Fundamenta Informaticae 175 (2020) 1–40
DOI 10.3233/FI-2020-1946
IOS Press

Discovering Object-centric Petri Nets

Wil M.P. van der Aalst[*]**, Alessandro Berti**
Process and Data Science (PADS), RWTH Aachen University
Aachen, Germany
Fraunhofer Institute for Applied Information Technology
Sankt Augustin, Germany
{*wvdaalst,a.berti*}*@pads.rwth-aachen.de*

Abstract. Techniques to discover Petri nets from event data assume precisely one case identifier per event. These case identifiers are used to correlate events, and the resulting discovered Petri net aims to describe the life-cycle of individual cases. In reality, there is not one possible case notion, but multiple intertwined case notions. For example, events may refer to mixtures of orders, items, packages, customers, and products. A package may refer to multiple items, multiple products, one order, and one customer. Therefore, we need to assume that each event refers to a collection of objects, each having a type (instead of a single case identifier). Such *object-centric event logs* are closer to data in real-life information systems. From an object-centric event log, we want to discover an *object-centric Petri net* with places that correspond to object types and transitions that may consume and produce collections of objects of different types. Object-centric Petri nets visualize the complex relationships among objects from different types. This paper discusses a novel process discovery approach implemented in PM4Py. As will be demonstrated, it is indeed feasible to discover holistic process models that can be used to drill-down into specific viewpoints if needed.

Keywords: Process mining, Petri nets, Process discovery, Multiple viewpoint models

[*]Address for correspondence: Process and Data Science (PADS), RWTH Aachen University, Aachen, Germany, Fraunhofer Institute for Applied Information Technology, Sankt Augustin, Germany

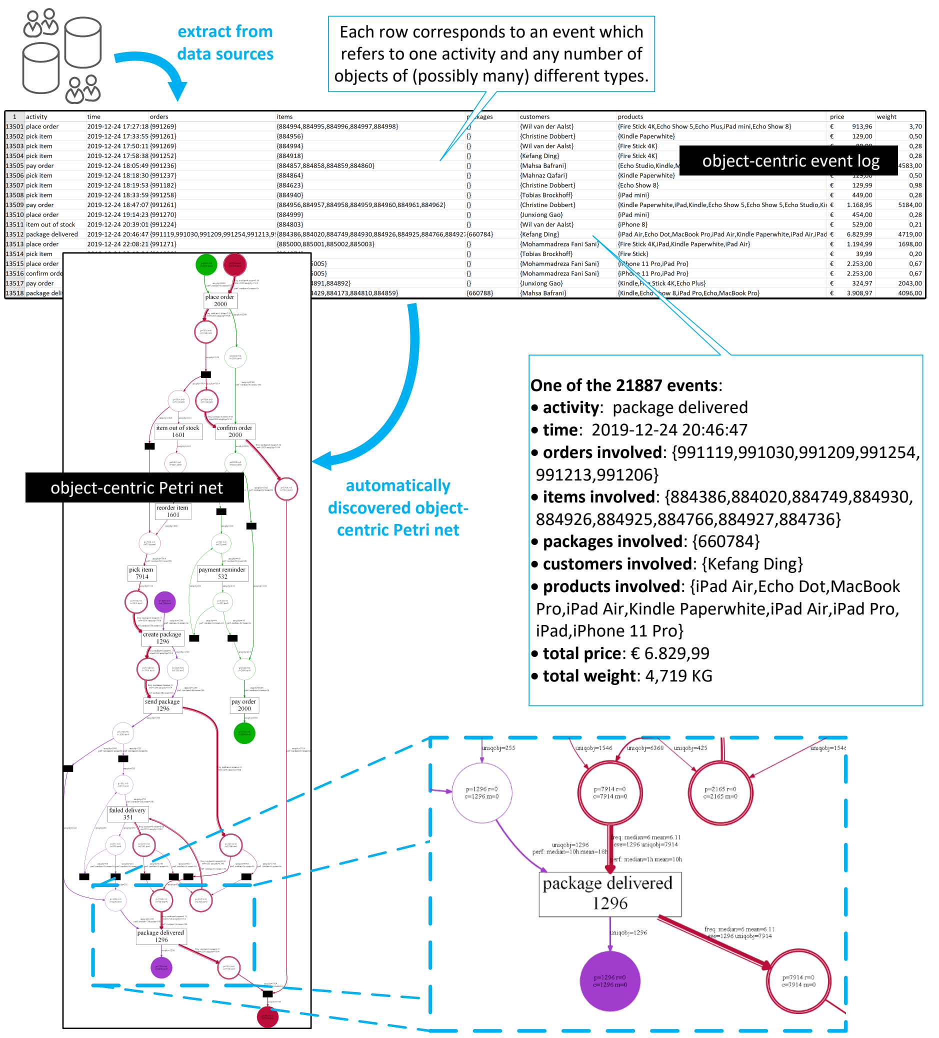
1. Introduction

The synthesis of "higher-level" process models from "lower-level" behavioral specifications has been subject of active research for decades. Examples of such "higher-level" process models are (colored) Petri nets, BPMN models, Statecharts, etc. Examples of "lower-level" behavioral specifications serving as input for synthesis are transition systems, languages, partial orders, and scenarios. In the context of Petri nets, the "Theory of Regions" has been very influential. Regions were introduced for elementary nets and transition systems in the seminal paper [1]. The goal was to create a Petri net with a reachability graph that is isomorphic to the transition system used as input. The core idea has been generalized in numerous directions. Different classes of target models have been investigated [2, 3, 4, 5], e.g., bisimilar Place Transition (P/T) nets [6], Petri nets with arc weights [7, 8], Petri nets with a/sync connections [9], τ-nets [5], zero-safe nets [10], etc. Typically, a transition system is used as input. However, there are various region-based approaches taking as input languages [11, 12, 13, 14, 15, 16], partial orders/scenarios [17, 18, 19], or other "lower-level" behavioral specifications.

Process mining is related to the field of synthesis (in particular language-based regions). However, the assumptions and goals are very different. Whereas classical synthesis approaches aim to obtain a "higher-level" process model that compactly describes the behavior of a "lower-level" behavioral specification, process mining techniques face a more difficult problem. The event logs used as input for process discovery typically contain only a fraction of the possible behavior. Traces in an event log can be seen as examples. If there are loops, one cannot expect to see all possible traces. If a model contains concurrency, one cannot expect to see all possible interleavings. If the model has multiple choices, one cannot expect to witness all possible combinations. There have been many attempts to extend region-based approaches to this setting [20, 11, 21, 22]. Unfortunately, region-based techniques are often computationally intractable, lead to overfitting models, and/or cannot discover process constructs such as skipping and mixtures of choice and synchronization (e.g., OR-joins). Hence, several more scalable and robust techniques have been developed. Commercial tools typically still resort to learning the so-called *Directly Follows Graph* (DFG) which typically leads to underfitting process models [23]. When activities appear out of sequence, loops are created, thus leading to Spaghetti-like diagrams suggesting repetitions that are not supported by the data. The inductive mining techniques [24, 25] and the so-called split miner [26] are examples of the state-of-the-art techniques to learn process models. These techniques are able to generalize and uncover concurrency.

This paper focuses on process discovery. However, rather than presenting a new discovery technique for traditional event logs, we start from *object-centric event logs* [27]. In a traditional event log each event is related to one activity, one timestamp, and one case (i.e., a process instance). We still make the assumption that each event refers to an activity and a point in time. However, we do *not* assume the existence of a single case notion. *Instead, an event may refer to any number of objects and these objects may be of different types.* This extends the reach of process mining dramatically. The step is comparable to going from Place Transition (P/T) nets to colored Petri nets. *Objects can be viewed as colored tokens and object types can be seen as color sets (i.e., place types).*

Based on object-centric event logs, we aim to automatically discover object-centric Petri nets. Such Petri nets have typed places that refer to the object types in the event log. Just like in colored



Each row corresponds to an event which refers to one activity and any number of objects of (possibly many) different types.

1	activity	time	orders	items	packages	customers	products	price	weight
13501	place order	2019-12-24 17:27:18	{991269}	{884994,884995,884996,884997,884998}	{}	{Wil van der Aalst}	{Fire Stick 4K,Echo Show 5,Echo Plus,iPad mini,Echo Show 8}	€ 913,96	3,70
13502	pick item	2019-12-24 17:33:55	{991261}	{884956}	{}	{Christine Dobbert}	{Kindle Paperwhite}	€ 129,00	0,50
13503	pick item	2019-12-24 17:50:11	{991269}	{884994}	{}	{Wil van der Aalst}	{Fire Stick 4K}	[illegible]	0,28
13504	pick item	2019-12-24 17:58:38	{991252}	{884918}	{}	{Kefang Ding}	{Fire Stick 4K}	[illegible]	0,28
13505	pay order	2019-12-24 18:05:49	{991236}	{884857,884858,884859,884860}	{}	{Mahsa Bafrani}	{Echo Studio,Kindle,M	[illegible]	4583,00
13506	pick item	2019-12-24 18:18:30	{991237}	{884864}	{}	{Mahnaz Qafari}	{Kindle Paperwhite}	[illegible]	0,50
13507	pick item	2019-12-24 18:19:53	{991182}	{884623}	{}	{Christine Dobbert}	{Echo Show 8}	€ 129,99	0,98
13508	pick item	2019-12-24 18:33:59	{991258}	{884940}	{}	{Tobias Brockhoff}	{iPad mini}	€ 449,00	0,28
13509	pay order	2019-12-24 18:47:07	{991261}	{884956,884957,884958,884959,884960,884961,884962}	{}	{Christine Dobbert}	{Kindle Paperwhite,iPad,Kindle,Echo Show 5,Echo Show 5,Echo Studio,Ki	€ 1.168,95	5184,00
13510	place order	2019-12-24 19:14:23	{991270}	{884999}	{}	{Junxiong Gao}	{iPad mini}	€ 454,00	0,28
13511	item out of stock	2019-12-24 20:39:01	{991224}	{884803}	{}	{Wil van der Aalst}	{iPhone 8}	€ 529,00	0,21
13512	package delivered	2019-12-24 20:46:47	{991119,991030,991209,991254,991213,9	{884386,884020,884749,884930,884926,884925,884766,88492	{660784}	{Kefang Ding}	{iPad Air,Echo Dot,MacBook Pro,iPad Air,Kindle Paperwhite,iPad Air,iPad	€ 6.829,99	4719,00
13513	place order	2019-12-24 22:08:21	{991271}	{885000,885001,885002,885003}	{}	{Mohammadreza Fani Sani}	{Fire Stick 4K,iPad,Kindle Paperwhite,iPad Air}	€ 1.194,99	1698,00
13514	pick item	[illegible]	[illegible]	[illegible]	{}	{Tobias Brockhoff}	{Fire Stick}	€ 39,99	0,20
13515	place order	[illegible]	[illegible]	5005}	{}	{Mohammadreza Fani Sani}	{iPhone 11 Pro,iPad Pro}	€ 2.253,00	0,67
13516	confirm orde	[illegible]	[illegible]	5005}	{}	{Mohammadreza Fani Sani}	{iPhone 11 Pro,iPad Pro}	€ 2.253,00	0,67
13517	pay order	[illegible]	[illegible]	4891,884892}	{}	{Junxiong Gao}	{Kindle,Fire Stick 4K,Echo Plus}	€ 324,97	2043,00
13518	package deli	[illegible]	[illegible]	429,884173,884810,884859}	{660788}	{Mahsa Bafrani}	{Kindle,Echo Show 8,iPad Pro,Echo,MacBook Pro}	€ 3.908,97	4096,00

One of the 21887 events:
- **activity**: package delivered
- **time**: 2019-12-24 20:46:47
- **orders involved**: {991119,991030,991209,991254, 991213,991206}
- **items involved**: {884386,884020,884749,884930, 884926,884925,884766,884927,884736}
- **packages involved**: {660784}
- **customers involved**: {Kefang Ding}
- **products involved**: {iPad Air,Echo Dot,MacBook Pro,iPad Air,Kindle Paperwhite,iPad Air,iPad Pro, iPad,iPhone 11 Pro}
- **total price**: € 6.829,99
- **total weight**: 4,719 KG

Figure 1. Overview of the approach presented in this paper. Object-centric event logs are used as an intermediate format in between the actual data sources and traditional event logs requiring a single case notion. Using this input, we discover object-centric Petri nets that are able to describe multiple object types in a single model.

Petri nets, a transition may consume or produce *multiple* tokens from a place during one execution. In this paper, we present the first technique to discover such nets. In the related work section (Section 9), we elaborate on the relation to earlier approaches such as the Object-Centric Behavioral Constraint (OCBC) models [28], synchronized transitions systems [29, 30], and artifact-centric discovery approaches [31, 32, 33].

Figure 1 illustrates the approach presented. Object-centric event logs can be extracted from any information system [27]. These logs can be seen as an intermediate format closer to the actual data collected by today's information systems. Unlike traditional event logs (e.g., XES logs), an event may refer to multiple objects and is not forced to be assigned to a single case. Enterprise Information

Systems (EIS), Customer Relationship Management (CRM) systems, Healthcare Information Systems (HIS), E-Learning Systems, Production Systems, Supply Chain Systems, etc. typically store information on a range of objects (customer, orders, patients, products, payments, etc.) in multiple tables that refer to each other. Figure 1 shows the objects related to one "package delivered" event. The event refers to six orders, nine items, one package, one customer, and nine products. In total, there are 22,367 events. Figure 1 shows an object-centric Petri net (not intended to be readable) discovered while focusing on orders, items, and packages. The places and arcs are typed. The colors red, green, and purple refer to respectively orders, items, and packages.

Just like for traditional process mining approaches it is possible to filter and seamlessly simplify the process model. By focusing on a particular object type, it is also possible to create traditional events logs that can be analyzed using traditional process mining techniques.

The remainder of this paper is organized as follows. Section 2 introduces event logs and process models. Object-centric event logs are introduced and motivated in Section 3. Given such logs, we first discuss techniques to learn process models for a single object type in Section 4. In Section 5, we introduce object-centric Petri nets, i.e., Petri nets with places referring to object types. Section 6 presents the main contribution of this paper: An approach to learn object-centric Petri nets from object-centric event logs. The discovery technique has been implemented in *PM4Py*, an open-source process mining platform written in Python. Section 7 presents the implementation and Section 8 demonstrates the feasibility of the approach. Related work is discussed in Section 9. Section 10 concludes the paper with a few final remarks.

2. Preliminaries

First, we introduce some preliminaries for people not familiar with process mining and accepting Petri nets. Input for process mining is an event log. A *traditional* event log views a process from a particular angle provided by the *case notion* that is used to *correlate events*. Each event in such an event 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. With some effort, such data can be extracted from any information system supporting operational processes. Process mining uses these event data to answer a variety of process-related questions. Process mining techniques such as process discovery, conformance checking, model enhancement, and operational support can be used to improve performance and compliance [34].

Each event in an event log has three *mandatory* attributes: case, activity, and timestamp. The case notion is used to group events, e.g., all events corresponding to the same order number are taken together. The timestamps are used to order the events and can be used to analyze bottlenecks, delays, etc. There may be many additional attributes, e.g., costs, resource, location, etc. However, most process discovery techniques first learn a model where only the order of activities within cases matters. Once the control-flow is clear, other attributes (e.g., time) can be added by replaying the event log on the model [34]. Therefore, we define a so-called "simple event log" that only records the ordering of activities for each case. Technically, an event log is a multiset of traces. $B \in \mathcal{B}(X) = X \to \mathbb{N}$ is a multiset over X where element $x \in X$ appears $B(x)$ times. For example, in $B = [a^5, b^2, c]$, a appears $B(a) = 5$ times, b twice, and c once.

Definition 2.1. (Simple Event Log)
Let $\mathbb{U}_{act}$ be the universe of activity names. A trace $\sigma \in \mathbb{U}^*_{act}$ is a sequence of activities. $L \in \mathcal{B}(\mathbb{U}^*_{act})$ is an event log, i.e., a multiset of traces. $\mathbb{U}_{SEL} = \mathcal{B}(\mathbb{U}^*_{act})$ is the universe of simple event logs.

For example, $\mathbb{U}_{act} = \{po, pi, sh, in, sr, pa, co, \ldots\}$ where po denotes activity *place order*, pi denotes activity *pick item*, sh denotes activity *ship item*, in denotes activity *send invoice*, sr denotes activity *send reminder*, pa denotes activity *pay order*, and co denotes activity *mark as completed*. Using this more compact notation we show three example traces: $\sigma_1 = \langle po, in, pi, sr, sh, pa, co\rangle$, $\sigma_2 = \langle po, pi, sh, in, pa, co\rangle$, and $\sigma_3 = \langle po, in, sr, sr, pi, sr, pa, sh, co\rangle$. Obviously, multiple cases can have the same trace. In an event log $L = [{\sigma_1}^{435}, {\sigma_2}^{366}, {\sigma_3}^{233}, \ldots]$ the above three traces appear respectively 435, 366, and 233 times. Given such an event log, process discovery techniques are able to learn a process model describing the observed traces. Such techniques often take into account frequencies, e.g., the model should cover the most frequent traces but may decide to abstract from infrequent ones. Figure 2 shows a process model discovered for event log L.

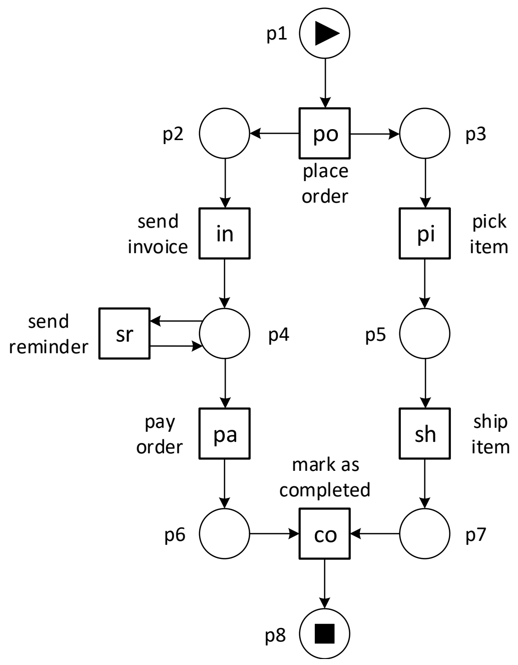


Figure 2. An accepting Petri net composed of eight places and seven transitions.

The discovered process model in Figure 2 is represented as an *accepting* Petri net where the transitions are *labeled*. We assume that the reader is familiar with standard Petri nets notations, but provide a few definitions to make the key notions explicit. We use Petri nets with a *labeling function* and *final marking*. This is driven by requirements from process mining. The labeling function is needed to model skips and duplicate activities. The final marking is needed because traces have a defined start and end.

Definition 2.2. (Labeled Petri Net)
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 $l \in T \not\rightarrow \mathbb{U}_{act}$ a labeling function.

A Petri net defines a directed graph with nodes $P \cup T$ and edges F. The state of a Petri net, called *marking*, is a multiset of places ($M \in \mathcal{B}(P)$). A transition $t \in T$ is *enabled* in marking M of net N if each of its input places $\bullet t = \{p \in P \mid (p, t) \in F\}$ contains at least one token. An enabled transition t may *fire*, i.e., one token is removed from each of the input places $\bullet t$ and one token is produced for each of the output places $t\bullet = \{p \in P \mid (t, p) \in F\}$. Assume that $[p1]$ is the initial marking of the Petri net in Figure 2. There are 11 markings reachable from this initial marking, including $[p1]$, $[p2, p3]$, $[p4, p7]$, and $[p8]$.

Note that the labeling function l may be partial and non-injective. This means that multiple transitions may refer to the same activity and that there may be transitions that are "silent" and do not correspond to an activity. Any firing sequence of a labeled Petri net corresponds to a visible trace obtained by mapping transitions onto activities using l. Firing an unlabeled transition does not add an activity to the trace. In Figure 2, all transitions are visible and unique. σ_1, σ_2, and σ_3 are examples of visible traces (assuming the short names as activity labels).

For process mining, we often focus on so-called *accepting Petri nets* that have an initial marking and a final marking. The reason is that we want to have a model that defines a language corresponding to the process that was used to produce the event log.

Definition 2.3. (Accepting Petri Net)
An accepting Petri net is a triplet $SN = (N, M_{init}, M_{final})$ 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. $\mathbb{U}_{APN}$ is the universe of accepting Petri nets.

In Figure 2, the initial marking $M_{init} = [p1]$ and the final marking $M_{final} = [p8]$ are denoted using the start and stop symbol.

Definition 2.4. (Language of an Accepting Petri Net)
An accepting Petri net $SN = (N, M_{init}, M_{final})$ defines a language $\phi(SN)$ that is composed of all *visible* traces (ignoring transition occurrences not having a label) starting in M_{init} and ending in M_{final}.

The accepting Petri net depicted in Figure 2 has infinitely many visible traces due to the loop involving sr. Without the loop, there would be six possible visible traces.

Assuming the basic setting with simple event logs and accepting Petri nets, we can now formally define the notion of *process discovery*. For any event log, we would like to construct a corresponding process model.

Definition 2.5. (Process Discovery Technique)
Discovery technique $disc$ is a function mapping simple event logs onto accepting Petri nets, i.e., $disc \in \mathbb{U}_{SEL} \rightarrow \mathbb{U}_{APN}$.

What makes process mining very difficult is that the event log only contains example behaviors. If Figure 2 represents the real process, we have the problem that no event log will contain all of its traces (due to the loop). Even when there are no loops, it is very unlikely to observe all possible traces for real-life processes due to combinations of choices and the interleaving of concurrent activities. Typically, only a fraction of the possible process is observed. Moreover, the event log may contain noise and infrequent behaviors that should not end up in the process model. This leads to notions such as recall (also called fitness), precision, generalization, and simplicity [34]. These are outside of the scope of this paper. However, we abstractly define the notion of *conformance checking*.

Definition 2.6. (Conformance Checking Technique)
Conformance checking technique *conf* is a function mapping a pair composed of an event log and an accepting Petri nets onto conformance diagnostics, i.e., $\mathit{conf} \in (\mathbb{U}_{SEL} \times \mathbb{U}_{APN}) \to \mathbb{U}_{diag}$.

$\mathit{conf}(L, SN) \in \mathbb{U}_{diag}$ provides diagnostics related to recall, precision, generalization, simplicity, etc. An example would be the fraction of traces in the event log that can be replayed by the accepting Petri net: $\mathit{conf}(L, SN) = |[\sigma \in L \mid \sigma \in \phi(SN)]| \,/\, |L|$. Given $L' = [\langle po, pi, sh, in, pa, co\rangle^8, \langle po, sh, pi, in, pa, co\rangle^2]$ and SN shown in Figure 2, $\mathit{conf}(L', SN) = 0.8$ given this conformance notion. Many other measures and diagnostics are possible. However, we leave $\mathbb{U}_{diag}$ deliberately vague.

3. Object-centric event logs

Section 2 provided a basic introduction to process mining, assuming that there is a clear case notion. In this section, we show that, for many applications, this assumption is not realistic (Section 3.1). Next, we formalize the notion of *object-centric event logs* (Section 3.2).

3.1. What if there is not a single case identifier?

In many applications, there are multiple candidate case notions leading to different views on the same process [27]. 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 or inconsistent.

To introduce the problem, consider the event log shown in Table 1. The table shows that each order may correspond to multiple items that are picked and shipped separately. This is a more realistic assumption (shops tend to allow customers to buy more than one product per order).

Table 1 has a column for order identifiers and item identifiers. Order 99001 corresponds to three items (88124, 88125, and 88126), order 99002 corresponds to two items (88127 and 88128), order 99003 corresponds to one item (88129), and order 99004 corresponds to five items (88130, 88131, 88132, 88133, and 88134). The pick and ship activities are executed for individual items. An order is marked as completed when all items have been picked and shipped and the order itself was paid. Note that the events *place order* and *mark as completed* for order 99001, both refer to four objects (one order and three items). The latter number is variable. For example, the event *place order* for order 99003 refers to only two objects. This cannot be expressed using the accepting Petri nets introduced before. Transitions need to consume and produce a variable number of tokens of different types. Therefore, we propose to use *object-centric Petri nets*. Note that we do *not* propose such nets as a new *modeling*

Table 1. A fragment of an event log: Each line corresponds to an event, possibly referring to multiple objects (i.e., orders and items).

activity	timestamp	order	item
...	...	...	...
place order	25-11-2019:09.35	$\{99001\}$	$\{88124, 88125, 88126\}$
pick item	25-11-2019:10.35	$\emptyset$	$\{88126\}$
place order	25-11-2019:11.35	$\{99002\}$	$\{88127, 88128\}$
pick item	26-11-2019:010.25	$\emptyset$	$\{88124\}$
send invoice	27-11-2019:08.12	$\{99001\}$	$\emptyset$
send invoice	28-11-2019:09.35	$\{99002\}$	$\emptyset$
pick item	29-11-2019:09.35	$\emptyset$	$\{88127\}$
send reminder	29-11-2019:10.35	$\{99002\}$	$\emptyset$
pick item	29-11-2019:11.15	$\emptyset$	$\{88128\}$
ship item	29-11-2019:12.35	$\emptyset$	$\{88124\}$
pick item	29-11-2019:13.30	$\emptyset$	$\{88125\}$
send reminder	29-11-2019:14.35	$\{99001\}$	$\emptyset$
ship item	29-11-2019:15.15	$\emptyset$	$\{88125\}$
send reminder	29-11-2019:16.15	$\{99002\}$	$\emptyset$
ship item	29-11-2019:17.45	$\emptyset$	$\{88126\}$
ship item	29-11-2019:18.00	$\emptyset$	$\{88128\}$
send reminder	30-11-2019:09.35	$\{99002\}$	$\emptyset$
ship item	30-11-2019:10.05	$\emptyset$	$\{88127\}$
pay order	30-11-2019:11.45	$\{99002\}$	$\emptyset$
pay order	30-11-2019:12.55	$\{99001\}$	$\emptyset$
mark as completed	01-12-2019:09.35	$\{99001\}$	$\{88124, 88125, 88126\}$
place order	02-12-2019:10.40	$\{99003\}$	$\{88129\}$
mark as completed	04-12-2019:11.05	$\{99002\}$	$\{88127, 88128\}$
place order	06-12-2019:14.18	$\{99004\}$	$\{88130, 88131, 88132, 88133, 88134\}$
...	...	...	...

language. It can be viewed as a subclass of colored Petri nets, but our focus is on learning a model describing the data in Table 1. Hence, we limit the modeling notation to what can be discovered for such data.

Figure 3 shows the object-centric Petri net we want to discover based on the event data in Table 1. There are now two types of places: the places that correspond to orders (colored green) and the places that correspond to items (colored blue). Transitions are colored based on the object types they refer to. Note that transitions po and co have two colors. A transition may consume multiple tokens from a place or produce multiple tokens for a place. The places and arcs involved in events that consume or produce multiple objects have compound double arrows to highlight this. Transition po in Figure 3 consumes one order object from place $o1$ and a variable number of items from place $i1$. po produces one order object for place $o2$ and a variable number of items for place $i2$. Transition pi in Figure 3 consumes one item object from place $i2$ and produces one item object for place $i3$. The items are also shipped individually. However, transition co in Figure 3 consumes one order object from place $o4$ and all items corresponding to the order from place $i4$.

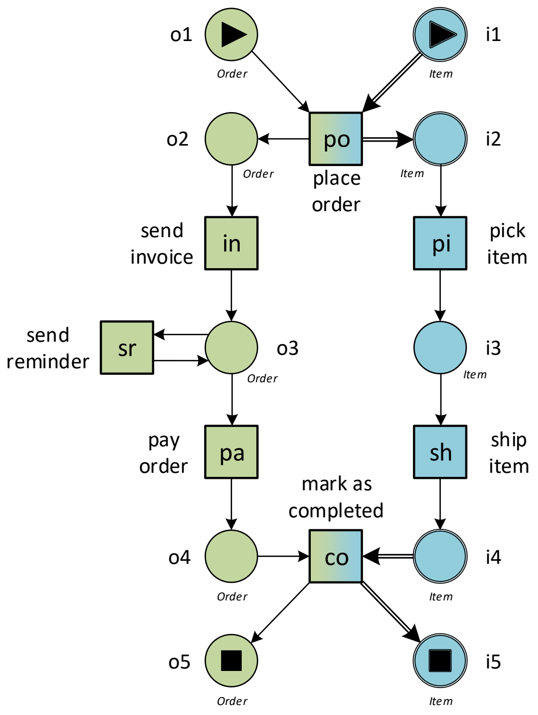


Figure 3. An object-centric Petri nets with two object types: *Order* and *Item*.

Although existing discovery techniques cannot handle the event data in Table 1, this is still a relatively simple scenario since there is a one-to-many relationship between orders and items. In real-life applications, there may also be many-to-many relationships. To illustrate this, consider the event

Table 2. A small fragment of a simple event log with three types of objects.

activity	timestamp	order	item	route
...	...	...	...	...
place order	25-11-2019:09.35	$\{99001\}$	$\{88124, 88125, 88126\}$	$\emptyset$
place order	25-11-2019:11.35	$\{99002\}$	$\{88127, 88128\}$	$\emptyset$
...	...	...	...	...
start route	25-11-2019:11.35	$\emptyset$	$\{88124, 88127\}$	$\{66222\}$
end route	25-11-2019:11.35	$\emptyset$	$\{88124, 88127\}$	$\{66222\}$
...	...	...	...	...
start route	25-11-2019:11.35	$\emptyset$	$\{88125, 88126, 88128\}$	$\{66223\}$
end route	25-11-2019:11.35	$\emptyset$	$\{88125, 88126, 88128\}$	$\{66223\}$
...	...	...	...	...
mark as completed	01-12-2019:09.35	$\{99001\}$	$\{88124, 88125, 88126\}$	$\emptyset$
mark as completed	04-12-2019:11.05	$\{99002\}$	$\{88127, 88128\}$	$\emptyset$
...	...	...	...	...

log fragment depicted in Table 2 where we added routes. On any particular route, multiple items can be delivered. The *ship item* activity is now replaced by the *start route* and *end route* activities that may refer to items from different orders. As shown in Table 2, route 66222 refers to two items (88124 and 88127) belonging to orders 99001 and 99002. Route 66223 refers to three items (88125, 88126, and 88128) belonging to orders 99001 and 99002.

Again it is obvious that this cannot be modeled using traditional process models that assume a single case notion.

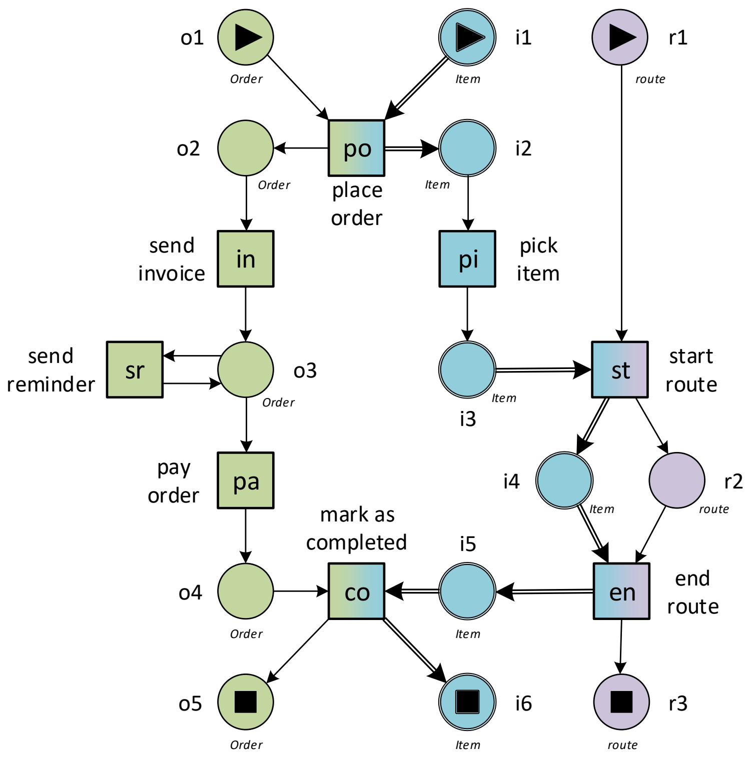


Figure 4. An object-centric Petri nets with three object types: *Order*, *Item*, and *Route*.

Figure 4 shows the object-centric Petri net discovered from the event log referred to by Table 2. There are now three types of places: *order* places (colored green), *item* places (colored blue), and *route* places (purple). Transition st in Figure 4 consumes a variable number of item objects from place $i3$ and one route object from place $r1$. st produces a variable number of item objects for place $i4$ and one route object for place $r2$. The coloring of the transitions and places and the two different types of arcs show the behaviors observed in the event log.

The problem is that existing process mining techniques assume a “flattened event log” where each event refers to precisely one case. However, we would like to see process models such as the one depicted in Figure 4. One quickly encounters the problems described in this section when applying process mining to ERP systems from SAP, Oracle, Microsoft, and other vendors of enterprise software.

3.2. Formalizing object-centric event logs

Tables 1 and 2 illustrate the type of data we use as input for discovery. Such data are in-between the real data in information systems (e.g., multiple tables in a relational database) and the traditional event data stored in the *eXtensible Event Stream* (XES) format [35]. Whereas XES requires one case identifier per event, our format supports any number of objects of different types per event. To define our *object-centric event logs*, we first define several universes used in the remainder (based on [27]).

Definition 3.1. (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 (also used to label transitions in an accepting Petri net),
- $\mathbb{U}_{time}$ 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),
- $type \in \mathbb{U}_{oi} \rightarrow \mathbb{U}_{ot}$ assigns precisely one type to each object identifier,
- $\mathbb{U}_{omap} = \{omap \in \mathbb{U}_{ot} \not\rightarrow \mathcal{P}(\mathbb{U}_{oi}) \mid \forall_{ot \in dom(omap)} \forall_{oi \in omap(ot)} \; type(oi) = ot\}$ is the universe of all object mappings indicating which object identifiers are included per type,[1]
- $\mathbb{U}_{att}$ 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,[2] 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.

An event $e = (ei, act, time, omap, vmap) \in \mathbb{U}_{event}$ is characterized by a unique event identifier ei, the corresponding activity act, the event's timestamp $time$, and two mappings $omap$ and $vmap$ for respectively object references and attribute values.

Definition 3.2. (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 $\pi_{vmap}(e) = vmap$.

$\pi_{omap}(e) \in \mathbb{U}_{ot} \not\rightarrow \mathcal{P}(\mathbb{U}_{oi})$ maps a subset of object types onto sets of object identifiers for an event e. Consider for example the first visible event in Table 2 and assume this is e. $\pi_{omap}(e)(Order) = \{99001\}$, $\pi_{omap}(e)(Item) = \{88124, 88125, 88126\}$, and $\pi_{omap}(e)(Route) = \emptyset$. Moreover, $\pi_{act}(e) =$ *place order* and $\pi_{time}(e) =$ 25-11-2019:09.35. $dom(\pi_{vmap}(e)) = \emptyset$ since no attribute values are mentioned in Table 2. If the event would have a cost of 30 euros and location Aachen, then $\pi_{vmap}(e)(cost) = 30$ and $\pi_{vmap}(e)(location) = Aachen$.

[1] $\mathcal{P}(\mathbb{U}_{oi})$ is the powerset of the universe of object identifiers, i.e., objects types are mapped onto sets of object identifiers. $omap \in \mathbb{U}_{ot} \not\rightarrow \mathcal{P}(\mathbb{U}_{oi})$ is a partial function. If $ot \notin dom(omap)$, then we assume that $omap(ot) = \emptyset$.
[2] $\mathbb{U}_{att} \not\rightarrow \mathbb{U}_{val}$ is the set of all partial functions mapping a subset of attribute names onto the corresponding values.

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

Definition 3.3. (Object-Centric Event Log)
$L = (E, \preceq_E)$ 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
- $\forall_{e_1,e_2 \in E}\ e_1 \preceq_E e_2 \ \Rightarrow\ \pi_{time}(e_1) \leq \pi_{time}(e_2)$.

Definition 3.3 allows for partially ordered event logs. However, in practice, we often use a total order, e.g., events are ordered based on timestamps and when two events have the same timestamp we assume some order. In the tabular format used before (e.g., Table 2) we were also forced to use a total order. However, there are process discovery techniques that take into account causalities [27, 36]. These can exploit such partial orders.

4. Discovering Petri nets for a single object type

Object-centric event logs generalize the traditional event log notion where each event has precisely one case identifier. We can mimic such logs using a special object type $case \in \mathbb{U}_{ot}$ such that $|\pi_{omap}(e)(case)| = 1$ for any event $e \in E$. Since traditional process mining techniques assume this, it is common practice to convert event data with events referring to a variable number of objects to classical event logs by "flattening" the event data. Assume that we take a specific object type as a case identifier. If an event has multiple objects of that type, then we can simply create one event for each object. If an event has no objects of that type, then we simply omit the event. If an event has precisely one object of the selected type, then we keep that event. This can be formalized as follows.

Definition 4.1. (Flattening Event Logs)
Let $L = (E, \preceq_E)$ be an object-centric event log and $ot \in \mathbb{U}_{ot}$ an object type serving as a case notion. The flattened event log is $L^{ot} = (E^{ot}, \preceq_E^{ot})$ with:[3]

- $e_i = ((\pi_{ei}(e), i), \pi_{act}(e), \pi_{time}(e), \pi_{omap}(e) \oplus (case, \{i\}), \pi_{vmap}(e))$ for any $e \in E$ and $i \in \pi_{omap}(e)(ot)$,
- $E^{ot} = \{e_i \mid e \in E \ \wedge\ i \in \pi_{omap}(e)(ot)\}$, and
- $\preceq_E^{ot} = \{(e'_i, e''_j) \in E^{ot} \times E^{ot} \mid e' \in E \wedge i \in \pi_{omap}(e')(ot) \ \wedge\ e'' \in E \wedge j \in \pi_{omap}(e'')(ot) \ \wedge\ e' \preceq_E e'' \ \wedge\ (e' = e'' \Rightarrow i = j)\}$.

A flattened event log is still an event log after removing and duplicating events.

[3] $f' = f \oplus (x, y)$ is a function such that $dom(f') = dom(f) \cup \{x\}$, $f'(x) = y$ and $f'(z) = f(z)$ for $z \in dom(f) \setminus \{x\}$.

Lemma 4.2. Let $L = (E, \preceq_E)$ be an object-centric event log and $ot \in \mathbb{U}_{ot}$ an object type serving as a case notion. The flattened event log $L^{ot} = (E^{ot}, \preceq_E^{ot})$ is indeed an event log as defined in Definition 3.3.

Proof:
$\preceq_E^{ot}$ defines a partial order. For any $e_i \in E^{ot}$, $e_i \preceq_E^{ot} e_i$ (reflexive). If $e'_i \preceq_E^{ot} e''_j$ and $e''_j \preceq_E^{ot} e'_i$, then $e' = e''$ and $i = j$, and hence also $e'_i = e''_j$ (antisymmetric). If $e'_i \preceq_E^{ot} e''_j$ and $e''_j \preceq_E^{ot} e'''_k$, then $e' \preceq_E e''$, $e'' \preceq_E e'''$, $(e' = e'' \Rightarrow i = j)$, and $(e'' = e''' \Rightarrow j = k)$. Hence, $e' \preceq_E e'''$ ($\preceq_E$ is transitive). If $e' \neq e'''$, then $e'_i \preceq_E^{ot} e'''_k$ due to the definition of $\preceq_E^{ot}$. If $e' = e'''$, then $e' = e''$ and $e'' = e'''$. Hence, $i = j$ and $j = k$ (see above), and $i = k$. Again we conclude that $e'_i \preceq_E^{ot} e'''_k$ (transitive). If $\pi_{ei}(e'_i) = \pi_{ei}(e''_j)$, then $(e', i) = (e'', j)$ making event identifiers unique. If $e'_i \preceq_E^{ot} e''_j$, then $e' \preceq_E e''$. Hence, $\pi_{time}(e'_i) = \pi_{time}(e') \leq \pi_{time}(e'') = \pi_{time}(e''_j)$ showing that time cannot go backwards. □

Table 2 shows eight events (the rest is omitted). Assume $L = (E, \preceq_E)$ is the log consisting of only these eight events. The flattened event log $L^{Order} = (E^{Order}, \preceq_E^{Order})$ has four events (the four middle events in Table 2 are removed). The flattened event log $L^{Item} = (E^{Item}, \preceq_E^{Item})$ has 20 events since all original events are replicated two or three times. The flattened event log $L^{Route} = (E^{Route}, \preceq_E^{Route})$ has four events.

Assume now that $L = (E, \preceq_E)$ is flattened using object type ot leading to event log $L^{ot} = (E^{ot}, \preceq_E^{ot})$. We then have a conventional event log with a selected case notion and can apply all existing process mining techniques. However, flattening the event log using ot as a case notion potentially leads to the following problems.

- *Deficiency*: Events in the original event log that have *no* corresponding events in the flattened event log disappear from the data set (i.e., $\pi_{omap}(e)(ot) = \emptyset$).
- *Convergence*: Events referring to *multiple* objects of the selected type are replicated, possibly leading to unintentional duplication (i.e., $|\pi_{omap}(e)(ot)| \geq 2$).
- *Divergence*: Events referring to *different* objects of a type *not* selected as the case notion are considered to be causally related. For example, two events refer to the same order but different times or two events refer to the same route but different items.

Definition 4.3. (Deficiency, Convergence, and Divergence)
Let $L = (E, \preceq_E)$ be an object-centric event log and $L^{ot} = (E^{ot}, \preceq_E^{ot})$ the flattened event log based on object type $ot \in \mathbb{U}_{ot}$. Event $e \in E$ has a deficiency problem if $\pi_{omap}(e)(ot) = \emptyset$ (i.e., the event is ignored when using ot as case notion). Event $e \in E$ has a convergence problem if $|\pi_{omap}(e)(ot)| \geq 2$ (i.e., the event is unintentionally replicated when using ot as case notion). Event $e \in E$ has a divergence problem if there exist another event $e' \in E$ and object type $ot' \in \mathbb{U}_{ot}$ such that $\pi_{omap}(e)(ot) \neq \emptyset$, $\pi_{omap}(e')(ot) \neq \emptyset$, $\pi_{omap}(e)(ot') \neq \emptyset$, $\pi_{omap}(e')(ot') \neq \emptyset$, $\pi_{omap}(e)(ot) = \pi_{omap}(e')(ot)$, and $\pi_{omap}(e)(ot') \neq \pi_{omap}(e')(ot')$.

Note that in case of divergence, there are two events e and e' and two candidate case notions ot and ot' such that both events refer to objects of both object types and the events "agree" on ot but not on ot'.

Consider again the eight events shown in Table 2. When taking *Order* or *Route* as the object type used to flatten the event log, half the events disappear (deficiency). When taking *Item* as the object type used to flatten the event log, the first event is replaced by three *place order* events, the second event is replaced by two *place order* events, etc. This is misleading since these replicated events occurred only once (convergence). To explain divergence, assume that an order consists of 10 items and object type *Order* is used to flatten the event log. There will be 10 pick events that are executed in a given order. Although they are independent, they will seem to be causally related (same case) and most discovery algorithms will introduce a loop, although there is precisely one pick event per item.

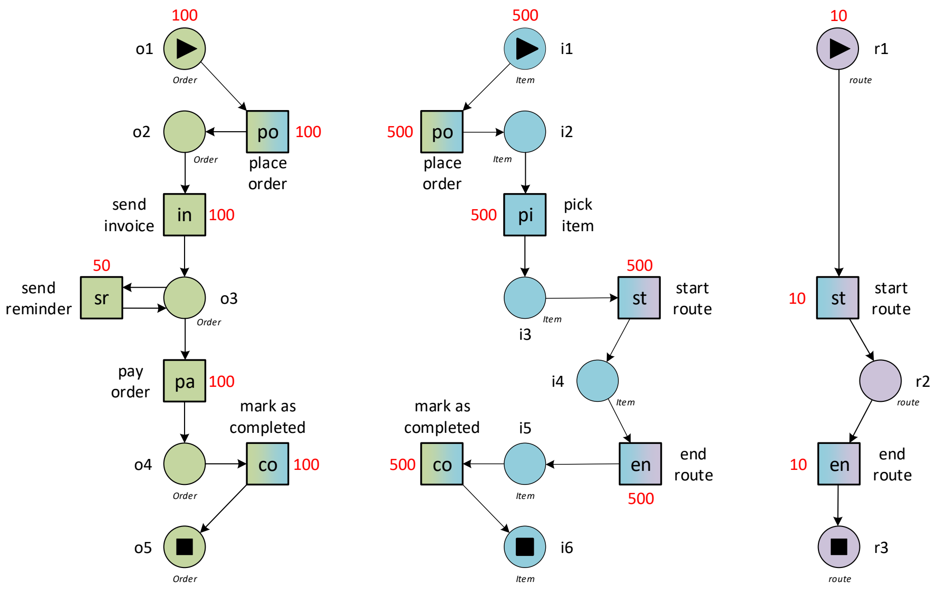


Figure 5. Three accepting Petri nets discovered for the three flattened event logs: $(E^{Order}, \preceq_E^{Order})$ (left), $(E^{Item}, \preceq_E^{Item})$ (middle), and $(E^{Route}, \preceq_E^{Route})$ (right). The numbers in red refer to the total number of tokens produced or consumed per arc.

Figure 5 shows three process models discovered for three flattened event logs: $L^{Order} = (E^{Order}, \preceq_E^{Order})$, $L^{Item} = (E^{Item}, \preceq_E^{Item})$, and $L^{Route} = (E^{Route}, \preceq_E^{Route})$. For example, the accepting Petri net in the middle was discovered based on L^{Item}, i.e., the original event log flattened using object type *Item*. Assume that there are 100 orders with on average 5 items per order. This implies that there are 500 items. Assume that each route consists, on average, of 50 items that need to be delivered, i.e., there are 10 routes in total. These numbers are depicted in Figure 5. Although the three accepting Petri nets look reasonable, they do not "fit" together (the frequencies of the corresponding activities are different). For example, in the left model (order) the *place order* activity is performed 100 times and in the middle model (item) the same activity is executed 500 times (factor 5). In the right model (route) the *start route* activity is performed 10 times and in the middle model (item) the same activity is executed 500 times (factor 50). These mismatches illustrate the convergence problem. One could argue that the accepting Petri net in the middle is wrong because the frequencies of activities do not match the frequencies in the original process model.

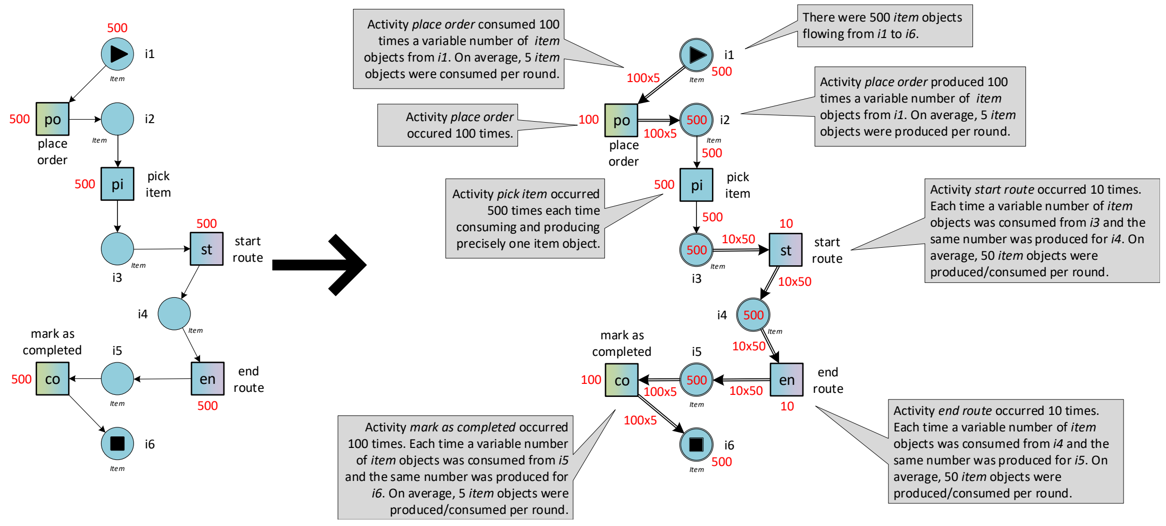

500
i1
500 po
place order
i2
500 pi
pick item
500 st
start route
i3
i4
mark as completed
i5
500 co
en
end route
500
i6
Activity *place order* consumed 100 times a variable number of *item* objects from *i1*. On average, 5 *item* objects were consumed per round.
Activity *place order* occured 100 times.
There were 500 *item* objects flowing from *i1* to *i6*.
Activity *place order* produced 100 times a variable number of *item* objects from *i1*. On average, 5 *item* objects were produced per round.
Activity *pick item* occurred 500 times each time consuming and producing precisely one item object.
Activity *start route* occurred 10 times. Each time a variable number of *item* objects was consumed from *i3* and the same number was produced for *i4*. On average, 50 *item* objects were produced/consumed per round.
Activity *mark as completed* occurred 100 times. Each time a variable number of *item* objects was consumed from *i5* and the same number was produced for *i6*. On average, 5 *item* objects were produced/consumed per round.
Activity *end route* occurred 10 times. Each time a variable number of *item* objects was consumed from *i4* and the same number was produced for *i5*. On average, 50 *item* objects were produced/consumed per round.
100x5
10x50

Figure 6. The model on the left was discovered for $L^{Item} = (E^{Item}, \preceq_E^{Item})$. Because of flattening, the frequencies of activities are not correct. However, it is known which transition occurrences belonged to each event and we can regroup them. This can be used to merge occurrences, leading to the process model on the right.

Figure 6 sketches how the problem of incorrect activity frequencies can be resolved using *variable arcs*, i.e., arcs that can be used to consume or produce multiple tokens in one step. Such "multiset arcs" are also possible in colored Petri nets [37, 38]. When an event was replicated to produce the "flat model" (e.g., Figure 5), we can merge the corresponding transition occurrences into one transition occurrence that may consume and produce multiple tokens. See for example transition *place order*. In the accepting Petri net on the left, transition *place order* fires 500 times when replaying the flattened event log L^{Item}. However, we know exactly which transition occurrences belong together. This can be used to reconstruct transition occurrences that consume and produce a variable number of tokens in one step. For transition *place order* this means that 500 occurrences are merged onto 100 occurrences that, on average, consume and produce 5 tokens per arc. To indicate this, we use compound double arrows with the annotation 100×5. Next, consider transition *start route*. In the model on the left, transition *start route* fires 500 times. However, we know exactly which of these 500 transition occurrences belong to the 10 routes.

Again these low-level transition occurrences can be merged into higher-level transition occurrences that consume and produce a variable number of tokens in one step. For transition *start route* this means that there are 10 occurrences that, on average, consumer and produce 50 tokens per arc. To indicate this, we use again compound double arrows, but now with the annotation 10×50. Only the occurrences of *pick item* did not change due to flattening. Hence, the corresponding arcs did not change.

Figure 6 sketches how we can create Petri nets for one object type where the frequency of each transition matches the actual number of corresponding events in the event log. These models can be merged into more holistic process models showing the different object types as is shown next.

5. Object-centric Petri nets

As indicated in the previous sections, we need to be able to distinguish the *different object types* and a single event (i.e., transition occurrence) may involve a *variable number of objects* (e.g., one order may have any number of items). An obvious way to model such processes is to use colored Petri nets where places can have different types [37, 38]. Figure 7 shows a screenshot of CPN Tools while simulating the scenario with 100 orders, 500 items, and 10 routes described before. The color sets *Order*, *Item*, and *Route* are used to type the places. The ten arcs with the annotation or produce or consume a single order. The two arcs with the annotation it produce or consume a single item. The four arcs with the annotation rt produce or consume a single route. There are eight arcs with the annotation its which is a variable of type *Items*, i.e., a list of items. These consume or produce a variable number of item objects. The four guards determine the correspondence between orders, routes, and items. For example, the guard $[its = oi(or)]$ of transition *place order* specifies the set of items its involved in a specific order or. The same guard is used for transition *mark as completed*. Transitions *start route* and *end route* use guard $[its = ri(rt)]$ to determine the items its involved in route rt.

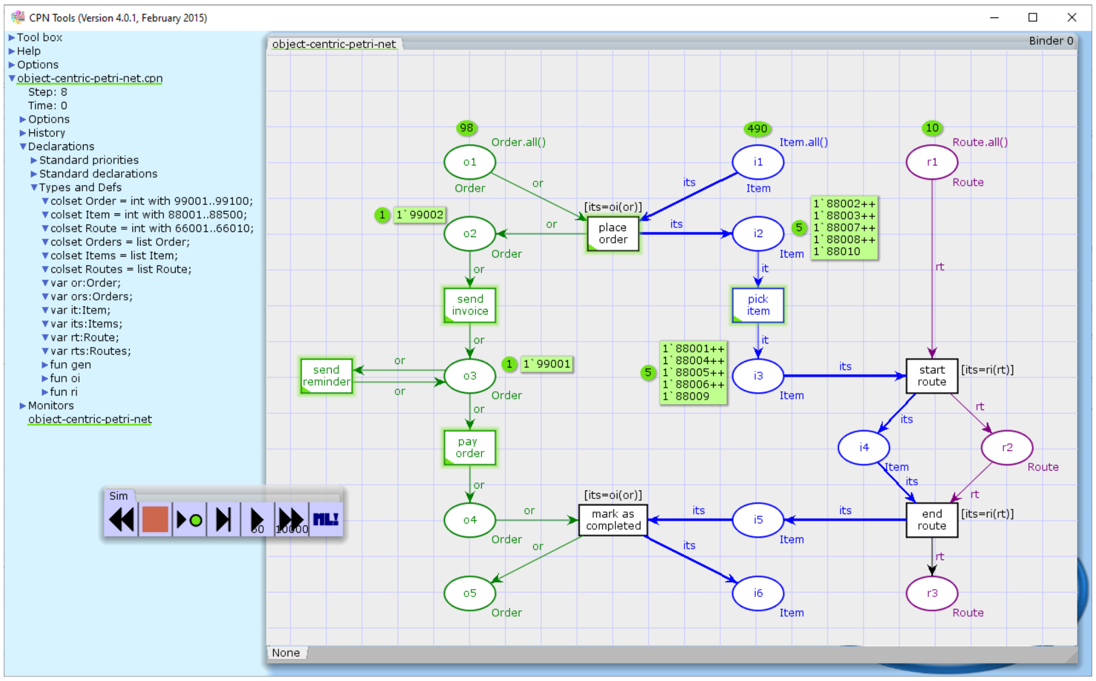

```
CPN Tools (Version 4.0.1, February 2015)
▶Tool box
▶Help
▶Options
▼object-centric-petri-net.cpn
   Step: 8
   Time: 0
  ▶Options
  ▶History
  ▼Declarations
    ▶Standard priorities
    ▶Standard declarations
    ▼Types and Defs
      ▼colset Order = int with 99001..99100;
      ▼colset Item = int with 88001..88500;
      ▼colset Route = int with 66001..66010;
      ▼colset Orders = list Order;
      ▼colset Items = list Item;
      ▼colset Routes = list Route;
      ▼var or:Order;
      ▼var ors:Orders;
      ▼var it:Item;
      ▼var its:Items;
      ▼var rt:Route;
      ▼var rts:Routes;
      ▶fun gen
      ▶fun oi
      ▶fun ri
  ▶Monitors
   object-centric-petri-net
```

Figure 7. A colored Petri net in CPN Tools [37, 38] modeling the process depicted in Figure 4 which was discovered from the event data in Table 2.

Figure 7 shows that one can model processes involving multiple objects using colored Petri nets (or related formalisms). However, it is infeasible to discover an *arbitrary* colored Petri net from an (object-centric) event log. We need a *representational bias* that corresponds to the information in the event log. Therefore, we aim to discover a specific type of colored Petri net. To simplify matters, we also abstract from the matching between the different objects (i.e., the guards in Figure 7). This allows us to use a more specific and more abstract representation called *object-centric Petri net*.[4]

Definition 5.1. (Object-Centric Petri Net)
An *object-centric Petri net* is a tuple $ON = (N, pt, F_{var})$ where $N = (P, T, F, l)$ is a labeled Petri net, $pt \in P \to \mathbb{U}_{ot}$ maps places onto object types, and $F_{var} \subseteq F$ is the subset of variable arcs.

Figure 4 shows an object-centric Petri net: $P = \{o1, \ldots, o5, i1, \ldots, i6, r1, r2, r3\}$, $T = \{po, in, pi, \ldots\}$, $F = \{(o1, po), (i1, po), (po, o2), (po, i2), \ldots\}$, $l(po) =$ *place order*, $l(in) =$ *send invoice*, etc., $pt(o1) = \mathit{Order}$, $pt(i1) = \mathit{Item}$, $pt(r1) = \mathit{Route}$, etc., and $F_{var} = \{(i1, po), (po, i2), \ldots\}$. Note that the graphical notation in Figure 4 fully defines the object-centric Petri net.

Definition 5.2. (Well-Formed)
Let $ON = (N, pt, F_{var})$ be an object-centric Petri net with $N = (P, T, F, l)$. We introduce the following notations:

- $pl(t) = \bullet t \cup t\bullet$ are the input and output places of $t \in T$, $pl_{var}(t) = \{p \in P \mid \{(p, t), (t, p)\} \cap F_{var} \neq \emptyset\}$ are the input and output places connected through variable arcs, and $pl_{nv}(t) = \{p \in P \mid \{(p, t), (t, p)\} \cap (F \setminus F_{var}) \neq \emptyset\}$ are the places connected through non-variable arcs.
- $tpl(t) = \{pt(p) \mid p \in pl(t)\}$, $tpl_{var}(t) = \{pt(p) \mid p \in pl_{var}(t)\}$, and $tpl_{nv}(t) = \{pt(p) \mid p \in pl_{nv}(t)\}$ are the corresponding place types.

ON is *well-formed* if for each transition $t \in T$: $tpl_{var}(t) \cap tpl_{nv}(t) = \emptyset$.

In a *well-formed* object-centric Petri net, the arcs should "agree" on variability, i.e., a combination of an object type and transition has variable arcs or normal arcs but not both. For example, because $(i1, po) \in F_{var}$ also $(po, i2) \in F_{var}$. Because $(o1, po) \notin F_{var}$ also $(po, o2) \notin F_{var}$. This assumption is reasonable when looking at an object-centric event log. Per event e and object type ot, $\pi_{omap}(e)(ot)$ is given. Therefore, it makes no sense to consider different sets of objects of the same type ot per transition t. In the remainder, we limit ourselves to well-formed object-centric Petri nets (without explicitly stating this).

A token denoted by (p, oi) resides in place p and refers to object oi. A marking is a multiset of such tokens. In the marking $[(p1, 666), (p2, 666), (p2, 555), (p3, 555)]$ there are four tokens (place $p2$ has two tokens referring to objects 555 and 666).

Definition 5.3. (Marking)
Let $ON = (N, pt, F_{var})$ be an object-centric Petri net with $N = (P, T, F, l)$. $Q_{ON} = \{(p, oi) \in P \times \mathbb{U}_{oi} \mid type(oi) = pt(p)\}$ is the set of possible tokens. A marking M of ON is a multiset of tokens, i.e., $M \in \mathcal{B}(Q_{ON})$.

To describe the semantics of an object-centric Petri net, we use the notion of *bindings*, similar to the notion of bindings in colored Petri nets. However, now the binding refers to the object references

[4]Terms similar to "object Petri nets" were already used by Rüdiger Valk, Charles Lakos, Jinzhong Niu, Li-Chi Wang, Daniel Moldt, and others. Note that our nets are different and some overloading of terminology is unavoidable.

of the corresponding event in the event log. A binding (t, b) refers to a transition t and a function b that maps a subset of object types to sets of object identifiers. The subset of object types corresponds to the object types of the surrounding places (i.e., $\mathit{tpl}(t)$). Moreover, for non-variable arcs the binding should select precisely one object (i.e., $|b(ot)| = 1$ for $ot \in \mathit{tpl}_{nv}(t)$). Consider transition t and one of its input place p (i.e., $p \in \bullet t$). If t fires with binding (t, b), then $pt(p) \in \mathit{dom}(b)$ and the objects $b(pt(p))$ are removed from input place p. If p is an output place of t ($p \in t\bullet$), then the objects $b(pt(p))$ are added to output place p. Therefore, binding (t, b) fully determines the new marking.

Definition 5.4. (Binding Execution)
Let $ON = (N, pt, F_{var})$ be an object-centric Petri net with $N = (P, T, F, l)$. $B = \{(t, b) \in T \times \mathbb{U}_{omap} \mid \mathit{dom}(b) = \mathit{tpl}(t) \ \wedge\ \forall_{ot \in \mathit{tpl}_{nv}(t)} \ |b(ot)| = 1\}$ is the set of all possible bindings. $(t, b) \in B$ is a binding and corresponds to the execution of transition t consuming selected objects from the input places and producing the corresponding objects for the output places (both specified by b). $\mathit{cons}(t, b) = [(p, oi) \in Q_{ON} \mid p \in \bullet t \ \wedge\ oi \in b(pt(p))]$ is the multiset of tokens to be consumed given binding (t, b). $\mathit{prod}(t, b) = [(p, oi) \in Q_{ON} \mid p \in t\bullet \ \wedge\ oi \in b(pt(p))]$ is the multiset of tokens to be produced given binding (t, b). Binding (t, b) is *enabled* in marking $M \in \mathcal{B}(Q_{ON})$ if $\mathit{cons}(t, b) \leq M$. The occurrence of an enabled binding (t, b) in marking M leads to the new marking $M' = M - \mathit{cons}(t, b) + \mathit{prod}(t, b)$.[5] This is denoted as $M \xrightarrow{(t,b)} M'$.

$M \xrightarrow{(t,b)} M'$ implies that binding (t, b) is enabled in marking M and that the occurrence of this binding leads to the new marking M'. It is also possible to have a sequence of enabled bindings $\sigma = \langle (t_1, b_1), (t_2, b_2), \ldots, (t_n, b_n)\rangle \in B^*$ such that $M_0 \xrightarrow{(t_1,b_1)} M_1 \xrightarrow{(t_2,b_2)} M_2 \xrightarrow{(t_3,b_3)} \ldots \xrightarrow{(t_n,b_n)} M_n$, i.e., it is possible to reach M_n from M_0 in n steps. This is denoted $M \xrightarrow{\sigma} M'$. It is also possible to map the transition names onto the corresponding activity names using l leading to the so-called *visible binding sequence* $\sigma_v = \langle (l(t_1), b_1), (l(t_2), b_2), \ldots, (l(t_n), b_n)\rangle$ (where $(l(t_i), b_i)$ is omitted if t_i has no label). Note that the visible binding sequence does not show silent steps (transitions with no label) and cannot distinguish duplicate activities (two transitions with the same label).

It should be noted that Definition 5.4 does not put any constraints on the binding other than that for non-variable arcs precisely one token is consumed/produced. In the colored Petri in Figure 7 there are four transitions with guards to link items to specific orders and routes. This is deliberately abstracted from in Definition 5.1 to enable the discovery of so-called *accepting object-centric Petri nets* with an initial and final marking.

Definition 5.5. (Accepting Object-Centric Petri Net)
An accepting object-centric Petri net is a tuple $AN = (ON, M_{\mathit{init}}, M_{\mathit{final}})$ composed of a well-formed object-centric Petri net $ON = (N, pt, F_{var})$, an initial marking $M_{\mathit{init}} \in \mathcal{B}(Q_{ON})$, and a final marking $M_{\mathit{final}} \in \mathcal{B}(Q_{ON})$.

Using the notion of a visible binding sequence, we can reason about all behaviors leading from the initial to the final marking.

[5]Summation ($+$), difference ($-$), and inclusion ($\leq$) are defined for multisets in the usual way, e.g., $[a, b] + [b, c] = [a, b^2, c]$, $[a, b^2, c] - [b, c] = [a, b]$, and $[a, b] \leq [a, b^2, c]$.

Definition 5.6. (Language of an Object-Centric Petri Net)
An accepting object-centric Petri net $AN = (ON, M_{init}, M_{final})$ defines a language $\phi(AN) = \{\sigma_v \mid M_{init} \xrightarrow{\sigma} M_{final}\}$ that is composed of all visible binding sequences starting in M_{init} and ending in M_{final}.

Note that the behavior of an accepting object-centric Petri net is deliberately "underspecified". There are only typing and cardinality constraints. Hence, objects of different types are unrelated. Compared to the colored Petri net in Figure 7, our process models do not use guards to relate objects of different types. Note that guards combine objects of different types that are only characterized by an identifier. Using just the identifiers would lead to overfitting models. How to find a rule telling that order 99001 is composed of items 88124, 88125, and 88126? This is contained in the data and cannot be handled by a precise and explicit rule. As mentioned in the conclusion, this a topic for future research (cf. Section 10).

6. Discovering object-centric Petri nets

First, we introduce a general approach to learn accepting object-centric Petri nets from object-centric event logs. Then we discuss performance-related annotations of the models, model views, and ways to combine these results with traditional process mining techniques.

6.1. Generic approach

Given an object-centric event log $L = (E, \preceq_E)$ (Definition 3.3), we would like to discover an accepting object-centric Petri net $AN = (ON, M_{init}, M_{final})$ (Definition 5.5). Rather than defining one specific discovery algorithm, we present a general approach leveraging existing process discovery techniques.

- **Step 1:** Given an object-centric event log $L = (E, \preceq_E)$, identify the object types $OT \subseteq \mathbb{U}_{ot}$ appearing in the event log. Then create a flattened event log $L^{ot} = (E^{ot}, \preceq_E^{ot})$ for each object type $ot \in OT$.

- **Step 2:** Discover an accepting Petri net $SN^{ot} = (N^{ot}, M_{init}^{ot}, M_{final}^{ot})$ with $N^{ot} = (P^{ot}, T^{ot}, F^{ot}, l^{ot})$ for each object type $ot \in OT$ using the flattened event log L^{ot}. For this purpose, any conventional discovery technique can be used. The only assumption we need to make is that there are no duplicated labels, i.e., labeling function l^{ot} is injective. However, we allow for silent transitions, i.e., l^{ot} may be partial.

- **Step 3:** Merge the accepting Petri nets into a Petri net N. To avoid name clashes, first ensure that the place names and names of silent transitions in the different nets are different. Also, ensure that transitions that have the same label also have the same name (this is possible because the labeling functions are injective). After renaming, create an overall labeled Petri net $N = (P, T, F, l)$ with: $P = \bigcup_{ot \in OT} P^{ot}$, $T = \bigcup_{ot \in OT} T^{ot}$, $F = \bigcup_{ot \in OT} F^{ot}$, and $l = \bigcup_{ot \in OT} l^{ot}$.

- **Step 4:** Assign object types to the places in the merged Petri net N: $pt(p) = ot$ for $p \in P^{ot}$ and $ot \in OT$. This is possible because the places for the different object types are disjoint.
- **Step 5:** Identify the variable arcs $F_{var} \subseteq F$. This can be determined in different ways (e.g., using replay results or diagnosing the flattening process). The goal is to identify the arcs where multiple tokens need to be consumed or produced. An example would be $F_{var} = \{(p,t) \in F \cap (P \times T) \mid \mathit{score}(l(t), pt(p)) < \tau\} \cup \{(t,p) \in F \cap (T \times P) \mid \mathit{score}(l(t), pt(p)) < \tau\}$ where τ is a threshold (e.g., 0.98) and $\mathit{score} \in (\mathbb{U}_{act} \times \mathbb{U}_{ot}) \not\rightarrow [0,1]$ such that $\mathit{score}(act, ot) = |\{e \in E \mid \pi_{act}(e) = act \ \wedge \ |\pi_{omap}(e)(ot)| = 1\}| \, / \, |\{e \in E \mid \pi_{act}(e) = act\}|$ is the fraction of act events that refer to precisely one object of type ot.
- **Step 6:** Combining the previous three steps allows us to create an object-centric Petri net $ON = (N, pt, F_{var})$. The initial and final markings are obtained by replicating the markings of the accepting Petri nets for each of the corresponding objects. $M_{init} = [(p, oi) \in Q_{ON} \mid \exists_{ot \in OT} \ p \in M^{ot}_{init} \ \wedge \ \exists_{e \in E} \ oi \in \pi_{omap}(e)(pt(p))]$. $M_{\mathit{final}} = [(p, oi) \in Q_{ON} \mid \exists_{ot \in OT} \ p \in M^{ot}_{\mathit{final}} \ \wedge \ \exists_{e \in E} \ oi \in \pi_{omap}(e)(pt(p))]$.
- **Step 7:** Return the accepting object-centric Petri net $AN = (ON, M_{init}, M_{\mathit{final}})$.

The above approach has two parameters: (1) the discovery technique used in Step 2 and (2) the selection of variable arcs in Step 5 (e.g., threshold τ and function score). For Step 2 any discovery technique that produces a Petri net without duplicate labels can be used (e.g., region-based techniques without label splitting or the inductive mining techniques). The scoring function described in Step 5 is just an example. Function $\mathit{score}(act, ot)$ counts the fraction of act events that refer to precisely one object of type ot. If this is rather low (below the threshold τ), then the corresponding arcs are considered to be variable (i.e., these arcs can consume/produce any number of tokens). The approach always returns a well-formed object-centric Petri net because the selection of F_{var} depends on the transition and place type only.

6.2. Annotations, views, and extractions

The main novelty of the work presented in this paper is that we discover a single process model with multiple object types allowing us to capture multiple one-to-many and many-to-many relationships in event data. Based on this, many ideas from traditional process mining can be converted to this more realistic setting. In this section, we briefly discuss a few.

It is rather straightforward to annotate process models with frequency information and time information. For example, the right-hand side of Figure 6 is already showing various frequencies and our implementation provides much more diagnostics.

- **Transition annotations:** The frequency of a transition shows how often the corresponding activity occurred in the object-centric event log. It is also possible to add statistics about the objects involved in the corresponding events (e.g., how many objects of a particular type were involved on average). If there is transactional information (start and complete), it is also possible to show information about the duration of the corresponding activity (average, median, variance, minimum, maximum, etc.).

- **Place annotations:** It is possible to show how many tokens have been consumed from and produced for each place. These tokens correspond to objects. Hence, it is also possible to show how many unique objects visited the place and what the average number of visits per object is. By taking the time difference between the moment a token is produced and consumed, it is possible to show timing information (average, median, variance, minimum, maximum, etc.). In case of compliance checking, one can also show missing and remaining tokens (see implementation).
- **Arc annotations:** There are two types of arcs: the variable arcs F_{var} and the non-variable $F \setminus F_{var}$. Both can be annotated with frequency and time information. For variable arcs, we can also show statistics about the numbers of tokens produced/consumed per transition occurrence. See Figure 6, where the annotations for variable arcs show averages. For example, annotation 100×5 shows that 100 times a multiset of tokens was moved along the arc and the average size of this multiset was 5, indicating that 500 objects were moved along the arc.

Next to adding annotations, it is also possible to select or deselect object types. The approach described in Section 6.1 first identifies the object types $OT \subseteq \mathbb{U}_{ot}$ appearing in the event log. However, we can take any nonempty subset $OT' \subseteq OT$. It is, for example, possible to leave out the object type *Order* and only use the types *Item* and *Route*. This way it is possible to create simplified *views*. Everything can also be combined with frequency-based filtering, i.e., adding sliders to seamlessly remove infrequent activities and arcs.

Since most process mining techniques cannot handle object-centric event logs, it is valuable to be able to generate classical event logs and apply traditional techniques. The holistic view provided by the accepting object-centric Petri net serves as a starting point for a more detailed analysis focusing on one object type. Definition 4.1 already showed that it is easy to flatten event logs. It is also possible to take as case identifier combinations of object types. This can be combined with views and interactive filtering. Of course, one should always be very careful when interpreting such results. Due to the convergence and divergence problems mentioned before the results may be misleading. However, the overall accepting object-centric Petri net helps to avoid misinterpretations.

7. Tool support for object-centric Petri nets

The concepts and techniques discussed have been fully implemented. In this section, we describe the implementation, the functionalities supported, and evaluate the performance.

7.1. Implementation

To support the discovery approach presented in this paper (including performance and conformance analysis using token-based replay), we extended *PM4Py* with an additional Python library *PM4Py-MDL*.[6] The tool can be installed by using the Python Package Installer (PIP) (use the command *pip install pm4pymdl*). Next to discovering object-centric Petri nets, *PM4Py-MDL* can also discover multi-dimensional directly-follows graphs [39, 40].

[6]The software can be downloaded from `www.pm4py.org` and `https://github.com/Javert899/pm4py-mdl.git`

Our implementation follows the approach described in this paper. The discovery of an object-centric Petri net is based on the discovery of Petri nets for the single object types. Then, these Petri nets are merged and annotated. For the discovery of a Petri net for each of the individual object types, a sound workflow net is obtained by applying the Inductive Miner Directly-Follows process discovery algorithm [41]. However, any discovery technique producing an accepting Petri net can be used.

The token-based replay approach described in [42] is used to annotate the *places* and the performance on the *arcs*. This approach improves the approach [43] and the implementation is considerably faster. For each place, the number of *produced* p, *consumed* c, *remaining* r, and *missing* m tokens are computed and displayed. These values are obtained by "playing the token game" using the flattened event log L^{ot} and accepting Petri net SN^{ot} for each object type ot. This is possible because each place has precisely one type. The numbers p and c refer to the number of produced and consumed tokens (reported per place). The number of missing tokens m refers to situations where a token is not present in the place although the log suggests that the output transition has fired. The number of remaining tokens r refers to the tokens that remain after replaying the event log. Our token-based replay approach is able to deal with silent transitions and duplicate transitions (i.e., the labeling function l is partial or non-injective). See [43, 42] for details.

For performance-related annotations, the sets of delays based on differences between the production times of tokens and the consumption times of tokens are used. Based on these measurements, minimum, maximum, average, variance, etc. can be calculated.

The annotations related to the transitions are derived directly from the event log (i.e., without replaying the event log). This way we can add the frequencies of transitions, the average number of objects involved, and the number of unique objects to the model.

7.2. Functionalities of the tool

Figure 8 shows two screenshots of our *PM4Py-MDL* tool. The following functions are supported:

- *Importing* and *exporting* of object-centric event logs in different formats. The currently supported formats are Multi-Dimensional Logs (MDL), Parquet and XOC (format connected to OCBC models).
- A range of *object-centric process discovery approaches* are supported. There are also several target formats next to the object-centric Petri nets introduced in this paper. Also Multiple View-Point (MVP) models are supported. These are essentially Directly Follows Graphs [23] with colored arcs, see [39, 40]. The approach presented in the paper can be combined with different low-level discovery techniques. In the examples, we use the Inductive Miner Directly-Follows process discovery algorithm [41].
- It is possible to set various *thresholds* to influence the discovery process, e.g., the minimal number of occurrences for activities and paths. It is also possible to specify, for each object type, the activities that are considered for that type.
- Several methods to *explore* the raw event data are provided (e.g., statistics on the number of related objects per type and distribution of events over time). These annotations can be attached to places, transitions, and arcs.

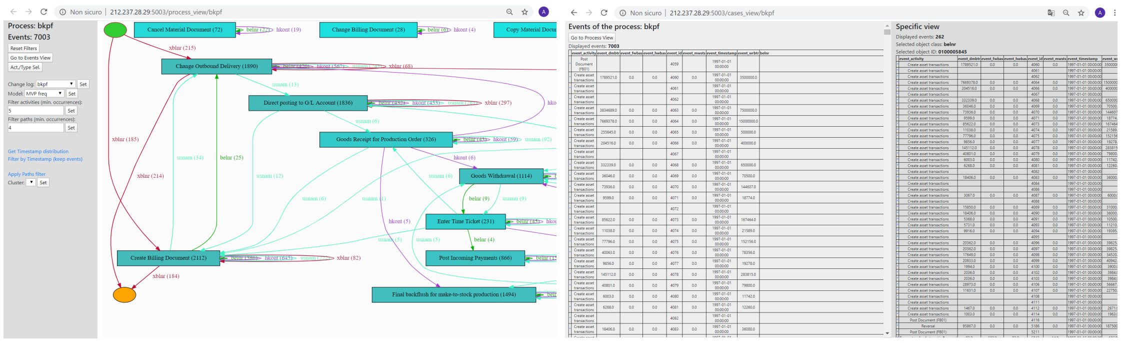

Process: bkpf
Events: 7003
Cancel Material Document (72)
Change Billing Document (28)
Change Outbound Delivery (1890)
Direct posting to G/L Account (1836)
Goods Receipt for Production Order (326)
Goods Withdrawal (114)
Enter Time Ticket (231)
Create Billing Document (2112)
Post Incoming Payments (866)
Final backflush for make-to-stock production (1494)
Events of the process: bkpf
Go to Process View
Displayed events: 7003
Specific view
Displayed events: 262
Selected object class: belnr

Figure 8. Web interface that is supporting the functionalities offered by the *PM4Py-MDL* library. The main components are the process discovery (left) and event exploration (right) ones.

- Token-based replay is supported for *performance* and *conformance* analysis. This allows for the identification of bottlenecks and deviating behavior.
- It is possible to *filter* based on activities, paths, number of related objects per type. Also, time-frame and attribute-based filters are supported.
- There is support for *clustering* and event *correlation* based on event graphs.
- At any point in time, it is possible to *flatten* an object-centric event log onto a *traditional event log* by selecting an object type. The resulting event log can be analyzed using conventional process mining techniques.

The web interface is organized mainly in two different components: process discovery and event exploration (see Figure 8). The visualization is highly interactive. The nodes are clickable in such a way that the statistics about the events of such activity can be inspected and filtering options can be set. The event exploration shows the events of the log in an interactive way. It is possible to interact with the related objects and show all the events related to an object in another panel. This way the understanding the lifecycle of objects is facilitated. Next, we evaluate the scalability of the approach and the implementation.

7.3. Scalability of the approach and implementation

The aim of this subsection is to analyze the scalability of the discovery of object-centric Petri nets as implemented in the *PM4Py-MDL* tool. We expect the discovery of object-centric Petri nets to be scalable, because the steps that are involved have at most linear complexity, excluding the application of the process discovery algorithm on the flattened logs. Moreover, we also support discovery techniques that are linear in the event log (given a bounded number of activities).

To analyze scalability, we use variants of the "running-example" object-centric event log also used in other parts of the paper. Three different settings have been examined:

1. The execution time of the algorithm in terms of *the number of events* in the event log (while keeping the number of unique activities and the number of objects per event constant).

2. The execution time of the algorithm in terms of *the number of unique activities* in the event log (while keeping the number of events and the number of objects per event constant).

3. The execution time of the algorithm in terms of *the number of objects per event* (while keeping the number of unique activities and the number of events constant).

7.3.1. Increasing the number of events

Figures 9 and 10(a) show the relationship between the overall execution time and the number of events in the object-centric event log. Figure 10(a) shows a linear relationship between the number of events and the execution time. Our initial "running-example" log contains 22,367 events. Different subsets of different sizes are taken such that the set of unique activities remains constant (we just consider fewer orders). In other words, the process is observed over shorter time periods. Analyzing the whole log takes less than a minute. This may seem long for a relatively small event log. However, the time needed for discovery is less than a second. Figure 9 splits the analysis time into six different components:

- The time needed for the log flattening operations for all event logs (**Log Fl.**).
- The time needed for the process discovery operations (**Disc.**). In these experiments, we use the inductive miner.

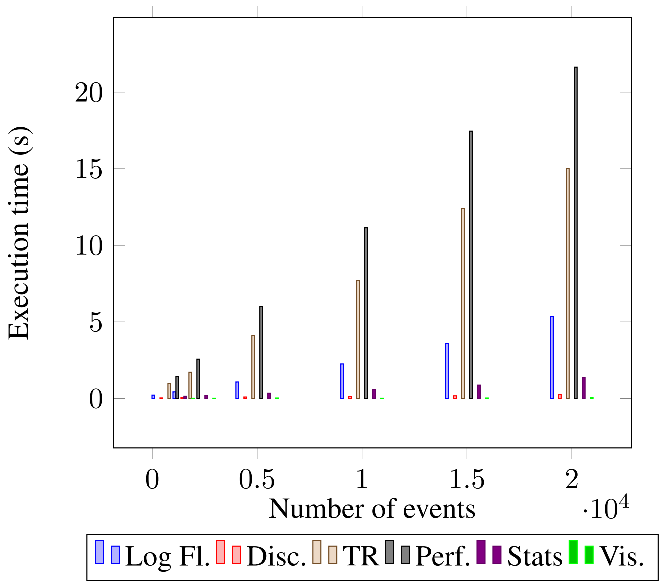


Figure 9. Detailed analysis of the overall execution time of the approach when increasing the number of events of the log. The measurements are grouped for the five sublogs. The columns inside a group represent event log flattening (**Log Fl.**), discovery (**Disc.**), token-based replay (**TR**), computing performance annotations (**Perf.**), computing statistics (**Stats**), and visualization (**Vis.**).

- The time needed for the token-based replay operations (**TR**).
- The time needed for computing the performance annotations based on the results of the token-based replay (**Perf.**).
- The time needed for the calculation of additional statistics from the log (**Stats**).
- The time needed for the visualization (**Vis.**).

Figure 9 clearly shows that most time is spent on the token-based replay operations (**TR**) and the computation of the performance annotations (**Perf.**). The first is done per control-flow variant (to avoid repeatedly solving the same problem) and the second one per object. This explains why **Perf.** takes more time than **TR**. Also, the event log preprocessing (**Log Fl.**) takes substantial time. Interestingly, the discovery itself is very fast compared to the other components.

Figures 9 and 10(a) show that the characteristics of our approach are similar to process mining on classical event logs. It takes more time to replay the event log to collect conformance and performance statistics than to discover the process model using techniques such as the inductive miner. This also holds for traditional process mining techniques using a single case notion.

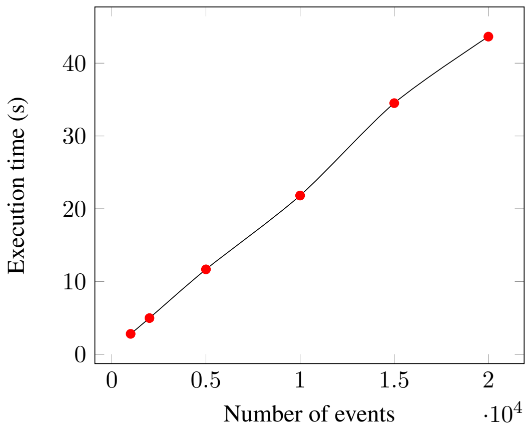
7.3.2. Increasing the number of activities

Figure 10(b) shows the execution time when increasing the number of unique activities. The event logs used were created using activity filtering while keeping the number of events constant. Table 7.3.2 shows the number of activities, the number of events, and the overall time needed. In row k, the k most frequent activities are retained and the event log is further filtered to have precisely 8159 events. The growth in overall computation time is explained by the fact that the most expensive operations are the token-based replay and computing the performance annotations, and the complexity of these operations grows linearly with the average length of the trace.[7]

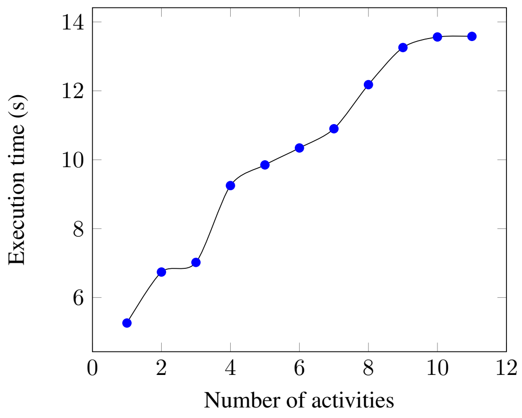
Table 3. The execution time while increasing the number of unique activities.

Number of Activities	Number of events	Execution time
1	8159	5.26
2	8159	6.74
3	8159	7.02
4	8159	9.25
5	8159	9.85
6	8159	10.34
7	8159	10.90
8	8159	12.18
9	8159	13.26
10	8159	13.56
11	8159	13.58

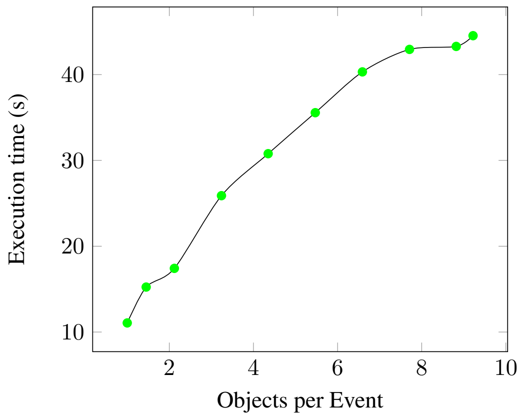
[7]Token-based replay, in contrast to other approaches such as alignments, does not suffer from the increase of the size of the trace, since decisions are made locally.



(a) Execution time while increasing the number of events.



(b) Execution time while increasing the number of activities.



c) Execution time while increasing the number of objects per event.

Figure 10. Scalability assessment of the object-centric Petri nets discovery algorithm. The different graphs show the overall time (including replay and annotation) when varying of the number of events, the number of activities, and the number of objects per event.

7.3.3. Increasing the number of related objects per event

Figure 10(c) shows the execution time when the number of related objects per event is increased. To analyze such a setting, different subsets of the "running-example" event log were created in such a way that the number of events and the number of different activities does not change. The set of related objects is selected such that at least one related object (of any type) remains for each event. The linear relation is as expected, since events are replicated for each object during analysis. Experiments also confirm that there is a linear relationship between the overall analysis time and the number of object *types* (not shown).

Overall, the results are very encouraging. *The approach scales linear in the number of events, the number of unique activities, and the number of objects.* The discovery times are negligible compared to the time needed for conformance checking and performance analysis. Hence, the approach can be applied to real-world event data.

Table 4. Fragment of a larger object-centric event log with 22,367 events and five object types: *orders*, *items*, *products*, *customers*, *packages*. There are 2000 different orders, 8159 items, 20 products, 17 customers, and 1325 packages. The table shows a few sample events and the first three object types.

event_activity	event_timestamp	orders	items	products
place order	2019-05-20 09:07:47	['990001']	['880001', '880002', '880003', '880004']	['Echo Show 8', 'Fire Stick 4K', 'Echo', 'Echo...
place order	2019-05-20 10:35:21	['990002']	['880005', '880006', '880007', '880008']	['iPad', 'Kindle', 'iPad Air', 'MacBook Air']
pick item	2019-05-20 10:38:17	['990002']	['880006']	['Kindle']
confirm order	2019-05-20 11:13:54	['990001']	['880001', '880002', '880003', '880004']	['Echo Show 8', 'Fire Stick 4K', 'Echo', 'Echo...
pick item	2019-05-20 11:20:13	['990001']	['880002']	['Fire Stick 4K']
place order	2019-05-20 12:30:30	['990003']	['880009', '880010', '880011', '880012']	['iPad Air', 'iPhone 11', 'Fire Stick', 'iPhon...
confirm order	2019-05-20 12:34:16	['990003']	['880009', '880010', '880011', '880012']	['iPad Air', 'iPhone 11', 'Fire Stick', 'iPhon...
item out of stock	2019-05-20 13:54:37	['990001']	['880004']	['Echo Studio']
place order	2019-05-20 14:20:47	['990004']	['880013', '880014']	['Echo Studio', 'Echo Show 8']
item out of stock	2019-05-20 15:19:49	['990003']	['880009']	['iPad Air']
place order	2019-05-20 16:01:22	['990005']	['880015', '880016']	['iPad Pro', 'iPad Air']
pick item	2019-05-20 16:56:02	['990004']	['880014']	['Echo Show 8']
pick item	2019-05-20 17:08:25	['990002']	['880008']	['MacBook Air']
place order	2019-05-20 17:22:31	['990006']	['880017', '880018', '880019']	['Echo Show 8', 'Fire Stick 4K', 'iPhone X']
pick item	2019-05-20 17:51:15	['990003']	['880011']	['Fire Stick']
pick item	2019-05-20 18:15:00	['990002']	['880007']	['iPad Air']
confirm order	2019-05-20 18:36:37	['990004']	['880013', '880014']	['Echo Studio', 'Echo Show 8']
place order	2019-05-20 19:04:49	['990007']	['880020', '880021', '880022']	['Echo Show 8', 'Echo Dot', 'Kindle Paperwhite']
...	...	...	...	...

8. Example application

To illustrate the feasibility of the approach and corresponding *PM4Py-MDL* implementation, we use the larger example briefly mentioned in the introduction (see Figure 1). The object-centric event log in CSV format can be obtained from `https://github.com/Javert899/pm4py-mdl/blob/master/example_logs/mdl/mdl-running-example.mdl`. A small fragment of the log, showing three selected object types, is visualized in Table 4. It can be considered to be an extension of the smaller examples used before. In total, there are 22,367 events. There are five object types: *orders*, *items*, *products*, *customers*, *packages*. The event log contains information about 2000 different orders, 8159 items, 20 products, 17 customers, and 1325 packages. Hence, the average number of items in one order is 4.08 and the average number of items in one package is 6.16.

We can filter out specific "activity - object type" (a, ot) combinations. This corresponds to removing objects related to activity a and object type ot. In Table 4, we removed all objects related to *customers* and *packages* for all activities. This boils down to removing the columns with customer and package information. We can also remove the rows related to certain activities. However, we can also use more fine-grained filtering where we keep specific "activity - object type" combinations.

Table 5. The first two columns show the "activity - object type" combinations used for analysis. For example, *place order* events also had information about products and customers, but these object types were removed. *failed delivery* events also had information about orders, products, and customers, but these were removed. Etc. The last three columns show statistics for the "activity - object type" combinations in the original event log (only for the object types *orders*, *items*, and *packages*). The three values are reported: the minimum number of objects / the average number of objects / the maximum number of objects.

Activity	Retained object types	Orders per event	Items per event	Packages per event
place order	orders, items	1 / 1.00 / 1	1 / 4.08 / 15	0 / 0.00 / 0
confirm order	orders, items	1 / 1.00 / 1	1 / 4.08 / 15	0 / 0.00 / 0
item out of stock	items	1 / 1.00 / 1	1 / 1.00 / 1	0 / 0.00 / 0
reorder item	items	1 / 1.00 / 1	1 / 1.00 / 1	0 / 0.00 / 0
pick item	items	0 / 0.00 / 0	1 / 1.00 / 1	0 / 0.00 / 0
payment reminder	orders	1 / 1.00 / 1	1 / 4.18 / 14	0 / 0.00 / 0
pay order	orders	1 / 1.00 / 1	1 / 4.08 / 15	0 / 0.00 / 0
create package	items, packages	1 / 3.32 / 9	1 / 6.16 / 22	1 / 1.00 / 1
send package	items, packages	1 / 3.32 / 9	1 / 6.16 / 22	1 / 1.00 / 1
failed delivery	items, packages	1 / 3.21 / 8	1 / 5.95 / 18	1 / 1.00 / 1
package delivered	items, packages	1 / 3.31 / 9	1 / 6.16 / 22	1 / 1.00 / 1

Table 5 shows example statistics for the 22,367 events in the original object-centric event log. For each activity, the minimum number of objects, the average number of objects, and the maximum number of objects of a given type are indicated. For example, *place order* events always refer to precisely one order object and a variable number of item objects (minimum=1, average=4.08, maximum=15) and *send package* events always refer to precisely one package object, a variable number of item objects (minimum=1, average=6.16, maximum=22), and a variable number of order objects (minimum=1, average=3.32, maximum=9).

For our running example, we considered the "activity - object type" combinations depicted in Table 5, i.e., we retain object types *orders*, *items*, and *packages*, keep all activities, but remove less relevant object types for some of the activities. Starting from the event log in Table 4 and the "activity - object type" combinations in Table 5, our discovery approach returns the object-centric Petri net shown in Figure 11.

The overall figure is hardly readable. However, we can use the filtering approaches discussed and seamlessly simplify the model (e.g., removing infrequent activities and selecting fewer object types). Moreover, we can zoom in on the different aspects of the model.

Figure 12 shows a fragment of the larger object-centric Petri net in Figure 11. The green source place is of type *orders*. The red source place is of type *items*. Activity *place order* occurred 2000 times, consuming precisely one token from the green place and a variable number of tokens from the red place. The compound double arrow reflects this, and the inscription shows that on average 4.08 item objects were consumed.

Figure 13 shows another fragment. The *package delivered* activity is the final activity of the lifecycle of both packages and items. The two compound double arrows denote that variable numbers of item objects are consumed and produced. The mean number of item objects consumed and produced by *package delivered* is 6.16. The annotations on the arcs tell that the average time from the previous activity for packages to this activity is 18 hours. The average time from the previous activity for items to this activity is 9 hours.

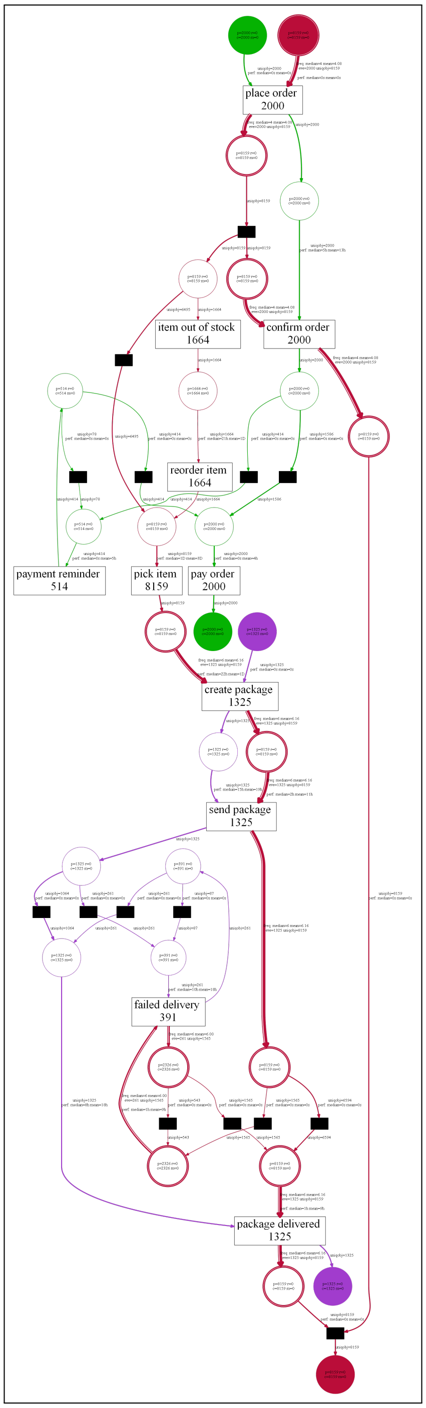


Figure 11. Object-centric Petri net discovered based on the example log considering three object types: *orders* (green), *items* (red), and *packages* (violet).

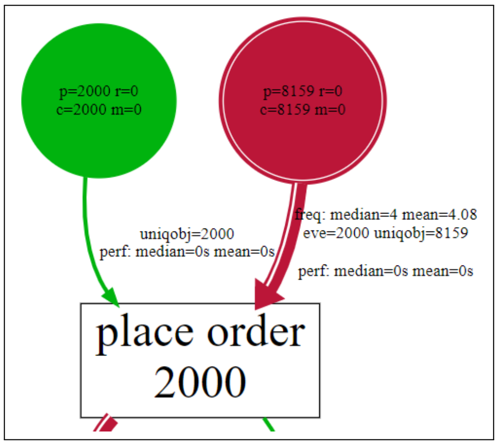


Figure 12. Fragment of the model showing the *place order* activity. There are 2000 unique orders in the log and *place order* occurs for each of them once. There are 8159 unique items distributed over the 2000 orders. The diagnostics show that, on average, 4.08 item objects are consumed from the red place of type *items*.

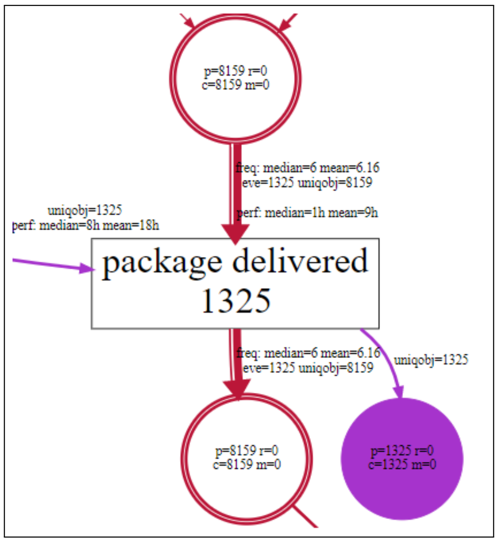


Figure 13. Fragment of the model showing the *package delivered* activity. This activity corresponds to the successful delivery of packages composed of multiple items. There are 8159 unique items distributed over 1325 packages. All packages were delivered as reflected by the frequency of *package delivered*. The mean number of item objects consumed and produced by the transition is 6.16. The number of package objects consumed and produced by the transition is always 1. Also, the average times are reported.

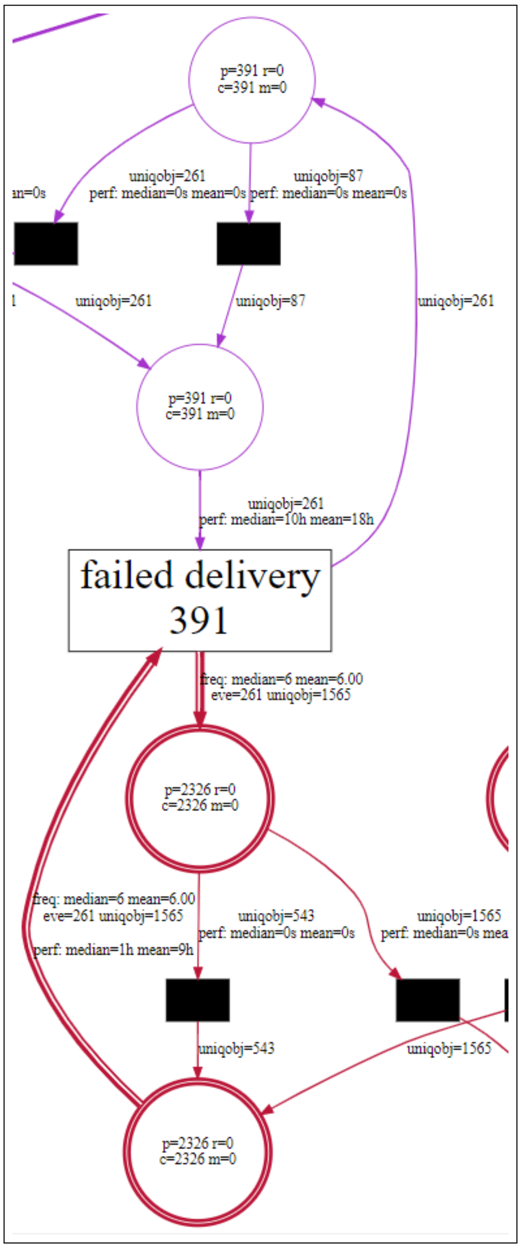


Figure 14. Fragment of the model showing the *failed delivery* activity. There were 391 failed deliveries, 261 packages had at least one failed delivery, and 1565 items out of the 8159 where involved in at least one failed delivery. 87 packages containing 543 items had a failed delivery two or more times.

Figure 14 zooms in on the failed deliveries. There were 391 failed deliveries involving 261 packages (87 failed multiple times) and 1565 items.

All the places in Figure 11 also show replay information: p is the number of tokens produced for the place, c is the number of tokens consumed, m is the number of tokens missing, r is the number of tokens remaining (see Section 7.1). In this example, the model fits perfectly. Therefore, $m = 0$ and $r = 0$ for all places.

From the discovered object-centric Petri net, we can also generate simpler views, create traditional event logs, and deploy traditional process mining techniques for further analysis. Moreover, the example shows many insights that could not have been discovered using traditional approaches. By flattening the event log, the relations between different types of objects would be lost. Moreover, any attempt to look at different types of objects would result in non-existing loops and misleading frequency/time diagnostics.

This section highlighted the main advantages of using our approach over traditional models using a separate process model for each object type. To summarize:

- Figure 11 (and the corresponding model fragments) provides an overview of the *whole* process and the *interactions* between the different object types. When considering each object type as a separate case notion, we get multiple *disconnected* models that do *not* show these interactions.
- *Deficiency*, *convergence*, and *divergence* problems are avoided (cf. Definition 4.3). All events are taken into count precisely once, i.e., events do not disappear and are not replicated unintentionally. Moreover, artificial loops due to divergence are avoided.
- Using token-based replay, we are able to project *performance and conformance information* onto one overall model. Most of the statistics would not be visible in the flattened process models (e.g., the average number of objects involved in an activity).

9. Related work

This section discusses traditional process mining techniques using a single case notion, modeling approaches dealing with multiple object types, and process mining approaches dealing with multiple object types.

9.1. Traditional process mining techniques using a single case notion

In the introduction, we mentioned several process discovery approaches based on classical event logs using a single case notion [34]. Many of these techniques discover classical Petri nets (e.g., place transition nets), e.g., region-based approaches can be used to derive places [2, 3, 11, 13, 15, 20, 8, 7, 6, 10, 14, 9, 5, 12, 19, 17, 16, 21, 22]. The region-based process discovery techniques are just a subset of all approaches to derive process models from event logs. The inductive mining techniques [24, 25] and the so-called split miner [26] are examples of the state-of-the-art techniques to learn process models. Commercial systems tend to use the Directly Follows Graph (DFG) having the obvious limitations explained in [23]. All of the above approaches assume a single case notion. This is consistent with traditional process models ranging from *workflow nets* [44, 45] and *process trees* [46] to *Business Process Modeling Notation (BPMN) models* [47] and *Event-driven Process Chains (EPCs)* [48] which assume a single case notion.

9.2. Modeling techniques using multiple object types

Although most process models use a single case notion, the problem that many processes cannot be captured properly in this way was identified early on. IBM's *FlowMark* system already supported the so-called "bundle" concept to handle cases composed of subcases [49]. This is related to the *multiple instance patterns*, i.e., a category of *workflow patterns* identified around the turn of the century [50]. One of the first process modeling notations trying to address the problem were the so-called *proclets* [51, 52]. 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* [53, 54, 55, 56]. See [57] 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.

9.3. Process mining techniques using multiple object types

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. This problem was also raised in Section 5.5 of [34] which discusses the need to "flatten" event data to produce traditional process models.

In [58] 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 [35] is the official IEEE standard for storing event data and supported by many process mining vendors. XES requires a case notion to correlate events. Next to the standard IEEE XES format [35], new storage formats such as *eXtensible Object-Centric* (XOC) [59] 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 [31, 32, 33] focus on interacting processes where each process uses its own case identifiers. In [33] interacting artifacts are discovered from ERP systems. In [31] 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 [29, 30]. These are also called *artifact-centric process models* but are simpler than the models used in [53, 54, 31, 32, 55, 56, 33]. 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 explicit 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 [30].

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 [60, 61, 62, 28]. 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* [63]. Instead of LTL-based constraints, simpler cardinality constraints are used. Several discovery techniques have been developed for OCBC models [28]. 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 [39, 40]. 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. The approach to discover object-centric Petri nets can be seen as a continuation of the work in [39, 40].

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 implemented in commercial systems. 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. In earlier papers [27, 39, 40], we already stressed the need for object-centric process mining. In this paper, we provided a concrete, but at the same time generic, discovery approach to learning object-centric Petri nets from object-centric events logs.

10. Conclusion

When looking at data models or database schemas, there are often one-to-many and many-to-many relations between different types of objects relevant for a process. Since mainstream process modeling and process mining approaches enforce the use of a specific case notion, the modeler or analyst is forced to select a specific perspective. This problem can be partly addressed by extracting multiple event logs to cover the different case notions and considering one model per case notion. It would be better to have one, more holistic, process model that is showing the interactions between the different types of objects. Moreover, the need to pick one or more specific case notions for analysis leads to the divergence and convergence problems discussed in this paper.

Therefore, this paper uses *object-centric event logs* as a representation in between the actual data in the information system and traditional event logs (e.g., based on XES). Object-centric event logs do not depend on a case notion. Instead, events may refer to arbitrary sets of objects. One event may refer to multiple objects of different types. Next to using a different input format, we also use a different target language: *object-centric Petri nets*. These nets are a restricted variant of colored Petri nets where places are typed, tokens refer to objects, and transitions correspond to activities. Unlike other mainstream notations, a transition can consume and produce a variable number of objects of different types. We presented a concrete, but also generic, approach to discover object-centric Petri nets from object-centric event logs. The approach has been implemented in *PM4Py* and various applications show that the approach provides novel insights and is highly scalable (linear in the number of objects, object types, events, and activities). Therefore, the ideas are directly implementable in commercial tools and the existing software can be used to analyze real-life event data in larger organizations.

This is the first paper that aims to learn object-centric Petri nets from object-centric event logs. Our findings show lots of opportunities for further research. These include:

- We aim to develop conformance checking techniques based on object-centric Petri nets and object-centric event logs. Next to checking whether the event log can be replayed on the process model, it is interesting to detect outliers using the cardinalities. In the current implementation, we already report missing and remaining tokens, but these are based on the flattened event logs.

- The approach presented is generic and can embed different process discovery algorithms independently working on flattened events logs (inductive miner, region-based techniques, etc.). The results are then folded into object-centric Petri nets. It is interesting to compare the different approaches and develop more integrated approaches (e.g., first discover a process model for one object type and then iteratively add the other object types). Moreover, it would be good to have dedicated quality measures (e.g., complexity and precision).

- Object-centric Petri nets in their current form can be seen as "over-approximations" of the actual behavior. It is interesting to think of ways to make the model more precise (e.g., automatically detecting guards or relating splits and joins). For example, in Figure 4, transition *marked as complete* should join the same set of objects earlier involved in an occurrence of transition *place order*. Similarly, we would like to add stochastics to the model (e.g., a probability distribution for the number of items in an order).

- The current object-centric event logs only contain object identifiers and no properties of objects. If an object identifier refers to a patient or customer, we do not know her age, weight, address, income, etc. If an object identifier refers to an order or machine, we do not know its value, The absence of object attributes automatically leads to the "over-approximations" mentioned. Hence, we are developing extended object-centric event logs.
- We also plan to investigate more sophisticated forms of performance analysis that go beyond adding timing a frequency diagnostics to transition, places, and arcs. How do the different object types influence each other? Next to analyzing the interactions between objects, we would like to better support the link to CPN Tools for "what if" analysis (e.g., replaying the event log on an improved process).
- We also aim to create a comprehensive, publicly available, set of object-centric event logs.

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

References

[1] Ehrenfeucht A, Rozenberg G. Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Informatica*, 1989. **27**(4):315–368. doi:10.1007/BF00264612.

[2] Badouel E, Bernardinello L, Darondeau P. Petri Net Synthesis. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2015. doi:10.5555/2851516.

[3] Badouel E, Darondeau P. Theory of Regions. In: Reisig W, Rozenberg G (eds.), Lectures on Petri Nets I: Basic Models, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998 pp. 529–586. doi:10.1007/3-540-65306-6_22.

[4] Desel J, Reisig W. The Synthesis Problem of Petri Nets. *Acta Informatica*, 1996. **33**(4):297–315. doi:10.1007/ s002360050046.

[5] Kleijn J, Koutny M, Pietkiewicz-Koutny M, Rozenberg G. Applying Regions. *Theoretical Computer Science*, 2017. **658**:205–215. doi:10.1016/j.tcs.2016.01.040.

[6] Cortadella J, Kishinevsky M, Lavagno L, Yakovlev A. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 1998. **47**(8):859–882. doi:10.1109/12.707587.

[7] Carmona J, Cortadella J, Kishinevsky M, Kondratyev A, Lavagno L, Yakovlev A. A Symbolic Algorithm for the Synthesis of Bounded Petri Nets. In: Applications and Theory of Petri Nets (Petri Nets 2008). 2008 pp. 92–111. doi:10.1007/978-3-540-68746-7_10.

[8] Carmona J, Cortadella J, Kishinevsky M. New Region-Based Algorithms for Deriving Bounded Petri Nets. *IEEE Transactions on Computers*, 2010. **59**(3):371–384. doi:10.1109/TC.2009.131.

[9] Kleijn J, Koutny M, Pietkiewicz-Koutny M. Regions of Petri nets with a/sync connections. *Theoretical Computer Science*, 2012. **454**:189–198. doi:10.1016/j.tcs.2012.04.016.

[10] Darondeau P. On the Synthesis of Zero-Safe Nets. In: Concurrency, Graphs and Models, volume 5065 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2008 pp. 364–378. doi:10.1007/978-3-540-68679-8_25.

[11] Bergenthum R, Desel J, Lorenz R, Mauser S. Process Mining Based on Regions of Languages. In: Alonso G, Dadam P, Rosemann M (eds.), International Conference on Business Process Management (BPM 2007), volume 4714 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2007 pp. 375–383. doi:10.1007/978-3-540-75183-0_27.

[12] Lorenz R, Bergenthum R, Desel J, Mauser S. Synthesis of Petri Nets from Finite Partial Languages. In: Basten T, Juhás G, Shukla S (eds.), International Conference on Application of Concurrency to System Design (ACSD 2007). IEEE Computer Society, 2007 pp. 157–166. doi:10.1109/ACSD.2007.34.

[13] Bergenthum R, Desel J, Lorenz R, Mauser S. Synthesis of Petri Nets from Finite Partial Languages. *Fundamenta Informaticae*, 2008. **88**(4):437–468. doi:10.1109/ACSD.2007.34.

[14] van Dongen B, Desel J, van der Aalst WMP. Aggregating Causal Runs into Workflow Nets. In: Jensen K, van der Aalst WMP, Marsan MA, Franceschinis G, Kleijn J, Kristensen L (eds.), Transactions on Petri Nets and Other Models of Concurrency (ToPNoC VI), volume 7400 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2012 pp. 334–363. doi:10.1007/978-3-642-35179-2_14.

[15] Bergenthum R, Desel J, Mauser S, Lorenz R. Synthesis of Petri Nets from Term Based Representations of Infinite Partial Languages. *Fundamenta Informaticae*, 2009. **95**(1):187–217. doi:10.3233/FI-2009-147.

[16] Lorenz R, Juhás G. How to Synthesize Nets from Languages: A Survey. In: Henderson S, Biller B, Hsieh M, Shortle J, Tew JD, Barton RR (eds.), Proceedings of the Wintersimulation Conference (WSC 2007). IEEE Computer Society, 2007 pp. 637–647. doi:10.1109/WSC.2007.4419657.

[17] Lorenz R, Juhas G. Towards Synthesis of Petri Nets from Scenarios. In: Donatelli S, Thiagarajan P (eds.), Application and Theory of Petri Nets 2006, volume 4024 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2006 pp. 302–321. doi:10.1007/11767589_17.

[18] Bergenthum R, Desel J, Lorenz R, Mauser S. Synthesis of Petri Nets from Scenarios with VipTool. In: Applications and Theory of Petri Nets (Petri Nets 2008), volume 5062 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2008 pp. 388–398. doi:10.1007/978-3-540-68746-7_25.

[19] Lorenz R, Desel J, Juhas G. Models from Scenarios. In: Jensen K, van der Aalst WMP, Balbo G, Koutny M, Wolf K (eds.), Transactions on Petri Nets and Other Models of Concurrency (ToPNoC VII), volume 7480 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2013 pp. 314–371. 10.1007/978-3-642-38143-0_9.

[20] Carmona J, Cortadella J, Kishinevsky M. A Region-Based Algorithm for Discovering Petri Nets from Event Logs. In: Business Process Management (BPM 2008). 2008 pp. 358–373. doi:10.1007/978-3-540-85758-7_26.

[21] Solé M, Carmona J. Process Mining from a Basis of State Regions. In: Lilius J, Penczek W (eds.), Applications and Theory of Petri Nets 2010, volume 6128 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2010 pp. 226–245. doi:10.1007/978-3-642-13675-7_14.

[22] van Zelst S, van Dongen B, van der Aalst WMP, Verbeek H. Discovering Workflow Nets Using Integer Linear Programming. *Computing*, 2018. **100**(5):529–556. doi:10.1007/s00607-017-0582-5.

[23] van der Aalst WMP. 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*. Elsevier, 2019 pp. 321–328. doi:10.1016/j.procs.2019.12.189.

[24] Leemans S, Fahland D, van der Aalst WMP. Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. In: Lohmann N, Song M, Wohed P (eds.), Business Process Management Workshops, Int. Workshop on Business Process Intelligence (BPI 2013), vol. 171 of *Lecture Notes in Business Information Processing*. Springer, 2014 pp. 66–78. doi:10.1007/978-3-319-06257-0_6.

[25] Leemans S, Fahland D, van der Aalst WMP. Scalable Process Discovery with Guarantees. In: Gaaloul K, Schmidt R, Nurcan S, Guerreiro S, Ma Q (eds.), Enterprise, Business-Process and Information Systems Modeling (BPMDS 2015), volume 214 of *Lecture Notes in Business Information Processing*. Springer-Verlag, Berlin, 2015 pp. 85–101. doi:10.1007/978-3-319-19237-6_6.

[26] Augusto A, Conforti R, Marlon M, La Rosa M, Polyvyanyy A. Split Miner: Automated Discovery of Accurate and Simple Business Process Models from Event Logs. *Knowledge Information Systems*, 2019. **59**(2):251–284. doi:10.1007/s10115-018-1214-x.

[27] van der Aalst WMP. Object-Centric Process Mining: Dealing With Divergence and Convergence in Event Data. In: Ölveczky P, Salaün G (eds.), Software Engineering and Formal Methods (SEFM 2019), volume 11724 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2019 pp. 3–25. doi:10.1007/978-3-030-30446-1_1.

[28] Li G, Medeiros de Carvalho R, van der Aalst WMP. Automatic Discovery of Object-Centric Behavioral Constraint Models. In: Abramowicz W (ed.), Business Information Systems (BIS 2017), volume 288 of *Lecture Notes in Business Information Processing*. Springer-Verlag, Berlin, 2017 pp. 43–58. doi:10.1007/978-3-319-59336-4_4.

[29] van Eck M, Sidorova N, van der Aalst WMP. Guided Interaction Exploration in Artifact-centric Process Models. In: IEEE Conference on Business Informatics (CBI 2017). IEEE Computer Society, 2017 pp. 109–118. doi:10.1109/CBI.2017.42.

[30] van Eck M, Sidorova N, van der Aalst WMP. Multi-instance Mining: Discovering Synchronisation in Artifact-Centric Processes. In: Daniel F, Sheng Q, Motahari H (eds.), Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2018), volume 342 of *Lecture Notes in Business Information Processing*. Springer-Verlag, Berlin, 2018 pp. 18–30. doi:10.1007/978-3-030-11641-5_2.

[31] Fahland D, de Leoni M, van Dongen B, van der Aalst WMP. Behavioral Conformance of Artifact-Centric Process Models. In: Abramowicz A (ed.), Business Information Systems (BIS 2011), volume 87 of *Lecture Notes in Business Information Processing*. Springer-Verlag, Berlin, 2011 pp. 37–49. doi:10.1007/978-3-642-21863-7_4.

[32] Fahland D, de Leoni M, van Dongen B, van der Aalst WMP. Many-to-Many: Some Observations on Interactions in Artifact Choreographies. In: Eichhorn D, Koschmider A, Zhang H (eds.), Proceedings of the 3rd Central-European Workshop on Services and their Composition (ZEUS 2011), CEUR Workshop Proceedings. CEUR-WS.org, 2011 pp. 9–15. URL: http://ceur-ws.org/Vol-705/paper1.pdf.

[33] Lu X, Nagelkerke M, van de Wiel D, Fahland D. Discovering Interacting Artifacts from ERP Systems. *IEEE Transactions on Services Computing*, 2015. **8**(6):861–873. doi:10.1109/TSC.2015.2474358.

[34] van der Aalst WMP. Process Mining: Data Science in Action. 2016. doi:10.1007/978-3-662-49851-4_1.

[35] IEEE Task Force on Process Mining. XES Standard Definition. www.xes-standard.org, 2013.

[36] Lu X, Fahland D, van der Aalst WMP. Conformance Checking Based on Partially Ordered Event Data. In: Fournier F, Mendling J (eds.), Business Process Management Workshops, International Workshop on Business Process Intelligence (BPI 2014), volume 202 of *Lecture Notes in Business Information Processing*. Springer-Verlag, Berlin, 2015 pp. 75–88. doi:10.1007/978-3-319-15895-2_7.

[37] van der Aalst WMP, Stahl C. Modeling Business Processes: A Petri Net Oriented Approach. MIT Press, Cambridge, MA, 2011. ISBN: 978-0-262-01538-7.

[38] Jensen K, Kristensen L. Coloured Petri Nets. Springer-Verlag, Berlin, 2009. doi:10.1007/b95112.

[39] Berti A, van der Aalst WMP. StarStar Models: Using Events at Database Level for Process Analysis. In: Ceravolo P, Keulen M, Lopez MG (eds.), International Symposium on Data-driven Process Discovery and Analysis (SIMPDA 2018), volume 2270 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2018 pp. 60–64. URL: http://ceur-ws.org/Vol-2270/short3.pdf.

[40] Berti A, van der Aalst WMP. Discovering Multiple Viewpoint Models from Relational Databases. In: Ceravolo P, Keulen M, Lopez MG (eds.), Postproceedings International Symposium on Data-driven Process Discovery and Analysis, volume 379 of *Lecture Notes in Business Information Processing*. Springer-Verlag, Berlin, 2020 pp. 24–51. doi:10.1007/978-3-030-46633-6_2.

[41] Leemans S, Fahland D, van der Aalst WMP. Scalable Process Discovery and Conformance Checking. *Software and Systems Modeling*, 2018. **17**(2):599–631. doi:10.1007/s10270-016-0545-x. doi:10.1007/s10270-016-0545-x.

[42] Berti A, van der Aalst WMP. Reviving Token-based Replay: Increasing Speed While Improving Diagnostics. In: Proceedings of the International Workshop on Algorithms and Theories for the Analysis of Event Data (ATAED 2019), volume 2371 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019 pp. 87–103. URL: `http://ceur-ws.org/Vol-2371/ATAED2019-87-103.pdf`.

[43] Rozinat A, van der Aalst WMP. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems*, 2008. **33**(1):64–95. doi:10.1016/j.is.2007.07.001.

[44] van der Aalst WMP. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 1998. **8**(1):21–66. doi:10.1142/S0218126698000043,

[45] van der Aalst WMP, Hee K, ter Hofstede A, Sidorova N, Verbeek H, Voorhoeve M, Wynn M. Soundness of Workflow Nets: Classification, Decidability, and Analysis. *Formal Aspects of Computing*, 2011. **23**(3):333–363. doi:10.1007/s00165-010-0161-4.

[46] Leemans S, Fahland D, van der Aalst WMP. Discovering Block-structured Process Models from Event Logs: A Constructive Approach. In: Colom J, Desel J (eds.), Applications and Theory of Petri Nets 2013, volume 7927 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2013 pp. 311–329. doi:10.1007/978-3-642-38697-8_17.

[47] OMG. Business Process Model and Notation (BPMN). Object Management Group, formal/2011-01-03, 2011. URL: https://www.omg.org/spec/BPMN/2.0/PDF.

[48] Scheer A. Business Process Engineering: Reference Models for Industrial Enterprises. Springer-Verlag, Berlin, 1994. ISBN: 978-3-540-58234-2.

[49] IBM. IBM MQSeries Workflow - Getting Started With Buildtime. IBM Deutschland Entwicklung GmbH, Boeblingen, Germany, 1999. URL: `ftp://public.dhe.ibm.com/ps/products/workflow/docu/v322/pdf/enu/fmcu0mst.pdf`.

[50] van der Aalst WMP, ter Hofstede A, Kiepuszewski B, Barros A. Workflow Patterns. *Distributed and Parallel Databases*, 2003. **14**(1):5–51. doi:10.1023/A:1022883727209.

[51] van der Aalst WMP, Barthelmess P, Ellis C, Wainer J. Workflow Modeling using Proclets. In: Etzion O, Scheuermann P (eds.), 7th International Conference on Cooperative Information Systems (CoopIS 2000), volume 1901 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2000 pp. 198–209. doi:10.1007/10722620_20.

[52] van der Aalst WMP, Barthelmess P, Ellis C, Wainer J. Proclets: A Framework for Lightweight Interacting Workflow Processes. *International Journal of Cooperative Information Systems*, 2001. **10**(4):443–482. doi:10.1142/S0218843001000412.

[53] Bhattacharya K, Gerede C, Hull R, Liu R, Su J. Towards Formal Analysis of Artifact-Centric Business Process Models. In: Alonso G, Dadam P, Rosemann M (eds.), International Conference on Business Process Management (BPM 2007), volume 4714 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2007 pp. 288–304. doi:10.1007/978-3-540-75183-0_21.

[54] Cohn D, Hull R. Business Artifacts: A Data-centric Approach to Modeling Business Operations and Processes. *IEEE Data Engineering Bulletin*, 2009. **32**(3):3–9. doi:10.1.1.183.76.

[55] Lohmann N. Compliance by Design for Artifact-Centric Business Processes. In: Rinderle S, Toumani F, Wolf K (eds.), Business Process Management (BPM 2011), volume 6896 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2011 pp. 99–115. doi:10.1016/j.is.2012.07.003.

[56] Nigam A, Caswell N. Business artifacts: An Approach to Operational Specification. *IBM Systems Journal*, 2003. **42**(3):428–445. doi:10.1147/sj.423.0428.

[57] Fahland D. Describing Behavior of Processes with Many-to-Many Interactions. In: Donatelli S, Haar S (eds.), Applications and Theory of Petri Nets 2019, volume 11522 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2019 pp. 3–24. doi:10.1007/978-3-030-21571-2_1.

[58] de Murillas EGL, Reijers H, van der Aalst WMP. Connecting Databases with Process Mining: A Meta Model and Toolset. In: Schmidt R, Guedria W, Bider I, Guerreiro S (eds.), Enterprise, Business-Process and Information Systems Modeling (BPMDS 2015), volume 248 of *Lecture Notes in Business Information Processing*. Springer-Verlag, Berlin, 2016 pp. 231–249. doi:10.1007/s10270-018-0664-7.

[59] Li G, de Murillas EGL, de Carvalho RM, van der Aalst WMP. Extracting Object-Centric Event Logs to Support Process Mining on Databases. In: Mendling J, Mouratidis H (eds.), Information Systems in the Big Data Era, CAiSE Forum 2018, volume 317 of *Lecture Notes in Business Information Processing*. Springer-Verlag, Berlin, 2018 pp. 182–199. doi:10.1007/978-3-319-92901-9_16.

[60] van der Aalst WMP, Artale A, Montali M, Tritini S. 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*. 2017 URL: `http://ceur-ws.org/Vol-1879/paper51.pdf`.

[61] van der Aalst WMP, Li G, Montali M. Object-Centric Behavioral Constraints. *CoRR*, 2017. **abs/1703.05740**. URL: `http://arxiv.org/abs/1703.05740`.

[62] Artale A, Calvanese D, Montali M, van der Aalst WMP. Enriching Data Models with Behavioral Constraints. In: Borgo S (ed.), Ontology Makes Sense (Essays in honor of Nicola Guarino). IOS Press, 2019 pp. 257–277. doi:10.3233/978-1-61499-955-3-257.

[63] van der Aalst WMP, Pesic M, Schonenberg H. Declarative Workflows: Balancing Between Flexibility and Support. *Computer Science - Research and Development*, 2009. **23**(2):99–113. doi:10.1007/s00450-009-0057-9.