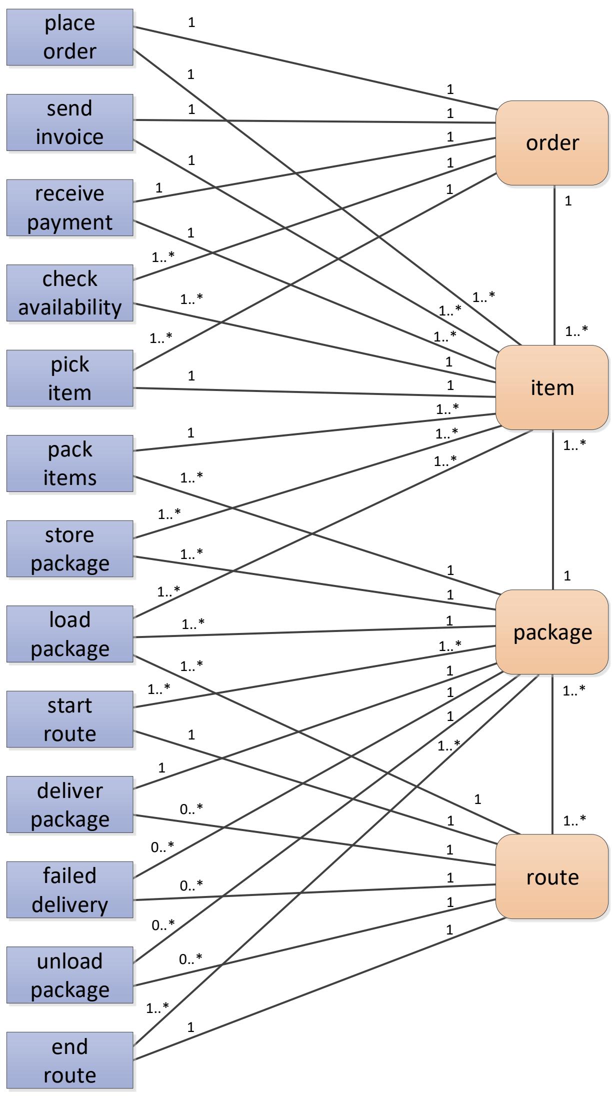

Fig. 3. Overview of the relationship between activities (left) and object types (right).

However, orders and packages may contain multiple items. There is a many-to-many relation between packages and routes. Moreover, implicitly there is also a many-to-many relation between orders and packages.

The right-hand side of Figure 3 shows the different activities. Most of the names are self-explanatory. The cardinality constraints between activities and object types help to understand the semantics of the activities.

- Activity *place_order* is the start of the order. An order is created consisting of one or more order lines (called items).
- Activity *send_invoice* is executed when all ordered items have been packed. An invoice refers to the order and the corresponding order lines.
- Activity *receive_payment* (hopefully) occurs after sending the invoice.
- Activity *check_availability* is executed for each ordered item. The check may fail and, if so, it is repeated later. The check refers to precisely one item (and the corresponding order).
- Activity *pick_item* is executed if the ordered item is available. A pick action refers to precisely one item (and the corresponding order).
- Activity *pack_items* involves possibly multiple items of possibly multiple orders from the same customer. These items are assembled into one package for a particular customer. The activity refers to one or more items and precisely one package.
- Activity *store_package* involves one package. After packing the items, the package is stored.
- Activity *load_package* involves one package and one route. Packages are loaded into the truck after the route has started.
- Activity *start_route* corresponds to the beginning of a route and involves multiple packages.
- Activity *deliver_package* corresponds to the successful delivery of a package on some route.
- Activity *failed_delivery* occurs when it is impossible to deliver a package on some route because the customer is not at home.
- Activity *unload_package* corresponds to the unloading of a package that could not be delivered. The package will be stored and later loaded onto a new route. This is repeated until the package is delivered.
- Activity *end_route* corresponds to the end of a route and involves multiple packages (delivered or not).

Figure 3 illustrates that there are multiple possible case notions. In principle, each of the object types *order*, *item*, *package*, and *route* could serve as a case notion. However, when picking one of the object types as a case notion, there may be events that refer to multiple cases and some events do not refer to any case. Therefore, it is, in general, impossible to reduce the complex reality to a classical event log.

Table 1 shows a fragment of an example instance of the problem. Each event is described by an event identifier, an activity name, a timestamp, the objects involved, and other optional attributes (here customer and costs). Let us focus on the columns showing which objects are involved. o_1, o_2, etc. are objects of type *order*, i_1, i_2, etc. are objects of type *item*, p_1, p_2, etc. are objects of type *package*, and r_1, r_2, etc. are objects of type *route*.

Following an object in a column, one can clearly see in which events (and related activities) the object is involved. For example, order o_1 is involved in events 9911 (activity *place_order*), 9912 (activity *check_availability*), 9914 (another *check_availability*), 9915 (activity *pick_item*), etc. Route r_1 is involved in

Table 1. A fragment of some event log: Each line corresponds to an event.

event	activity	timestamp	objects involved				attribute values	
identifier	name		order	item	package	route	customer	costs
...	...	...	...	...	...	...	...	...
9911	*place_order*	20-7-2019:08.15	$\{o_1\}$	$\{i_1, i_2\}$	$\emptyset$	$\emptyset$	Apple	3500€
9912	*check_availability*	20-7-2019:09.35	$\{o_1\}$	$\{i_1\}$	$\emptyset$	$\emptyset$		
9913	*place_order*	20-7-2019:09.38	$\{o_2\}$	$\{i_3, i_4, i_5\}$	$\emptyset$	$\emptyset$	Google	4129€
9914	*check_availability*	20-7-2019:10.20	$\{o_1\}$	$\{i_2\}$	$\emptyset$	$\emptyset$		
9915	*pick_item*	20-7-2019:11.05	$\{o_1\}$	$\{i_1\}$	$\emptyset$	$\emptyset$		
9916	*check_availability*	20-7-2019:11.19	$\{o_2\}$	$\{i_3\}$	$\emptyset$	$\emptyset$		
9917	*pick_item*	20-7-2019:11.55	$\{o_2\}$	$\{i_3\}$	$\emptyset$	$\emptyset$		
9918	*check_availability*	20-7-2019:13.15	$\{o_2\}$	$\{i_4\}$	$\emptyset$	$\emptyset$		
9919	*pick_item*	20-7-2019:14.25	$\{o_2\}$	$\{i_4\}$	$\emptyset$	$\emptyset$		
9920	*check_availability*	20-7-2019:15.25	$\{o_2\}$	$\{i_5\}$	$\emptyset$	$\emptyset$		
9921	*check_availability*	20-7-2019:16.34	$\{o_1\}$	$\{i_2\}$	$\emptyset$	$\emptyset$		
9922	*pick_item*	20-7-2019:16.38	$\{o_1\}$	$\{i_2\}$	$\emptyset$	$\emptyset$		
9923	*pack_items*	20-7-2019:16.44	$\emptyset$	$\{i_1, i_2, i_3\}$	$\{p_1\}$	$\emptyset$		
9924	*store_package*	20-7-2019:16.55	$\emptyset$	$\{i_1, i_2, i_3\}$	$\{p_1\}$	$\emptyset$		
9925	*start_route*	20-7-2019:16.56	$\emptyset$	$\emptyset$	$\{p_1\}$	$\{r_1\}$		
9926	*load_package*	21-7-2019:08.00	$\emptyset$	$\{i_1, i_2, i_3\}$	$\{p_1\}$	$\{r_1\}$		
9927	*send_invoice*	21-7-2019:08.17	$\{o_1\}$	$\{i_1, i_2\}$	$\emptyset$	$\emptyset$		
9928	*place_order*	21-7-2019:08.25	$\{o_3\}$	$\{i_6\}$	$\emptyset$	$\emptyset$	Microsoft	1894€
9929	*failed_delivery*	21-7-2019:08.33	$\emptyset$	$\emptyset$	$\{p_1\}$	$\{r_1\}$		
9930	*unload_package*	21-7-2019:08.56	$\emptyset$	$\emptyset$	$\{p_1\}$	$\{r_1\}$		
9931	*end_route*	21-7-2019:09.15	$\emptyset$	$\emptyset$	$\{p_1\}$	$\{r_1\}$		
9932	*check_availability*	21-7-2019:10.25	$\{o_3\}$	$\{i_6\}$	$\emptyset$	$\emptyset$		
9933	*receive_payment*	21-7-2019:11.55	$\{o_1\}$	$\{i_1, i_2\}$	$\emptyset$	$\emptyset$		
9934	*check_availability*	22-7-2019:08.19	$\{o_2\}$	$\{i_5\}$	$\emptyset$	$\emptyset$		
9935	*pick_item*	22-7-2019:08.44	$\{o_2\}$	$\{i_5\}$	$\emptyset$	$\emptyset$		
9936	*send_invoice*	22-7-2019:08.55	$\{o_2\}$	$\{i_3, i_4, i_5\}$	$\emptyset$	$\emptyset$		
9937	*receive_payment*	22-7-2019:09.15	$\{o_2\}$	$\{i_3, i_4, i_5\}$	$\emptyset$	$\emptyset$		
9938	*check_availability*	22-7-2019:10.35	$\{o_3\}$	$\{i_6\}$	$\emptyset$	$\emptyset$		
9939	*pick_item*	22-7-2019:11.23	$\{o_3\}$	$\{i_6\}$	$\emptyset$	$\emptyset$		
9941	*pack_items*	23-7-2019:09.11	$\emptyset$	$\{i_4, i_5, i_6\}$	$\{p_2\}$	$\emptyset$		
9942	*send_invoice*	22-7-2019:11.45	$\{o_3\}$	$\{i_6\}$	$\emptyset$	$\emptyset$		
9943	*store_package*	23-7-2019:09.19	$\emptyset$	$\{i_4, i_5, i_6\}$	$\{p_2\}$	$\emptyset$		
9944	*start_route*	23-7-2019:09.28	$\emptyset$	$\emptyset$	$\{p_1, p_2\}$	$\{r_2\}$		
9945	*load_package*	23-7-2019:10.05	$\emptyset$	$\{i_1, i_2, i_3\}$	$\{p_1\}$	$\{r_2\}$		
9946	*load_package*	23-7-2019:10.09	$\emptyset$	$\{i_4, i_5, i_6\}$	$\{p_2\}$	$\{r_2\}$		
9947	*deliver_package*	23-7-2019:11.25	$\emptyset$	$\emptyset$	$\{p_2\}$	$\{r_2\}$		
9948	*deliver_package*	24-7-2019:09.37	$\emptyset$	$\emptyset$	$\{p_1\}$	$\{r_2\}$		
9949	*end_route*	24-7-2019:09.48	$\emptyset$	$\emptyset$	$\{p_1, p_2\}$	$\{r_2\}$		
9950	*receive_payment*	24-7-2019:09.55	$\{o_3\}$	$\{i_6\}$	$\emptyset$	$\emptyset$		
...	...	...	...	...	...	...	...	...

events 9925 (activity *start_route*), 9926 (activity *load_package*), 9929 (activity *failed_delivery*), etc.

To cast Table 1 into a traditional event log (e.g., in XES format), we would need to have precisely one case identifier per event. This is impossible without duplicating events (convergence problem) or ordering unrelated events (divergence problem). Moreover, the example shows that a traditional event log is merely a view on the more complex reality depicted in Table 1.

4 Defining Event Data

As illustrated by the example in the previous section, we *cannot* assume that there is a *single* case notion and that each event refers to *precisely one* case. Therefore, we provide a more realistic event log notion where multiple case notions (called object types) may coexist and where an event may refer to any number of objects corresponding to different object types. To do this, we start by defining some universes.

Definition 1 (Universes). *We define the following universes to be used throughout the paper:*

- $\mathbb{U}_{ei}$ *is the universe of event identifiers,*
- $\mathbb{U}_{act}$ *is the universe of activity names,*
- $\mathbb{U}_{time}$ *is the universe of timestamps,*
- $\mathbb{U}_{ot}$ *is the universe of object types (also called classes),*
- $\mathbb{U}_{oi}$ *is the universe of object identifiers (also called entities),*
- $type \in \mathbb{U}_{oi} \to \mathbb{U}_{ot}$ *assigns precisely one type to each object identifier,*
- $\mathbb{U}_{omap} = \{omap \in \mathbb{U}_{ot} \to \mathcal{P}(\mathbb{U}_{oi}) \mid \forall_{ot \in \mathbb{U}_{ot}} \forall_{oi \in omap(ot)} type(oi) = ot\}$ *is the universe of all object mappings indicating which object identifiers are included per type,*[4]
- $\mathbb{U}_{att}$ *is the universe of attribute names,*
- $\mathbb{U}_{val}$ *is the universe of attribute values,*
- $\mathbb{U}_{vmap} = \mathbb{U}_{att} \not\rightarrow \mathbb{U}_{val}$ *is the universe of value assignments,*[5] *and*
- $\mathbb{U}_{event} = \mathbb{U}_{ei} \times \mathbb{U}_{act} \times \mathbb{U}_{time} \times \mathbb{U}_{omap} \times \mathbb{U}_{vmap}$ *is the universe of events.*

$e = (ei, act, time, omap, vmap) \in \mathbb{U}_{event}$ is an event with identifier ei, corresponding to the execution of activity act at time $time$, referring to the objects specified in $omap$, and having attribute values specified by $vmap$. Each row in Table 1 defines such an event.

Definition 2 (Event Projection). *Given* $e = (ei, act, time, omap, vmap) \in \mathbb{U}_{event}$, $\pi_{ei}(e) = ei$, $\pi_{act}(e) = act$, $\pi_{time}(e) = time$, $\pi_{omap}(e) = omap$, *and* $\pi_{vmap}(e) = vmap$.

[4] $\mathcal{P}(\mathbb{U}_{oi})$ is the powerset of the universe of object identifiers, i.e., objects types are mapped onto sets of object identifiers.

[5] $\mathbb{U}_{att} \not\rightarrow \mathbb{U}_{val}$ is the set of all partial functions mapping a subset of attribute names onto the corresponding values.

Let e_{9911} be the first event depicted in Table 1. $\pi_{ei}(e_{9911}) = 9911$, $\pi_{act}(e_{9911}) = \mathit{place\_order}$, $\pi_{time}(e_{9911}) = \text{20-7-2019:08.15}$, $\pi_{omap}(e_{9911}) = \mathit{omap}_{9911}$, and $\pi_{vmap}(e_{9911}) = \mathit{vmap}_{9911}$ such that $\mathit{omap}_{9911}(\mathit{order}) = \{o_1\}$, $\mathit{omap}_{9911}(\mathit{item}) = \{i_1, i_2\}$, $\mathit{omap}_{9911}(\mathit{package}) = \emptyset$, $\mathit{omap}_{9911}(\mathit{route}) = \emptyset$, $\mathit{vmap}_{9911}(\mathit{customer}) = \mathit{Apple}$ and $\mathit{vmap}_{9911}(\mathit{costs}) = 3500$€.

An event log is a collection of *partially ordered events*. Event identifiers are unique, i.e., two events cannot have the same event identifier.

Definition 3 (Event Log). $(E, \preceq_E)$ *is an event log with* $E \subseteq \mathbb{U}_{event}$ *and* $\preceq_E \subseteq E \times E$ *such that:*

- $\preceq_E$ *defines a partial order (reflexive, antisymmetric, and transitive),*
- $\forall_{e_1, e_2 \in E}\ \pi_{ei}(e_1) = \pi_{ei}(e_2) \ \Rightarrow\ e_1 = e_2$*, and*
- $\forall_{e_1, e_2 \in E}\ e_1 \preceq_E e_2 \ \Rightarrow\ \pi_{time}(e_1) \leq \pi_{time}(e_2)$.

Table 1 shows an example of an event log. Note that the values in the first column need to be unique and time is non-decreasing. Although Table 1 is totally ordered, we can also consider partially ordered events logs. There are two main reasons to use partially ordered event logs:

- When the timestamps are coarse-grained, we may not know the actual order. For example, event logs may only show the day and not the precise time. In such cases, we do not want to order the events taking place on the same day.
- We may exploit information about causality. When two causally unrelated events occur, we may deliberately not use the order in which they occurred. This makes it easy to create better process models that also capture concurrency without seeing all possible interleavings.

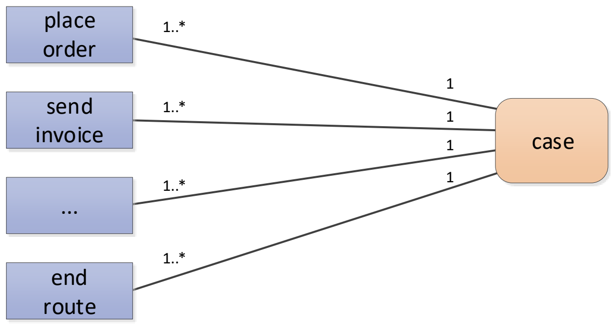

Fig. 4. Classical event log where each event refers to precisely one case identifier.

We advocate using event logs that follow Definition 3 rather than flattened, totally ordered event logs using a single case notion. Note that conventional event logs are a special case of Definition 3 as illustrated by the following definition.

Definition 4 (Classical Event Log). *An event log $(E, \preceq_E)$ is a classical event log if and only if $\preceq_E$ is a total order and there exists an $ot \in \mathbb{U}_{ot}$ such that for any $e \in E$: $|\pi_{omap}(e)(ot)| = 1$ and for all $ot' \in \mathbb{U}_{ot} \setminus \{ot\}$: $\pi_{omap}(e)(ot') = \emptyset$.*

By comparing Figure 3 and Figure 4 one can clearly see that in most cases it does not make any sense to try and straightjacket event data into a conventional event log. This only makes sense for selected views on the event data.

In Table 1, *check_availability* events refer to an item and an order. One could argue that the reference to the order is redundant, after a *place_order* event the items are linked to the order and do not need to be repeated. Similarly, *store_package* events refer to the items in the corresponding package, but one could argue that after the *pack_items* event, these are known and the relation does not need to be repeated in later events related to the package. Compare Figure 5 to Figure 3. Both show the relation between the activities (left-hand side) and object types (right-hand side). However, Figure 5 aims to remove some of the redundancy (i.e., not include information that can be derived from other events). Table 2 shows the event log where fewer objects are associated with events (based on Figure 5).

The choice between Table 1 and Table 2 depends on the intended process scope. For example, are *check_availability* and *pick_item* part of the lifecycle of an order? Are *store_package*, *load_package*, *send_invoice*, and *receive_payment* part of the lifecycle of an item? Are *start_route* and *end_route* part of the lifecycle of a package? These are important scoping choices that influence the models generated using process mining techniques.

In Table 2, *load_package*, *deliver_package*, *failed_delivery*, and *unload_package* events still refer to both a package and a route. This is due to the fact that the same package may be part of multiple routes.

5 A Baseline Discovery Approach

The goal of this paper is to show the challenges related to object-centric process mining, and not to present a specific process mining algorithm. Nevertheless, we describe a baseline discovery approach using more realistic event data as specified in Definition 3, i.e., a collection of events pointing to any number of objects and a partial order $(E, \preceq_E)$.

Any event log $(E, \preceq_E)$ can be projected onto a selected object type ot. Just the events that refer to objects of type ot are kept. The partial order is updated based on the selected object type ot. In the updated partial order, events are related when they share a particular object. To ensure that the resulting order is indeed a partial order, we need to take the transitive closure.

Definition 5 (Object Type Projection). *Let $(E, \preceq_E)$ be an event log and $ot \in \mathbb{U}_{ot}$ an object type. $(E^{ot}, \preceq_E^{ot})$ is the event log projected onto object type*

Table 2. A modified version of the event log in Table 1. Still, each line corresponds to an event, but events refer to a minimal amount of object types. Also, the additional attributes are not shown.

event identifier	activity name	timestamp	objects involved			
			order	item	package	route
...	...	...	...	...	...	...
9911	*place_order*	20-7-2019:08.15	$\{o_1\}$	$\{i_1, i_2\}$	$\emptyset$	$\emptyset$
9912	*check_availability*	20-7-2019:09.35	$\emptyset$	$\{i_1\}$	$\emptyset$	$\emptyset$
9913	*place_order*	20-7-2019:09.38	$\{o_2\}$	$\{i_3, i_4, i_5\}$	$\emptyset$	$\emptyset$
9914	*check_availability*	20-7-2019:10.20	$\emptyset$	$\{i_2\}$	$\emptyset$	$\emptyset$
9915	*pick_item*	20-7-2019:11.05	$\emptyset$	$\{i_1\}$	$\emptyset$	$\emptyset$
9916	*check_availability*	20-7-2019:11.19	$\emptyset$	$\{i_3\}$	$\emptyset$	$\emptyset$
9917	*pick_item*	20-7-2019:11.55	$\emptyset$	$\{i_3\}$	$\emptyset$	$\emptyset$
9918	*check_availability*	20-7-2019:13.15	$\emptyset$	$\{i_4\}$	$\emptyset$	$\emptyset$
9919	*pick_item*	20-7-2019:14.25	$\emptyset$	$\{i_4\}$	$\emptyset$	$\emptyset$
9920	*check_availability*	20-7-2019:15.25	$\emptyset$	$\{i_5\}$	$\emptyset$	$\emptyset$
9921	*check_availability*	20-7-2019:16.34	$\emptyset$	$\{i_2\}$	$\emptyset$	$\emptyset$
9922	*pick_item*	20-7-2019:16.38	$\emptyset$	$\{i_2\}$	$\emptyset$	$\emptyset$
9923	*pack_items*	20-7-2019:16.44	$\emptyset$	$\{i_1, i_2, i_3\}$	$\{p_1\}$	$\emptyset$
9924	*store_package*	20-7-2019:16.55	$\emptyset$	$\emptyset$	$\{p_1\}$	$\emptyset$
9925	*start_route*	20-7-2019:16.56	$\emptyset$	$\emptyset$	$\emptyset$	$\{r_1\}$
9926	*load_package*	21-7-2019:08.00	$\emptyset$	$\emptyset$	$\{p_1\}$	$\{r_1\}$
9927	*send_invoice*	21-7-2019:08.17	$\{o_1\}$	$\emptyset$	$\emptyset$	$\emptyset$
9928	*place_order*	21-7-2019:08.25	$\{o_3\}$	$\{i_6\}$	$\emptyset$	$\emptyset$
9929	*failed_delivery*	21-7-2019:08.33	$\emptyset$	$\emptyset$	$\{p_1\}$	$\{r_1\}$
9930	*unload_package*	21-7-2019:08.56	$\emptyset$	$\emptyset$	$\{p_1\}$	$\{r_1\}$
9931	*end_route*	21-7-2019:09.15	$\emptyset$	$\emptyset$	$\emptyset$	$\{r_1\}$
9932	*check_availability*	21-7-2019:10.25	$\emptyset$	$\{i_6\}$	$\emptyset$	$\emptyset$
9933	*receive_payment*	21-7-2019:11.55	$\{o_1\}$	$\emptyset$	$\emptyset$	$\emptyset$
9934	*check_availability*	22-7-2019:08.19	$\emptyset$	$\{i_5\}$	$\emptyset$	$\emptyset$
9935	*pick_item*	22-7-2019:08.44	$\emptyset$	$\{i_5\}$	$\emptyset$	$\emptyset$
9936	*send_invoice*	22-7-2019:08.55	$\{o_2\}$	$\emptyset$	$\emptyset$	$\emptyset$
9937	*receive_payment*	22-7-2019:09.15	$\{o_2\}$	$\emptyset$	$\emptyset$	$\emptyset$
9938	*check_availability*	22-7-2019:10.35	$\emptyset$	$\{i_6\}$	$\emptyset$	$\emptyset$
9939	*pick_item*	22-7-2019:11.23	$\emptyset$	$\{i_6\}$	$\emptyset$	$\emptyset$
9941	*pack_items*	23-7-2019:09.11	$\emptyset$	$\{i_4, i_5, i_6\}$	$\{p_2\}$	$\emptyset$
9942	*send_invoice*	22-7-2019:11.45	$\{o_3\}$	$\emptyset$	$\emptyset$	$\emptyset$
9943	*store_package*	23-7-2019:09.19	$\emptyset$	$\emptyset$	$\{p_2\}$	$\emptyset$
9944	*start_route*	23-7-2019:09.28	$\emptyset$	$\emptyset$	$\emptyset$	$\{r_2\}$
9945	*load_package*	23-7-2019:10.05	$\emptyset$	$\emptyset$	$\{p_1\}$	$\{r_2\}$
9946	*load_package*	23-7-2019:10.09	$\emptyset$	$\emptyset$	$\{p_2\}$	$\{r_2\}$
9947	*deliver_package*	23-7-2019:11.25	$\emptyset$	$\emptyset$	$\{p_2\}$	$\{r_2\}$
9948	*deliver_package*	24-7-2019:09.37	$\emptyset$	$\emptyset$	$\{p_1\}$	$\{r_2\}$
9949	*end_route*	24-7-2019:09.48	$\emptyset$	$\emptyset$	$\emptyset$	$\{r_2\}$
9950	*receive_payment*	24-7-2019:09.55	$\{o_3\}$	$\emptyset$	$\emptyset$	$\emptyset$
...	...	...	...	...	...	...

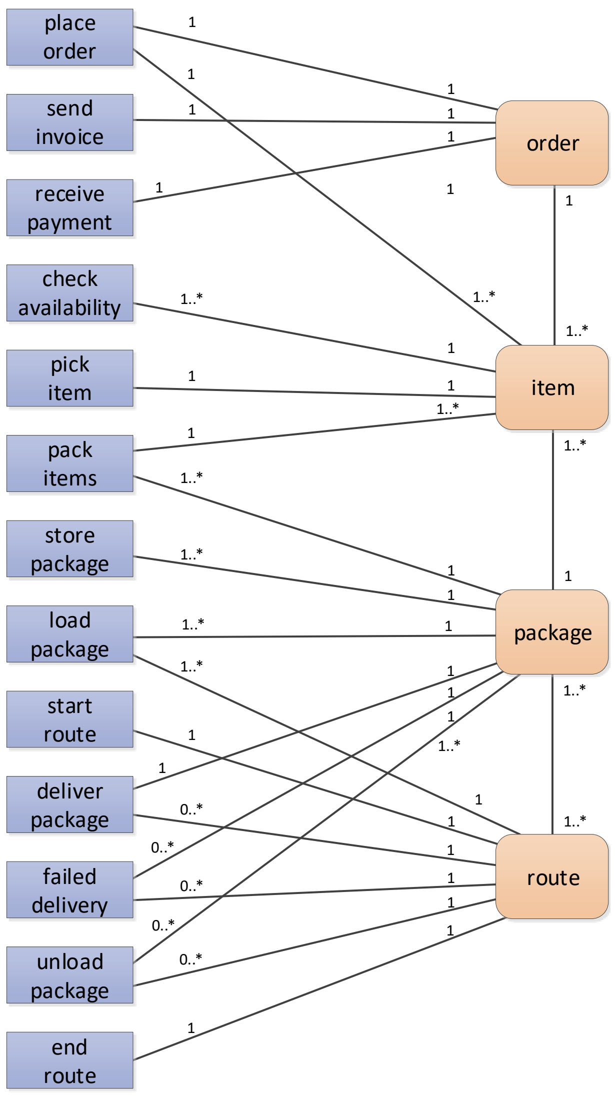

Fig. 5. Relating activities (left-hand side) to object types (right-hand side) while minimizing redundancy compared to Figure 3.

ot where $E^{ot} = \{e \in E \mid \pi_{omap}(e)(ot) \neq \emptyset\}$ *and* $\preceq_E^{ot} = \{(e_1, e_2) \in E^{ot} \times E^{ot} \mid e_1 \preceq_E e_2 \ \wedge \ \pi_{omap}(e_1)(ot) \cap \pi_{omap}(e_2)(ot) \neq \emptyset\}^*$.[6]

It is easy to verify that $\preceq_E^{ot}$ is indeed reflexive, antisymmetric, and transitive. Also note that if there are three events $e_1 \preceq_E e_2 \preceq_E e_3$ with $\pi_{omap}(e_1)(ot) =$

[6] R^* is the transitive closure of relation R. Hence, $\preceq_E^{ot}$ is a partial order (reflexive, antisymmetric, and transitive).

$\{o_1\}$, $\pi_{omap}(e_2)(ot) = \{o_1, o_2\}$, and $\pi_{omap}(e_3)(ot) = \{o_2\}$, then $e_1 \preceq_E^{ot} e_3$ although $\pi_{omap}(e_1)(ot) \cap \pi_{omap}(e_3)(ot) = \emptyset$.

Projections can be generalized to multiple object types. For $OT \in \mathbb{U}_{ot}$: $E^{OT} = \{e \in E \mid \exists_{ot \in OT}\ \pi_{omap}(e)(ot) \neq \emptyset\}$ and $\preceq_E^{OT} = \{(e_1, e_2) \in E^{OT} \times E^{OT} \mid e_1 \preceq_E e_2\ \wedge\ \exists_{ot \in OT}\ \pi_{omap}(e_1)(ot) \cap \pi_{omap}(e_2)(ot) \neq \emptyset\}^*$.

We would like to discover Directly-Follows Graphs (DFGs) based on the projections specified in Definition 5. To do this, we use the well-known covering relation to capture the direct causal relations between events. The covering relation is the equivalent of the transitive reduction of a finite directed acyclic graph, but applied to relations rather than graphs.

Definition 6 (Covering Relation). *Let $\preceq$ be a partial order (reflexive, anti-symmetric, and transitive). $\prec = \{(x, y) \in \preceq \mid x \neq y\}$. $\lessdot$ is the covering relation of $\preceq$, i.e., $\lessdot = \{(x, y) \in \prec \mid \nexists_z\ x \prec z \prec y\}$.*

We can construct the covering relation for any partially ordered set of events. It is known that the covering relation is unique. The graphical representation of the partial order based on the covering relation is also known as the Hasse diagram. $\lessdot_E$ and $\lessdot_E^{ot}$ refer to the covering relations of $\preceq_E$ and $\preceq_E^{ot}$ respectively.

Using the covering relation $\lessdot_E^{ot}$ for an event log $(E, \preceq_E)$ projected onto an object type ot, we can construct a variant of the Directly-Follows Graph (DFG). However, there are two differences with a normal DFG: we consider partial orders and focus on the projection.

Definition 7 (Directly-Follows Graph). *Let $(E, \preceq_E)$ be an event log and $ot \in \mathbb{U}_{ot}$ an object type. (A^{ot}, R^{ot}) with $A^{ot} = [\pi_{act}(e) \mid e \in E^{ot}]$ and $R^{ot} = [(\pi_{act}(e_1), \pi_{act}(e_2)) \mid (e_1, e_2) \in \lessdot_E^{ot}]$ is the Directly-Follows Graph (DFG) for object type ot.*

The resulting DFG (A^{ot}, R^{ot}) has a multiset of activity nodes A^{ot} and a multiset of arcs R^{ot}. Both are multisets, because we would like to keep track of frequencies. Given some activity a, $A^{ot}(a)$ is the number of a events that refer to an object of type ot. Given a pair of activities (a_1, a_2), $R^{ot}(a_1, a_2)$ is the number times an a_1 event was causally followed by an a_2 event where both events shared an object of type ot.

Figure 6 shows an example DFG (without multiplicities) for each object type using the event log illustrated by Table 1. Note that we also indicate the initial and final activities (shown using the incoming arcs and outgoing arcs). This can be achieved by adding a dummy start and end activity to each object type. The dummy start corresponds to the creation of the object. The dummy end activity corresponds to its completion. These dummy activities are not shown in Figure 6, but the corresponding arcs are.

As mentioned before, the scoping of object identifiers greatly influences the process models that are returned. To illustrate this, consider the reduced event log shown in Table 2 again. Figure 7 shows the DFGs (again without multiplicities) for each object type using this reduced event log. As before, we indicate start and end activities.

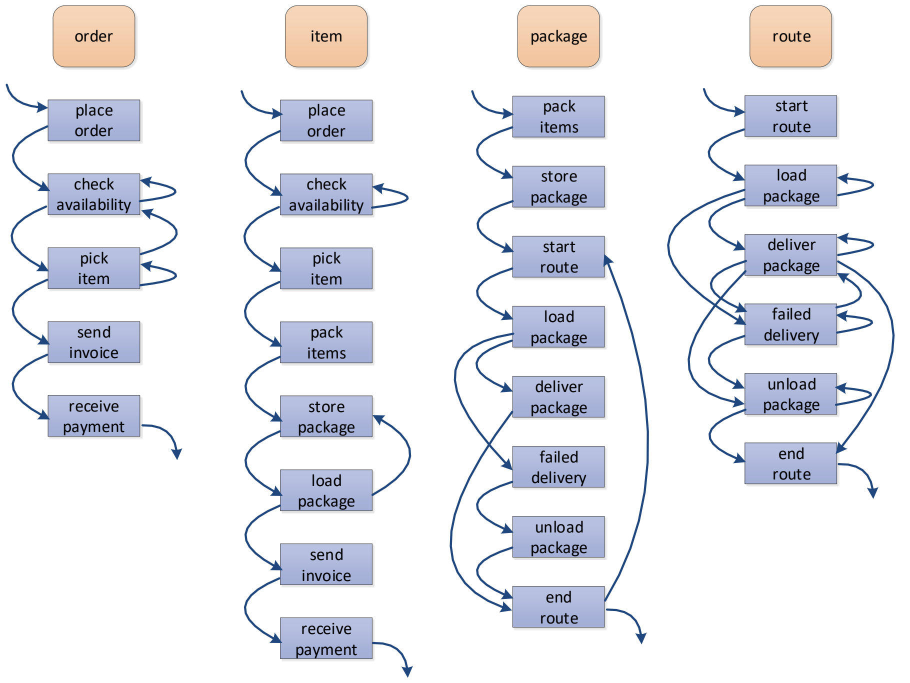

Fig. 6. DFGs per object type learned from Table 1. Not all arcs are included and also note that Table 1 is just an excerpt of the larger event log.

The different DFGs can be simply merged into a *labeled multigraph* where the arcs correspond to specific object types. The arcs are now labeled with object types. For example, (a_1, ot, a_2) is the arc connecting activities a_1 and a_2 via objects of type ot. Two arcs may connect the same pair of activities. As before, we use multisets to represent cardinalities, i.e., $R(a_1, ot, a_2)$ is the frequency of the arc connecting activities a_1 and a_2 via objects of type ot.

Definition 8 (Overall Directly-Follows Multigraph). *Let* $(E, \preceq_E)$ *be an event log.* (A, R) *with* $A = [\pi_{act}(e) \mid \exists_{ot \in \mathbb{U}_{ot}}\ e \in E^{ot}]$ *and* $R = [(\pi_{act}(e_1), ot, \pi_{act}(e_2)) \mid \exists_{ot \in \mathbb{U}_{ot}}\ (e_1, e_2) \in \lessdot_E^{ot}]$ *is the Overall Directly-Follows Multigraph (ODFM).*

In general, the Overall Directly-Follows Multigraph (ODFM) will often be too complicated to understand easily. Nevertheless, it is valuable to see the whole event log with multiple case notions in a single diagram. To simplify the multigraph, it is possible to consider any subset of object types $OT \subseteq \mathbb{U}_{ot}$.

Definition 9 (Selected Directly-Follows Multigraph). *Let* $(E, \preceq_E)$ *be an event log and* $OT \subseteq \mathbb{U}_{ot}$ *a set of object types.* (A^{OT}, R^{OT}) *with* $A^{OT} = [\pi_{act}(e) \mid \exists_{ot \in OT}\ e \in E^{ot}]$ *and* $R^{OT} = [(\pi_{act}(e_1), ot, \pi_{act}(e_2)) \mid \exists_{ot \in OT}\ (e_1, e_2) \in \lessdot_E^{ot}]$ *is the Selected Directly-Follows Multigraph (SDFM).*

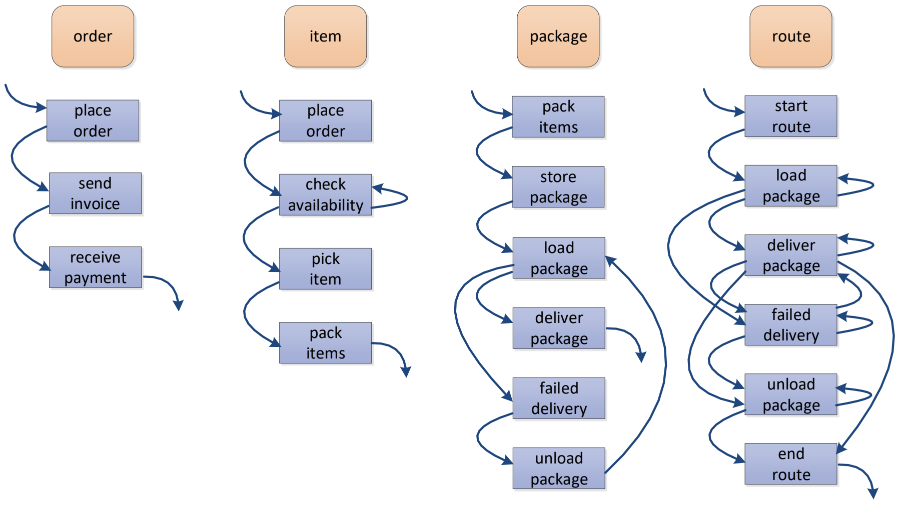

Fig. 7. DFGs per object type learned from Table 2.

The Selected Directly-Follows Multigraph (SDFM) selects two or more object types to create a particular view on the event data. As an example, we can take $OT = \{order, item, package\}$ as shown in Figure 8. The object types included are *order*, *item*, and *package*, i.e., only *route* is excluded to provide the view. The arcs are colored based on the corresponding object types. For clarity, also the names have been added. Again we do not show multiplicities on nodes and arcs. Also note that the start and end activities are indicated.

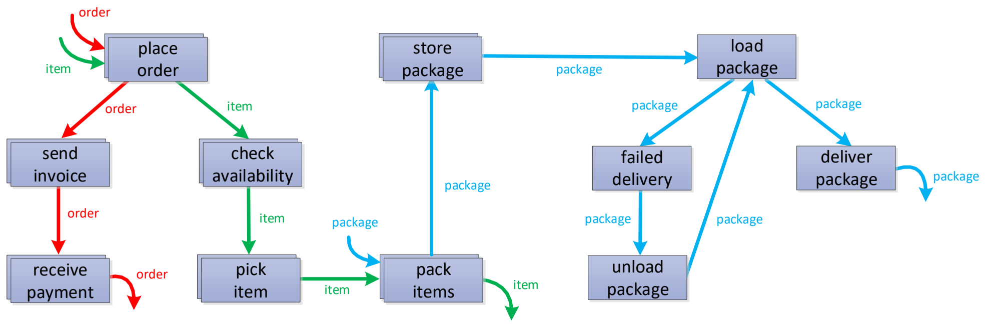

Fig. 8. Directly-Follows Multigraph (DFM) learned from Table 2 for the object types *order*, *item*, and *package*.

An ODFM is a special case of SDFM, i.e., $OT = \mathbb{U}_{ot}$. Therefore, we will use the term Directly-Follows Multigraph (DFM) to refer to both.

The DFM shown in Figure 8 illustrates that it is possible to deal with non-flattened event data. The diagram shows the relationships between different activities that are connected through various types of objects. DFMs are not as easy to interpret as traditional DFGs using a single case notion. However, trying to flatten the model by straightjacketing the event data into a traditional event log will often lead to convergence and divergence problems.

Given a DFM, it is easy to select a desired case notion, generate a conventional flat event log, and apply standard process mining techniques.

6 Beyond Directly-Follows Graphs

The Directly-Follows Multigraph (DFM) does not use higher-level process modeling constructs (e.g., XOR/OR/AND-splits/joins). Note that an event log $(E, \preceq_E)$, as defined in this paper, is a partial order. The partial order can take into account causality. Assume that $\preceq_E$ is the reflexive transitive closure of $\{(e_1, e_2), (e_1, e_3), (e_2, e_4), (e_3, e_4)\}$. The order of e_2 and e_3 is not fixed, but both causally depend on e_1. Event e_4 causally depends on e_2 and e_3. Normally, the Directly Follows Graph (DFG) does not take causality into account. In the example, the temporal ordering of e_2 and e_3 influences the graph constructed even though they do not depend on each other.

Although the partial order can take into account causality, the resulting DFM does not explicitly show concurrency. However, traditional process mining approaches can be used starting from event data projected onto a specific object type. Recall that the DFG for object type ot, i.e., (A^{ot}, R^{ot}), is based on a projected event log. We can use the same approach in conjunction with existing process discovery techniques. Two examples are the Petri-net-based place discovery technique presented in [3] and the *Declare*-based discovery techniques presented in [29, 32, 33]. In [3] monotonicity results are used to exploit finding places that are constraining behavior to the behavior seen. In [29, 32, 33] LTL-based declarative constraints are learned. Also places can be viewed as constraints. Note that a Petri net without places allows for any behavior of the transitions (activities) included. Hence, process discovery can be viewed as learning constraints. This view is compatible with the orthogonal nature of the different object types in a DFM. Therefore, it is not difficult to enhance DFMs such as the one shown in Figure 8 with more sophisticated constraints (e.g., places or LTL-based constraints).

7 Conclusion

This paper focused on the limitations of process mining techniques that assume a single case notion and just one case per event. Yet, existing approaches assume "flattened event data" (e.g., stored using XES or a CSV file with one column for the case identifier). Real-life processes are often more complex, not allowing for these simplifying assumptions. Flattened event data only provide one of many possible views, leading to convergence and divergence problems.

To address the problem, we proposed a more faithful event log notion $(E, \preceq_E)$ where events can refer to any number of objects and these may be of different object types. Hence, events can depend on each other in different ways. Moreover, we assume partially ordered events. For example, events may refer to mixtures of orders, items, packages, and delivery routes. The Directly-Follows Multigraph (DFM) can be used to get a more holistic view on the process.

The paper is also a "call for action". First of all, it is important to extract more realistic event logs. Currently, techniques developed in research and the tools provided by vendors assume "flat" event data (e.g., in XES format), because it is the information widely available (in public data sets and the data sets used in competitions). However, the data stored in information systems are not flat. Availability of more realistic event data will positively influence research and tools. Second, novel techniques are needed. The DFM is just a starting point for more sophisticated object-centric process mining techniques. However, it is vital to keep things simple and avoid the complexity associated with artifact-centric approaches. Whereas the focus in this paper is on process discovery, the insights also apply to other forms of process mining such as conformance checking, bottleneck analysis, and operational support (e.g., prediction).

Acknowledgments: We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research.

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. W.M.P. van der Aalst. *Process Mining: Data Science in Action.* Springer-Verlag, Berlin, 2016.
3. W.M.P. van der Aalst. Discovering the "Glue" Connecting Activities - Exploiting Monotonicity to Learn Places Faster. In F. de Boer, M. Bonsangue, and J. Rutten, editors, *It's All About Coordination*, Lecture Notes in Computer Science, pages 1–20. Springer-Verlag, Berlin, 2018.
4. W.M.P. van der Aalst. StarStar Models: Using Events at Database Level for Process Analysis. In P. Ceravolo, M.T. Gomez Lopez, and M. van Keulen, editors, *International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2018)*, volume 2270 of *CEUR Workshop Proceedings*, pages 60–64. CEUR-WS.org, 2018.
5. W.M.P. van der Aalst, A. Artale, M. Montali, and S. Tritini. Object-Centric Behavioral Constraints: Integrating Data and Declarative Process Modelling. In *Proceedings of the 30th International Workshop on Description Logics (DL 2017)*, volume 1879 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.
6. W.M.P. van der Aalst, P. Barthelmess, C.A. Ellis, and J. Wainer. Workflow Modeling using Proclets. In O. Etzion and P. Scheuermann, editors, *7th International Conference on Cooperative Information Systems (CoopIS 2000)*, volume 1901 of *Lecture Notes in Computer Science*, pages 198–209. Springer-Verlag, Berlin, 2000.
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, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, and M.T. Wynn. Soundness of Workflow Nets: Classification, Decidability, and Analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.
9. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
10. W.M.P. van der Aalst, G. Li, and M. Montali. Object-Centric Behavioral Constraints. *CoRR*, abs/1703.05740, 2017.
11. W.M.P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative Workflows: Balancing Between Flexibility and Support. *Computer Science - Research and Development*, 23(2):99–113, 2009.
12. 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.
13. A. Artale, D. Calvanese, M. Montali, and W.M.P. van der Aalst. Enriching Data Models with Behavioral Constraints. In S. Borgo, editor, *Ontology Makes Sense (Essays in honor of Nicola Guarino)*, pages 257–277. IOS Press, 2019.
14. K. Bhattacharya, C. Gerede, R. Hull, R. Liu, and J. Su. Towards Formal Analysis of Artifact-Centric Business Process Models. 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 288–304. Springer-Verlag, Berlin, 2007.
15. J. Carmona, B. van Dongen, A. Solti, and M. Weidlich. *Conformance Checking: Relating Processes and Models*. Springer-Verlag, Berlin, 2018.
16. D. Cohn and R. Hull. Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. *IEEE Data Engineering Bulletin*, 32(3):3–9, 2009.
17. E. González López de Murillas, H.A. Reijers, and W.M.P. van der Aalst. Connecting Databases with Process Mining: A Meta Model and Toolset. In R. Schmidt, W. Guedria, I. Bider, and S. Guerreiro, editors, *Enterprise, Business-Process and Information Systems Modeling (BPMDS 2015)*, volume 248 of *Lecture Notes in Business Information Processing*, pages 231–249. Springer-Verlag, Berlin, 2016.
18. M.L. van Eck, N. Sidorova, and W.M.P. van der Aalst. Guided Interaction Exploration in Artifact-centric Process Models. In *IEEE Conference on Business Informatics (CBI 2017)*, pages 109–118. IEEE Computer Society, 2017.
19. 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.
20. 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.
21. D. Fahland, M. De Leoni, B. van Dongen, and W.M.P. van der Aalst. Many-to-Many: Some Observations on Interactions in Artifact Choreographies. In D. Eichhorn, A. Koschmider, and H. Zhang, editors, *Proceedings of the 3rd Central-European Workshop on Services and their Composition (ZEUS 2011)*, CEUR Workshop Proceedings, pages 9–15. CEUR-WS.org, 2011.
22. IBM. *IBM MQSeries Workflow - Getting Started With Buildtime*. IBM Deutschland Entwicklung GmbH, Boeblingen, Germany, 1999.

23. IEEE Task Force on Process Mining. XES Standard Definition. www.xes-standard.org, 2013.
24. 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.
25. G. Li, E. González López de Murillas, R. Medeiros de Carvalho, and W.M.P. van der Aalst. Extracting Object-Centric Event Logs to Support Process Mining on Databases. In J. Mendling and H. Mouratidis, editors, *Information Systems in the Big Data Era, CAiSE Forum 2018*, volume 317 of *Lecture Notes in Business Information Processing*, pages 182–199. Springer-Verlag, Berlin, 2018.
26. 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.
27. N. Lohmann. Compliance by Design for Artifact-Centric Business Processes. In S. Rinderle, F. Toumani, and K. Wolf, editors, *Business Process Management (BPM 2011)*, volume 6896 of *Lecture Notes in Computer Science*, pages 99–115. Springer-Verlag, Berlin, 2011.
28. 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.
29. F.M. Maggi, R.P. Jagadeesh Chandra Bose, and W.M.P. van der Aalst. Efficient Discovery of Understandable Declarative Process Models from Event Logs. In J. Ralyte, X. Franch, S. Brinkkemper, and S. Wrycza, editors, *International Conference on Advanced Information Systems Engineering (Caise 2012)*, volume 7328 of *Lecture Notes in Computer Science*, pages 270–285. Springer-Verlag, Berlin, 2012.
30. A. Nigam and N.S. Caswell. Business artifacts: An Approach to Operational Specification. *IBM Systems Journal*, 42(3):428–445, 2003.
31. OMG. Business Process Model and Notation (BPMN). Object Management Group, formal/2011-01-03, 2011.
32. M. Rovani, F.M. Maggi, M. de Leoni, and W.M.P. van der Aalst. Declarative Process Mining in Healthcare. *Expert Systems With Applications*, 42(23):9236–9251, 2015.
33. R.P. Jagadeesh Chandra, F.M. Maggi, and W.M.P. van der Aalst. Enhancing Declare Maps Based on Event Correlations. In F. Daniel, J. Wang, and B. Weber, editors, *International Conference on Business Process Management (BPM 2013)*, volume 8094 of *Lecture Notes in Computer Science*, pages 97–112. Springer-Verlag, Berlin, 2013.
34. A.W. Scheer. *Business Process Engineering: Reference Models for Industrial Enterprises.* Springer-Verlag, Berlin, 1994.
35. M.L. van Eck, N. Sidorova, and W.M.P. van der Aalst. Multi-instance Mining: Discovering Synchronisation in Artifact-Centric Processes. In F. Daniel, Q.Z. Sheng, and H. Motahari, editors, *Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2018)*, volume 342 of *Lecture Notes in Business Information Processing*, pages 18–30. Springer-Verlag, Berlin, 2018.