# Explainable Concept Drift in Process Mining

Jan Niklas Adams[a,*], Sebastiaan J. van Zelst[b,a], Thomas Rose[b,c], Wil M.P. van der Aalst[a,d,b]

[a]*Process and Data Science, RWTH Aachen University, Ahornstraße 55, Aachen, 52074, North Rhine-Westphalia, Germany*
[b]*Fraunhofer Institute for Applied Information Technology, Konrad-Adenauer-Straße, Sankt Augustin, Bonn, 53757, North Rhine-Westphalia, Germany*
[c]*Information Systems and Databases, RWTH Aachen University, Ahornstraße 55, Aachen, 52074, North Rhine-Westphalia, Germany*
[d]*Celonis SE, Theresienstraße 6, Munich, 80333, Bavaria, Germany*

**Abstract**

The execution of processes leaves trails of event data in information systems. These event data are analyzed to generate insights and improvements for the underlying process. However, companies do not execute these processes in a vacuum. The fast pace of technological development, constantly changing market environments, and fast consumer responses expose companies to high levels of uncertainty. This uncertainty often manifests itself in significant changes in the executed processes. Such significant changes are called concept drifts. Transparency about concept drifts is crucial to respond quickly and adequately, limiting the potentially negative impact of such drifts. Three types of knowledge are of interest to a process owner: When did a drift occur, what happened, and why did it happen. This paper introduces a framework to extract concept drifts and their potential root causes from event data. We extract time series describing process measures, detect concept drifts, and test these drifts for correlation. This framework generalizes existing work such that *object-centric event data with multiple case notions, non-linear relationships*, and *an arbitrary number of process measures are supported.* We provide an extendable implementation and evaluate our framework concerning the sensitivity of the time series construction and scalability of cause–effect testing. Furthermore, we provide a case study uncovering an explainable concept drift.

*Keywords:* Process Mining, Concept Drift, Cause-Effect, Object-Centric Process Mining, Explainability

## 1 Introduction

Throughout the past decades, information systems have become an elementary component supporting various businesses. Executions of business processes [1] leave traces of event data in such information systems, describing the conducted actions. These data can be analyzed to generate insights and improvements for the supported processes. The techniques dealing with these types of problems are summarized under the term *process mining* [2]. Applying such techniques helps businesses increasing efficiency through understand their processes and is, therefore, a fundamental part of every competitive company.

Different process mining techniques have different objectives of knowledge discovery. Process discovery, for example, delivers a visual representation of the possible executions paths within a process [3]. Conformance checking techniques allow a user to uncover deviations of process executions and measure the cor-

*Corresponding author

*Email addresses:* `niklas.adams@pads.rwth-aachen.de` (Jan Niklas Adams), `s.j.v.zelst@pads.rwth-aachen.de` (Sebastiaan J. van Zelst), `thomas.rose@fit.fraunhofer.de` (Thomas Rose), `wvdaalst@pads.rwth-aachen.de` (Wil M.P. van der Aalst)

respondence of a process to the recorded event data [4]. Other techniques aim to identify and solve problems within a process, e.g., by simulation [5] or by predicting problematic process executions [6].

*Concept drift* is one process-related problem that has drawn attention over the past years [7, 8, 9]. A concept drift is a significant change in a process over time. The occurrence of a concept drift may lead to several problems or inefficiencies. However, concept drifts are challenging to detect and analyze because the process is already dynamic, exhibiting stochastic behavior. In the literature, three main questions about problems related to concept drifts are of importance: **(1)** *When did the concept drift occur?* If a concept drift happens in the control-flow of the process, i.e., the ordering of executed actions, process discovery, conformance checking, and predictive monitoring techniques have to be adjusted accordingly [10]. **(2)** *What happened and how did it happen?* Significant changes need to be understood by the process owner to react accordingly [11]. **(3)** *Why did it happen?* Significant changes might be caused by other changes in the process [12]. Uncovering potential cause-effects helps uncover problems and improve the process.

These three questions are the main research lines related to concept drift in process mining. Different works aim to solve one or multiple of these questions. In this work, we focus on dealing with the third research question. However, we can also generate insights for the first two research questions as a byproduct of our work.

**RQ1** **Detection** The existence of a concept drift needs to be detected, and its change point needs to be located as precisely as possible.

**RQ2** **Characterisation** The nature of the concept drift needs to be described as accurately and extensively as possible.

**RQ3** **Explanation** Potential cause-effects of a concept drift contained within the event data need to be uncovered.

Several techniques have addressed **RQ1** over the last years, with a focus on solving this problem for the control-flow of a process [13]. Generally, these techniques first calculate some numeric representation of the control-flow and, subsequently, use one of many change point detection algorithms [14] such as hypothesis testing, cost-based segmentation techniques, or visual inspection. Different techniques have been proposed to answer **RQ2**, both in an interactive and fully automated manner. Yeshchenko et al. [9] provide an extensive visual analytics framework for humans to explore concept drifts and understand the dynamics and changes behind them. Ostovar et al. [15] uncover change patterns of a concept drift by testing the data against predefined business process change patterns. These are given as textual output to the user. In recent work, we propose a general framework to answer **RQ3** [12]. In this paper, we generalize and extend our previous work to address several challenges encountered in real-life information systems.

The first of these challenges is the so-called "object-centricity" of information systems: Traditional process mining techniques assume the existence of a single case notion and that each recorded event is associated with exactly one object of the case notion. In reality, an information system, e.g., an ERP system, consists of many case notions, e.g., different document types, and events may be related to multiple objects of different case notions [16]. To apply traditional process mining techniques, these object-centric event data have to be *flattened* first, forcing them into traditional event log format[17]. Flattening is related to certain problems (cf. Subsection 4.1) and provides misleading insights. Therefore, process mining techniques have to be adapted into the object-centric setting to provide accurate insights. Recently, academia and industry have picked up this challenge. On the one hand, many techniques have been proposed to translate traditional process mining problems to the object-centric setting [18, 19, 20, 21, 22, 23, 24]. On the other hand, industry leaders provide initial support of object-centric event data, e.g., Celonis by supporting a secondary case identifier, or Mehrwerk Process Mining by supporting multiple case identifiers.

The second challenge is non-linear cause-effect relationships contained in information systems. The investigation of cause-effect relationships behind concept drifts cannot be limited to linear relationships. Different dynamics in processes, e.g., workloadproductivity relationships of resources [25], may show non-linear behavior.

The third challenge is the absence of domain knowledge. One cannot always assume that a basic knowledge or suspicion of a candidate perspective for a concept drift and its potential cause is present. In our original framework, the user has to choose one perspective to be investigated for concept drifts and another perspective to be investigated for potential causes. However, an approach stripped from its necessity for domain knowledge would be more generally applicable.

Therefore, the work presented in this paper is a generalization and extension of our original framework [12] in the following ways: a) Our revised framework supports event data with multiple case notions. b) We support the detection of non-linear relationships. c) No choice about a primary and secondary perspective has to be made. The user can choose arbitrarily many features.

Our new framework is depicted in Figure 1. The event log is first segmented into subsequent windows. For each of these windows, we calculate multiple numerical features subject to the user's choice. These values are concatenated into a time series for each feature. Subsequently, we detect concept drifts in these time series. For each pair of features and each pair of concept drifts of the two series, we test for Granger causality [26] given the time difference between drifts. Non-linear relationships are covered by applying a kernel function. Granger-causal concept drift pairs are given to the user as explainable concept drifts.

We answer **RQ1** by detecting concept drifts using existing concept drift detection techniques. The time series provide a visualization to explore the nature of the concept drift, helping in answering **RQ2**. The correlated concept drifts give explanations and potential root causes for drifts, answering **RQ3**.

The remainder of this paper is structured as follows: Section 2 introduces related work on concept drift in process mining. We formalize object-centric event data in Section 3. Extracting time series from object-centric event logs is introduced in Section 4. We give a general definition for concept drift detection in Section 5. Section 6 introduces Granger causality for time series and testing for non-linear relationships. Our general framework and a short overview of the implementation is given in Section 7. In Section 8, the framework is, first, evaluated for sensitivity and scalability and, second, applied to a real-life event log uncovering an explainable concept drift in a case study. We conclude this paper in Section 9.

## 2 Related Work

Over the past years, many techniques dealing with problems related to concept drifts in processes have been introduced. This section discusses the scope of these approaches compared to our framework. Furthermore, we discuss the differentiation of this work from our previous work [12], for which this work constitutes an extension and generalization. Table 1 depicts an overview of the scope of papers on concept drift in process mining.

Most papers deal with the detection of the drift, i.e., pinpointing the occurrence of the drift in time. [13] provides an extensive overview of the techniques and the feature encodings and algorithms they use. Characterizing refers to describing the drift's nature, i.e., the underlying change, e.g., the decrease of an indicator or the removal of an activity. Most techniques use either visual analytics [9] or automatically detect change patterns [15]. Since our technique provides a time series, it can be used as a starting point to visually characterize the concept drift even though we do not explicitly aim to provide a characterization technique. The explanation of a concept drift refers to finding potential reasons, e.g., pre-occurring, correlated concept drifts, for a concept drift within the event data itself. This problem was, so far, only tackled in [12]. However, our framework introduced in [12] only allows for the detection of linear relationships. Furthermore, two features have to be explicitly chosen. In this work, we generalize this framework by allowing for the testing of non-linear relationships. Furthermore, we allow for an arbitrary number of features of interest to be chosen and automatically test all combinations. At last, we generalize the framework such that object-centric event data can be used as an input. Object-centric event data is a general

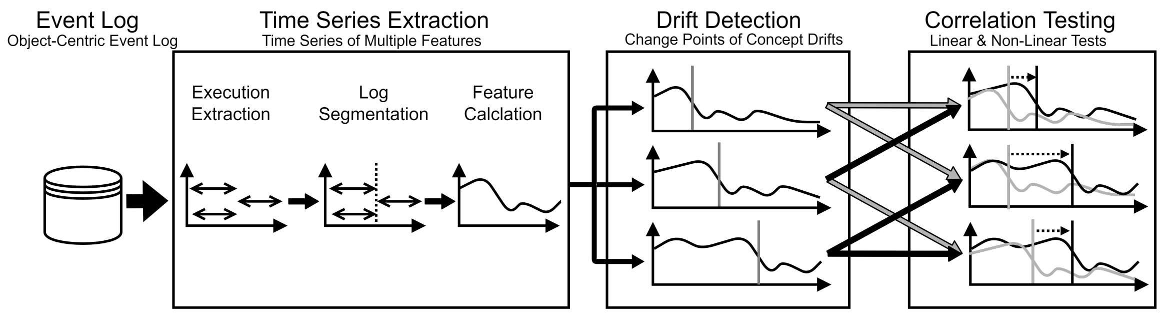

Figure 1: General Framework for deriving explainable concept drifts from an event log. First, multiple time series are extracted based on a set of features. These time series are investigated for concept drifts. We test for linear and non-linear relationships between each pair of concept drifts between time series.

Table 1: Overview of the scope of related approaches for concept drift detection in process mining. The framework proposed in this paper is the first technique to cover object-centric event data. A plethora of techniques deal with detecting and characterizing drifts. This paper extends the work of [12] by also considering non-linear relationships and object-centric event data with multiple case notions.

| | Object-Centric Event Data | Detect. | Charact. | Explanation | |
|---|---|---|---|---|---|
| | | | | Linear | Non-Linear |
| [27], [7], [8], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37] | | ✓ | | | |
| [15], [38], [39], [11], [9], [40] | | ✓ | ✓ | | |
| [12] | | ✓ | (✓) | ✓ | |
| This work | ✓ | ✓ | (✓) | ✓ | ✓ |

form of event data with multiple case notions for a single event [18].

Traditional "flat" event data consists of a set of event sequences where each event belongs to exactly one event sequence, and each sequence is of the same type. Each sequence is associated to an object of that type, e.g., a specific customer or a manufacturing item. In contrast, *object-centric event data* generalize the classical event log notion and allow for events that refer to any number of objects of different types. Therefore, events can be shared between the sequences of different objects, expressing interactions. The new OCEL standard [41] provides a data format for such event data.

The problem of applying traditional process mining techniques on such a general form of event data has been formulated in [17]. Object-centric event data can only be used by traditional process mining techniques when first *flattening* them. However, flattening leads to altered event data and, therefore, misleading results (cf. Subsection 4.1). Therefore, process mining techniques need to be adapted to object-centric event data. Adams et al. [23] propose to generalize the process mining concepts cases and variants from sequences to graphs, allowing for accurate support of object-centric event data. The graph-based approach is, also, supported by other authors encountering problems of object-centricity [19, 22, 20]. We generalize the framework proposed in [12] such that object-centric event data can also be processed by using the concept of process executions, i.e., object-centric cases[1]. This is the first work dealing with concept drift in object-centric event data to the best of our knowledge.

Inferring causality between time series has been an

[1]Please note, traditional event data is a specific case of object-centric event data. Therefore, our approach is backward compatible.

important task in many application domains such as medicine [42], finance [43] or social sciences [44]. As the basic definition of Granger causality [26] only allows for the testing of linear relationships between univariate time series, many adaptations to different needs such as multivariate time series [45] or non-linear relationships [46] have been proposed. The extensions to non-linear relationships are of interest to our framework. In essence, solutions incorporating non-linear relationships [47] employ the kernel trick [48]. Instead of projecting the data into a high-dimensional feature space, the data is transformed within the feature space using a kernel function. After applying a suitable kernel function, the relationship can be detected by linear classifiers. Granger causality for non-linear relationships employs kernel functions by applying a non-linear transformation to one time series and testing for standard Granger causality. Employed kernel functions are either explicitly chosen [49] or learned [47]. In our framework, we employ a general case of choosing a kernel function, leaving the opportunity to learn a kernel function if necessary.

## 3 Event Data

First, we introduce some notations used throughout this paper. A sequence $\sigma : \{1, \ldots, n\} \rightarrow X$ assigns positions to elements of a set $X$. We denote a sequence $\sigma \in X^*$ with $\sigma = \langle x_1, \ldots, x_n \rangle$ for elements $x_1, \ldots, x_n \in X$. A sequence $\sigma = \langle x_1, \ldots, x_n \rangle$ is of length $len(\sigma) = n$. We denote subsequences with $\sigma(l, k) = \langle x_j, \ldots, x_k \rangle$ for $l < k$. The notation $x \in \sigma$ is overloaded to express $x \in range(\sigma)$.

Event data describe the executions of a process as a collection of events. $\mathcal{E}$ is the universe of events. Each event corresponds to the execution of a single action in a process, i.e., an *activity*. The activity is an attribute of the event. Additionally to the activity attribute, an event can contain many different attributes, such as cost or other associated data. The universe of attributes is denoted with $\mathcal{A}$. The universe of attribute values is denoted with $\mathcal{V}$. Each event has a timestamp describing the execution time. The universe of timestamps is the positive real numbers $\mathbb{R}_0^+$. The universe of time intervals is denoted by $\mathcal{TI} = \{(t_{start}, t_{end}) \in \mathbb{R}_0^+ \times \mathbb{R}_0^+ \mid t_{end} > t_{start}\}$.

Events are associated with cases. A case describes a process instantiation. Traditionally, each event is associated with exactly one case [2]. For many processes, this is a valid assumption. Imagine an insurance claim handling process: Each claim is a new instantiation of the process, i.e., a new case. No event is associated with multiple claims since each claim is handled separately. However, this assumption does not hold for all types of event data generated by processes: There might not always be one clear case identifier for each event. Consider the example of an order handling process depicted in Table 2: We assume that the information system records the process of ordering items at an online shop. Orders of multiple items are placed. These items are collected, packed together, and shipped together. There are events with just a single case identifier, e.g., an item being picked (cf. $e_2$ in Table 2). However, there are also events with multiple case identifiers, e.g., two items that are shipped together (cf. $e_9$ in Table 2).

In this general setting of event data, we refer to case identifiers as *objects*. $\mathcal{O}$ denotes the universe of all objects. For traditional, single-case event logs, such objects would be homogeneous, i.e., all objects would be of the same type. In the example of the insurance claim handling process, each object would be of the type insurance claim. However, when looking at the example of the order handling process, this assumption is also invalid: Objects can be of different types, e.g., of type order or type item. $\mathcal{OT}$ denotes the universe of all object types. Every object has one type $\pi_{type} : \mathcal{O} \rightarrow \mathcal{OT}$. Furthermore, objects can have attributes. In summary, such event data describe multiple instances of different processes that overlap with non-trivial relationships.

An event log is a set of events related to objects and attributes. Each object is associated with a sequence of events. This general form of an event log is known as an *object-centric event log*[2].

**Definition 1** (Event Log) *Let $A \subseteq \mathcal{A}$ be a set of*

[2]`www.ocel-standard.org`

*event attributes, $V \subseteq \mathcal{V}$ be the set of possible attribute values and $T \subseteq \mathcal{T}$ be a set of timestamps. An event log $L = (E, O, \pi_{time}, trace, \pi_{attr})$ is a tuple of*

- *a set of event identifiers $E \subseteq \mathcal{E}$,*
- *a set of objects $O \subseteq \mathcal{O}$,*
- *a timestamp mapping $\pi_{time} : E \rightarrow T$,*
- *a sequence mapping $trace : O \rightarrow E^*$ such that $\forall_{\langle e_1,\ldots,e_n\rangle \in range(trace)} \forall_{1 \leq i < n} \; \pi_{time}(e_i) < \pi_{time}(e_{i+1})$*
- *a mapping of attributes to attribute values for events and objects $\pi_{attr} : E \times A \nrightarrow V \cup O \times A \nrightarrow V$*

An example of such an event log is depicted in Table 2. Each row corresponds to an event. Events can be associated with objects of two types: orders and items. For example, event $e_1$ is associated to object o1 of type order and objects i1, i2 of type item. Furthermore, each event has an activity, timestamp, and an additional attribute, the associated cost. Each object is associated to the sequence of events of its appearances, e.g., $trace(i1) = \langle e_1, e_3, e_5, e_{12}, e_{13}\rangle$ or $trace(o3) = \langle e_{14}, e_{16}, e_{17}\rangle$. Using multiple objects for events enables an event log to express interactions and dependencies between event sequences of different objects.

## 4 Time Series Extraction

This section introduces a general approach for transforming an object-centric event log into a time series. This approach is split into three steps: First, process executions are extracted from the event log. Second, the event log is segmented into windows of timeframes. Third, a numerical feature is calculated for each window.

### *4.1 Process Execution Extraction*

Traditionally, extracting process executions (also: cases or process instances) from an event log is trivial: Each event belongs to exactly one sequence, given by an identifier. The sequence of each identifier is considered to be one process execution. However, this is not possible in object-centric event data: events may belong to multiple sequences. Therefore, process executions consist of multiple, connected sequences forming an event graph rather than an event sequence. We extend the process execution definition typically used in process mining to event graphs of connected objects. To extract connected objects, we use the concept of an object graph, describing the co-occurrence relationships between objects in the event log.

**Definition 2** (Object Graph) *Let $L = (E, O, \pi_{time}, trace, \pi_{attr})$ be an event log. We introduce the following notations:*

- *$obj_L(e) = \{o \in O \mid e \in trace(o)\}$ are the objects associated to an event,*
- *$G_O = (O, \{\{o, o'\} \in O \times O \mid o \neq o' \wedge \exists_{e \in E} \; o, o' \in obj_L(e)\})$ is the object graph of an event log.*

The objects of an event log form the nodes of the object graph. The edges are introduced between two objects that share an event. The left-most column of Figure 2 depicts the object graph for Table 2. Connected subgraphs of this object graph describe objects that directly or transitively depend on each other by sharing events. We use a general definition of a process execution containing connected objects and their events. A process execution is a graph of events constructed by merging the event sequences of connected objects.

**Definition 3** (Process Execution) *Let $L = (E, O, \pi_{time}, trace, \pi_{attr})$ be an event log and $G_O$ the associated object graph. For a set of objects $X \subseteq O$ forming a connected subgraph in $G_O$, a process execution $p_X = (E_X, D_X)$ consists of*

- *events $E_p = \{e \in E \mid X \cap obj_L(e) \neq \emptyset\}$*
- *directly-follows relations $D_X = \{(e, e') \in E \times E \mid \exists_{o \in X} \; trace(o) = \langle e_1, \ldots, e_n\rangle \wedge \exists_{1 \leq i < n} e = e_i \wedge e' = e_{i+1}\}$*

Each connected subgraph of the object graph defines a process execution. Selecting a subset of connected subgraphs of the object graph to extract process executions is a non-trivial decision and influences

Table 2: Exemplary event log in table format, depicting two event attributes, i.e., activity and cost. Object attributes are dropped for accessibility reasons.

| ID | activity | Order | Item | timestamp | cost |
|---|---|---|---|---|---|
| $e_1$ | Place Order | o1 | i1,i2 | 03.03.2022 12:15 | 1 |
| $e_2$ | Pick Item |  | i2 | 03.03.2022 14:21 | 3 |
| $e_3$ | Out of Stock |  | i1 | 03.03.2022 14:57 | 4 |
| $e_4$ | Place Order | o2 | i3,i4,i5 | 04.03.2022 07:12 | 1 |
| $e_5$ | Reorder |  | i1 | 04.03.2022 08:45 | 2 |
| $e_6$ | Pick Item |  | i3 | 04.03.2022 14:01 | 2 |
| $e_7$ | Pick Item |  | i5 | 04.03.2022 14:01 | 2 |
| $e_8$ | Pick Item |  | i4 | 04.03.2022 14:06 | 3 |
| $e_9$ | Ship Order | o2 | i3,i4,i5 | 04.03.2022 14:56 | 120 |
| $e_{10}$ | Receive Payment | o2 |  | 06.03.2022 09:00 | 6 |
| $e_{11}$ | Receive Payment | o1 |  | 06.03.2022 09:03 | 4 |
| $e_{12}$ | Reorder arrived |  | i1 | 07.03.2022 08:32 | 10 |
| $e_{13}$ | Ship Order | o1 | i1,i2 | 07.03.2022 08:01 | 80 |
| $e_{14}$ | Place Order | o3 | i6 | 10.03.2022 15:38 | 1 |
| $e_{15}$ | Pick Item |  | i6 | 10.03.2022 16:59 | 3 |
| $e_{16}$ | Ship Order | o3 | i6 | 10.03.2022 17:07 | 40 |
| $e_{17}$ | Receive Payment | o3 |  | 13.03.2022 09:01 | 4 |

the interpretation of results. We define three extraction techniques to extract process executions through certain classes of subgraphs.

**Definition 4** (Execution Extraction) *Let $L = (E, O, \pi_{time}, trace, \pi_{attr})$ be an event log and $G_O$ the associated object graph. We define three different process execution extraction techniques:*

- $ex_{flat}(L, ot) = \{p_{\{o\}} \mid o \in O \wedge \pi_{type}(o) = ot\}$ *for an object type* $ot \in \mathcal{OT}$
- $ex_{comp}(L) = \{p_X \mid X \subseteq O \wedge X$ *forms a connected component in* $G_O\}$
- $ex_{lead}(L, ot) = \{p_X \mid o \in O \wedge \pi_{type}(o) = ot \wedge X = \{o' \in O \wedge dist(o, o') \neq \perp \wedge \neg\exists_{o'' \in O}\ \pi_{type}(o') = \pi_{type}(o'') \wedge dist(o, o'') < dist(o, o')\}\}$ *where* $dist : O \times O \to \mathbb{N} \cup \{\perp\}$ *provides the shortest path length between two objects in* $G_O$.

*The universe of all execution extraction functions is defined as* $\mathbb{EX} = \{ex_{flat}, ex_{comp}, ex_{lead}\}$

For each extraction technique, we depict the considered subgraphs of the object graph and the retrieved process executions in Figure 2. $ex_{flat}$ is the extraction technique known as flattening [17]: One chooses one object type and collects all objects of this type. The event sequences of each of those objects are considered to be process executions. While this is straightforward and the resulting process executions are compatible with all traditional process mining techniques, there are three main problems associated with flattening. First, events not associated with any object of the chosen types will be discarded, eliminating information. This is called *deficiency.* Second, events that are associated with multiple objects of the chosen object type will be included in al object's process executions, leading to a duplication of events and potentially misleading results. This is called *convergence.* Third, events of different object types might be forced into a precedence relationship, introducing misleading precedence constraints. This can best be exemplified by thought experiment: One could create composite objects of co-appearing objects of

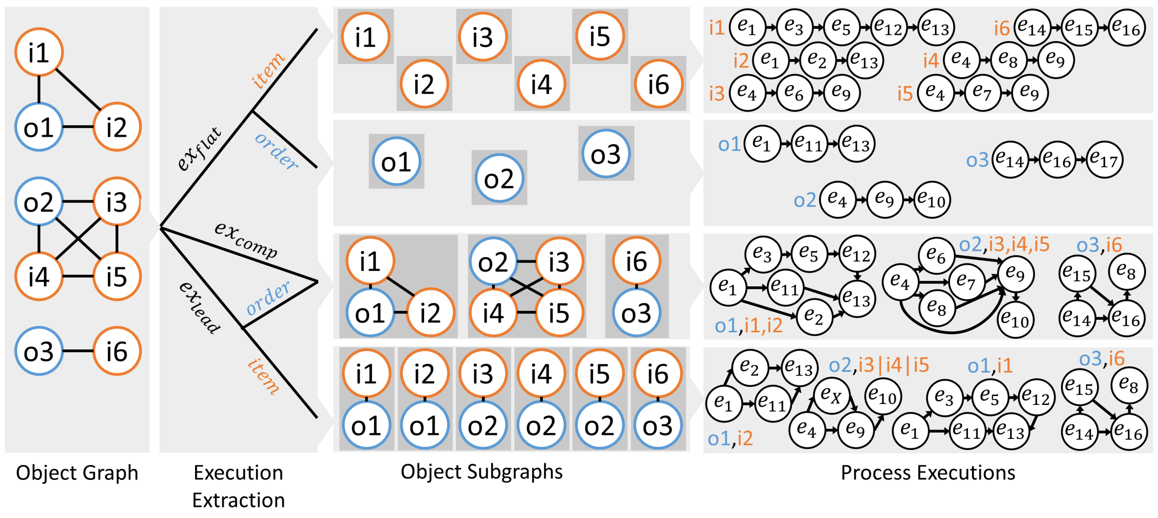

Figure 2: Object graph for Table 2. We present different execution extraction techniques and the process execution they would retrieve on the exemplary event log of tab:ocel.

different types and flatten using this new composite object type. This would tackle the deficiency and convergence problem. Use the objects $o2, i3, i4, i5$ of Table 2 as an example. A composite object would be associated with the events $\{e_4, e_6, e_7, e_8, e_9\}$. Flattening with this composite object, one would retrieve a sequence where three *Pick item* events follow each other. This implies some precedence between these activities that is not present in the actual event log. Therefore, flattening can not be used to retrieve process executions that are free of deficiency, convergence, and divergence.

Since flattening can not accurately capture the structure of object-centric event data further extraction techniques are needed. One extraction technique is connected components extraction $ex_{comp}$. Each connected component of the object graph is a set of transitively dependent objects. By extracting one process execution for each connected component all dependencies contained in the event log can be encoded into the process executions. The set of connected components is a partitioning of the graph, i.e., it is mutually exclusive but collectively exhaustive. Therefore, neither deficiency nor convergence problems are present in the resulting event log. The precedence constraints of the event log are transferred to the process executions as they are contained in the event log. Therefore, no misleading precedence constraints are added.[3] However, there is one drawback using connected components: There may be large clusters of transitively connected objects. For some tasks, this might render the retrieved process executions too complex.

We use a third technique $ex_{lead}$ allowing users to extract only a subset of transitively dependent objects. This aims to achieve a balance between incorporated dependencies and size of the retrieved process executions. One leading object type is chosen by the user. Each object of this leading type forms the basis of one process execution. Starting from this object the object graph is traversed for transitively related objects. All objects are added to the process execution for which no other object of the same type exists that is reachable with a shorter shortest path length. Depending on the chosen leading object type

[3]Depending on how the object-centric event log is extracted and how objects are logged on the event level, misleading precedence constraints may still be introduced. Imagine logging the associated order at each *Pick item* activity.

different results are retrieved (cf. Figure 2). Similar to $ex_{comp}$ no misleading precedence constraints are added. Therefore, the divergence problem is not present in this extraction technique. However, this extraction technique may come with convergence and deficiency issues: Objects not connected to any leading object will not end up in any process execution (deficiency) and some objects might end up in multiple process executions (convergence, cf. Figure 2). The discussed drawbacks should be considered when interpreting the results. [4]

Scalability experiments for $ex_{comp}$ and $ex_{lead}$ can be found in [23]. The extraction technique $ex_{comp}$ will be used for the remainder of the running example.

### *4.2 Log Segmentation*

A log segmentation generates sub logs from an event log. As we are interested in the development of the process over time, we need strategies for segmenting the event log based on time attributes. First, we define the start and end of a process execution.

**Definition 5** (Start and End of a Process Execution) *Let $L = (E, O, \pi_{time}, trace, \pi_{attr})$ be an event log, $ex \in \mathbb{EX}$ an extraction technique, and $p_X = (E_X, D_X) \in ex(L)$ a process execution. We introduce the following notations:*

- $Start_{p_X} = min(\{\pi_{time}(e) \mid e \in E_X\})$
- $End_{p_X} = max(\{\pi_{time}(e) \mid e \in E_X\})$

Table 3 depicts the start and end of the process executions derived from Table 2 using $ex_{comp}$. We need to extract the behavior of an event log specific to a certain time interval. However, there can be different strategies to decide whether event data belongs to a certain time interval. These can be based on the time interval covered by process executions or on the timestamp of the event. We introduce five inclusion functions.

**Definition 6** (Inclusion Function) *Let $L = (E, O, \pi_{time}, trace, \pi_{attr})$ be an event log, $ex \in \mathbb{EX}$ an extraction technique, and $P = ex(L)$ a set of process executions. An inclusion function maps process executions onto a set of process executions given a time interval. We define different inclusion functions:*

- $f^{in}_{start}(P, (t_{start}, t_{end})) = \{p \in P \mid Start_p \geq t_{start} \wedge Start_p < t_{end}\}$
- $f^{in}_{end}(P, (t_{start}, t_{end})) = \{p \in P \mid End_p \geq t_{start} \wedge End_p < t_{end}\}$
- $f^{in}_{contained}(P, (t_{start}, t_{end})) = \{p \in P \mid Start_p \geq t_{start} \wedge End_p \leq t_{end}\}$
- $f^{in}_{spanning}(P, (t_{start}, t_{end})) = \{p \in P \mid (Start_p \leq t_{start} \wedge End_p \geq t_{start}) \vee (Start_p \leq t_{end} \wedge End_p \geq t_{end})\}$

*Furthermore, we define an event-based inclusion function $f^{in}_{event}(P, (t_{start}, t_{end})) = \{p_Y = (E_Y, D_Y) \mid p_X = (E_X, D_X) \in f^{in}_{spanning}(P, (t_{start}, t_{end})) \wedge E_Y = \{e \in E_X \mid t_{start} \leq \pi_{time}(e) \leq t_{end}\} \wedge Y = \{o \in X \mid \exists_{e \in E_Y}\ e \in trace(o)\} \wedge D_Y = D_X \cap E_Y \times E_Y\}$. $\mathbb{F}^{in} = \{f^{in}_{start}, f^{in}_{end}, f^{in}_{contained}, f^{in}_{spanning}, f^{in}_{event}\}$ is the universe of inclusion functions.*

Figure 3 depicts the different inclusion functions. Process executions are included based on start and end timestamps. Our four execution-based techniques cover all reasonable inclusion strategies: executions starting within a time interval, ending within a time interval, starting and ending in a time interval, and overlapping a time interval. Furthermore, we provide one event-based inclusion function. Events are included based on their timestamps, not the timestamps of the process executions. This also has one important side effect: Events of the same process execution can end up in different windows. Therefore, the process execution is split between different windows. Instead of one process execution, as contained in the original event log, the windows contain shorter parts of the original process execution.

[4]If the object-centric event log only contains one object type and each event is associated with exactly one object of that type (i.e. a traditional event log) all of our extraction techniques yield the traditional sequential case concept. Therefore, our approach is a generalization of traditional process execution extraction and backward-compatible.

Table 3: Start and end timestamp for the process executions of Table 2

| Process Execution | $Start_p$ | $End_p$ |
|---|---|---|
| $p_{\{o1,i1,i2\}}$ | 03.03.2022 12:15 | 07.03.2022 08:01 |
| $p_{\{o2,i3,i4,i5\}}$ | 04.03.2022 07:12 | 06.03.2022 09:00 |
| $p_{\{o3,i6\}}$ | 10.03.2022 15:38 | 13.03.2022 09:01 |

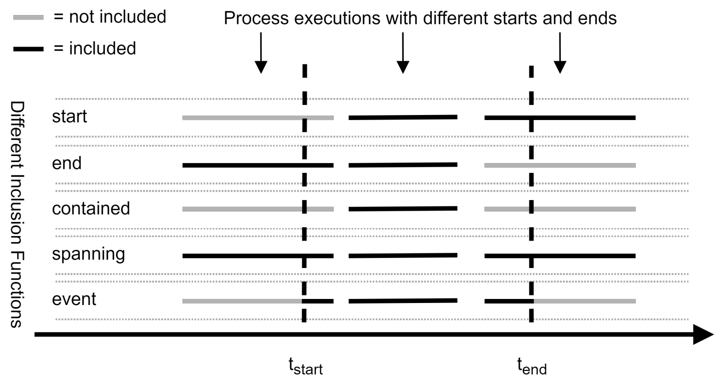
= not included
= included
Process executions with different starts and ends
Different Inclusion Functions
start
end
contained
spanning
event
$t_{start}$
$t_{end}$

Figure 3: The three lines in each lane show the time intervals covered by three process executions. For a time interval $[t_{start}, t_{end}]$, the figure depicts which process executions are included for which functions. Grey areas indicate not included events, black areas indicate included ones. For the events inclusion function, process executions can be partly included.

Based on the later usage of the windows, e.g., calculating the number of process executions per window, this might lead to misleading measures.

The event log can be segmented into subsequent sub logs using a window size (time interval) and an inclusion function. For each subsequent window, the behavior of the event log specific to this time interval is extracted using the inclusion function. We retrieve a sequence of sets of process executions.

**Definition 7** (Log Segmentation) *Let $L = (E, O, \pi_{time}, trace, \pi_{attr})$ be an event log, $ex \in \mathbb{EX}$ an extraction technique, and $P = ex(L)$ a set of process executions, $w \in \mathbb{R}_0^+$ be a window size and $f^{in} \in \mathbb{F}^{in}$ be an inclusion function. We introduce the following notations:*

- *The start time of an event log is $Start_L = min(\{\pi_{time}(e) \mid e \in E\})$,*
- *The end time is $End_L = max(\{\pi_{time}(e) \mid e \in E\})$,*
- *The number of sublogs is $m = \lceil \frac{End_L - Start_L}{w} \rceil$.*

*$seg(P, w, f^{in}) = \langle P_1, \ldots, P_m \rangle$ with $P_i = f^{in}(P, (Start_L + i \cdot w, Start_L + (i+1) \cdot w))$ for $i \in \{1, \ldots, m\}$ segment the event log into $m$ sublogs of subsequent time intervals based on the inclusion function.*

We show an example of such a log segmentation based on the event log in Table 2 and the process executions of Table 3. We use a window size $w = 7$ days. The start time of the event log is 03.03.2022 12:15, the end time 13.03.2022 09:01. We retrieve $m = 2$ sublogs. With the inclusion function $f^{in}_{contained}$, the segmentation yields the following sublogs $seg(\{p_{\{o1,i1,i2\}}, p_{\{o2,i3,i4,i5\}}, p_{\{o3,i6\}}\}, 7 \text{ days}, f^{in}_{contained}) = \langle \{p_{\{o1,i1,i2\}}, p_{\{o2,i3,i4,i5\}}\}, \{p_{\{o3,i6\}}\} \rangle$.

## *4.3 Feature Calculation*

A feature is a numerical measure for an event log. It can be used to describe one aspect of a process. We, first, give a general definition of a feature extraction function. Subsequently, we provide a taxonomy of features used throughout process mining to describe different perspectives and measures of a process. We describe the challenges of adapting these features to the object-centric setting for each of these perspectives. Finally, we define the retrieval of the time series based on a segmented event log and a feature.

#### *4.3.1 Object-Centric Features*

First, we provide a general definition for a feature extraction function.

**Definition 8** (Feature Extraction Function) *Let $L$ be an event log, $ex \in \mathbb{EX}$ an extraction technique, and $P = ex(L)$ a set of process executions. A feature function $feat(P) \in \mathbb{R}$ maps process executions onto a real valued number. We denote the universe of feature functions with $\mathcal{F}$.*

Table 4 describes a taxonomy of features used in process mining to describe different perspectives of a process. We extend the work in [12] by the perspective of objects as we are looking at object-centric event data. Thus, we divide features into five perspectives: control-flow, performance, data, resources, and objects. Most of the features can be trivially translated to the object-centric setting. However, for some features, a non-trivial adaptation is necessary and will be discussed in the following paragraphs. Please note that all features used throughout process mining can be used when flattening the data, i.e., selecting an object type and extracting the process executions for each object of that type. A recent overview of feature adaptations to the object-centric setting can be found in [56].

*Control-Flow.* Control-flow features are numerical representations of the activity relationships within event sequences typically used for the discovery of process models. We, furthermore, consider conformance measures describing the correspondence between model and log. Directly-follows frequencies can be easily adapted, either per object type or across all object types. $\alpha$-relations build on top of directly-follows relation and can, therefore, also be adapted. The same holds for the $a \Rightarrow_W b$ score of the heuristic miner. Conformance measures are not that easily adaptable, as they need to be applied to object-centric process models. [21] provides an adaptation of replay fitness and escaping edges precision [57] to object-centric event logs and object-centric Petri nets.

*Performance.* The performance perspective deals with measures indicating or influencing the commercial success of the process. Based on the timestamp of events, one can calculate activity durations, durations of whole process executions, customers' waiting times, and more. Significant changes in these measures are almost always crucial for the process owner. Performance measures for object-centric event data have recently been adapted and extended [24]. First, this work adapted the calculation of process execution level performance metric: Throughput times of a process execution are calculated based on the process execution graph not a flattened event sequence of one object type. Second, the graph structure of process executions was leveraged to define new performance metrics on the event level: synchronization time (difference between the first and last object arriving at one event), pooling time (time to collect all objects of one type for one event), and lagging time (difference between the last objects of two object types arriving at one event). Furthermore, the flow time was defined as sum of synchronization time and sojourn time.

*Data.* Most of the features calculated on the data perspective can be trivially adapted to the object-centric setting. However, traditional process mining assumes features on the case level. Features on the object level now replace these. For different object types, different attributes may be available. For calculating aggregations of object features, one would need to specify the object type next to the attribute. See [58] for an extensive discussion on feature calculation based on the data perspective.

*Resources.* The resource perspective is concerned with measuring organizational matters. The primary differentiation between traditional and object-centric event data is the potential involvement of multiple resources in one event. This raises the question of how, e.g., the individual workload should be calculated for multiple involved resources. The workload

Table 4: A collection of feature extraction functions for different perspectives used throughout process mining.

| Control-Flow | Performance | Data | Resources | Objects |
|---|---|---|---|---|
| Directly-follows frequencies [50] | Service times [25] | Aggregation of object attributes | Workload [25] | Number of Objects |
| $\alpha$-relations [50] | Overtime executions [51] | Aggregation of activity attributes | Involved resources | Number of Object Types |
| $\alpha^+$-relations [52] | Execution durations [53] | Number of events or executions | Number of active resources | Object relationships |
| Heuristic Miner's $a \Rightarrow_W b$ score [54] | Activity sojourn time [53] | Threshold exceedings | Aggregation of attribute values | Object attributes |
| Conformance [21] | Waiting time [53] | | Social Network [55] | |
| DECLARE constraints [11] | Synchronization time [24] | | | |
| ... | ... | ... | ... | ... |

could, e.g., be uniformly distributed. Similar questions are present for the calculation of social network features, e.g., the handover of work [55]. The total workload calculation does not need to be adapted.

*Objects.* Features in the object perspective are concerned with encoding information about objects and their relationships. Simple calculations such as the average number of objects per event or executions can be performed. By analyzing the object graph, one can obtain frequent relationships present in the event log. Counting such patterns might yield interesting insights about changes in the process. Furthermore, object attributes can be aggregated. Three new object features have recently been introduced by Galanti et. al [59] to be employed in predictive process monitoring.

#### *4.3.2 Time Series Retrieval*

A feature function is applied to each set of process executions of a segmented log. The resulting values are concatenated in the timely order of the sub logs, forming a times series. This time series describes the development of the specific process measure over time.

**Definition 9** (Time Series) *Let $L = (E, O, \pi_{time}, trace, \pi_{attr})$ be an event log, $ex \in \mathbb{EX}$ an extraction technique, and $P = ex(L)$ a set of process executions, $w \in \mathbb{R}_0^+$ be a window size, $f^{in} \in \mathbb{F}^{in}$ be an inclusion function and $feat \in \mathcal{F}$ be a feature function. $\overrightarrow{s}(P, w, f^{in}, feat) = \langle v_1, \dots, v_m \rangle$ with $v_i = feat(seq(P, w, f^{in})(i))$ is the resulting time series.*

We construct a time series for the exemplary event data of Table 2. We use the segmented log $\langle \{p_{\{o1,i1,i2\}}, p_{\{o2,i3,i4,i5\}}\}, \{p_{\{o3,i6\}}\}, \{p_3\} \rangle$ that was provided earlier. We use a feature function that calculates average number of objects per process execution $avg_{o,exec}(\{p_{O_1}, \dots, p_{O_k}\}) = \frac{1}{k} \cdot \sum_{i \in \{1,\dots,k\}} |O_i|$ The resulting time series is $\overrightarrow{s}(\{p_{\{o1,i1,i2\}}, p_{\{o2,i3,i4,i5\}}, p_{\{o3,i6\}}\}, 7 \text{ days}, f^{in}_{contained}, avg_{o,exec}) = \langle 2.5, 1 \rangle$.

## 5 Concept Drift Detection

In process mining, a plethora of techniques has been applied to detect concept drifts and locate their *change points* in time series constructed from event data of a process. Many of the techniques use hypothesis testing to compare the distribution of values for subsequent time windows [29, 27] or global cost function-based segmentation of the time series [11, 12]. We provide a general definition for a concept drift detection technique.

**Definition 10** (Concept Drift Detection) *Let $s \in \mathbb{R}^*$ be a time series. A concept drift detection technique $cp(s) \subseteq \{1, \dots, len(s)\}$ maps a time series to the points of significant change.*

Figure 4 depicts two examples of possible concept drift detection techniques. Hypothesis testing and cost functions are two popular techniques for detecting concept drifts in time series. In the case of hypothesis testing, two windows of values before and after a reference point are extracted. If their distribution differs significantly, this reference point is

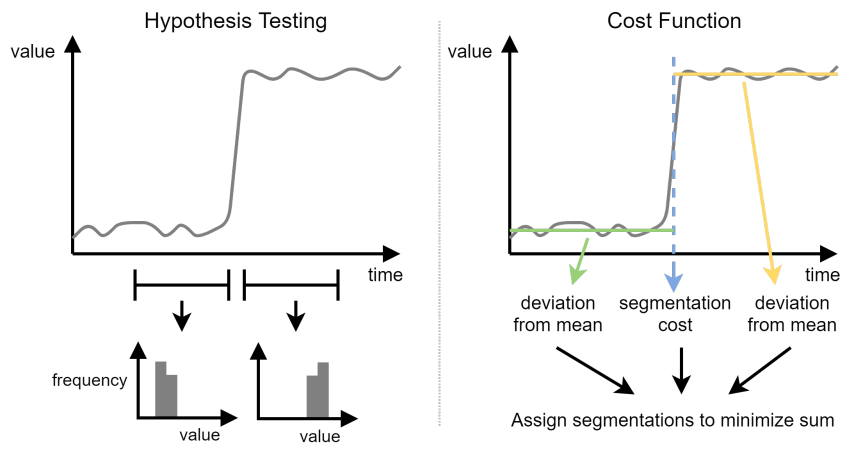

Figure 4: Example of two concept drift detection techniques. Hypothesis testing compares the distribution of values between (subsequent) windows for significant differences. Furthermore, one can minimize a cost function depending on segmentations to find change points.

classified as a change point. This test can be done for each point in time as a reference point. Using a cost function, one can globally assign segmentations to a time series where the borders between segments are change points. These segments are chosen as parameters to a cost minimization problem: The deviations of points from their segment's mean are summed up. Additionally, each segmentation adds a penalty cost to this term, avoiding overfitting. The segmentation with the lowest cost yields the concept drifts. We refer to the papers discussed in related work for further concept drift detection techniques and in-depth discussion.

## 6 Drift Correlations

This section introduces the general formulation of *Granger causality* to determine linear and non-linear correlations between concept drifts. We first define Granger causality and subsequently provide a general definition to test for non-linear relationships. We combine these techniques to our setting to find concept drift correlations.

Granger causality [26] describes a statistical test to determine a weak form of causality between two time series based on linear regression. A time lag between the two time series is considered. First, future values of the first time series are predicted based on its past values. Subsequently, future values of the first time series are predicted based on its past values and the lagged values of the second time series. If the prediction significantly improves, the second time series is considered Granger causal to the first one. Granger causality cannot be considered *true causality* since confounding effects can not be ruled out. However, it is often interpreted as predictive causality. The exact mathematical definition can be found in [26].

**Definition 11** (Granger Causality) *Let $n \in \mathbb{N}$ and $S^n = \{\sigma \in \mathbb{R}^* \mid len(\sigma) = n\}$ be time series of length $n$. Granger-causality* $\textsc{Granger} : S^n \times S^n \times \{1, \dots, n-1\} \to [0, 1]$ *maps two time series on a real value under consideration of a time lag.*

Only linear relationships can be detected using this basic formulation of Granger causality. To test for non-linear relationships, several approaches use kernel functions [48] to transform the original time series [46, 47].

**Definition 12** (Kernel Functions) *Let $s = \langle s_1, \dots s_n \rangle \in \mathbb{R}^*$ be a time series and $\Phi : \mathbb{R} \to \mathbb{R}$ be a kernel function. $\Phi(s) = \langle \Phi(s_1), \dots, \Phi(s_n) \rangle$ is a transformed time series.*

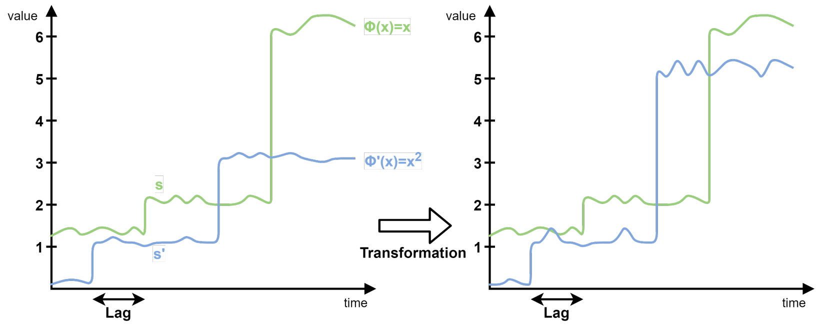

Figure 5: Example of applying a kernel function to transform time series. One time series is transformed using a quadratic kernel, the other time series is not transformed.

We depict the transformation of time series using kernel functions in Figure 5. The changes of series $s$ are correlated to preceding changes in series $s'$ in a quadratic manner. After transforming $s'$ with a quadratic kernel, the transformed time series are correlated linearly. The original formulation of Granger causality can be applied to the transformed time series to detect the quadratic relationship.

We use these two concepts to determine whether two concept drifts of different time series are correlated. We consider a pair of concept drifts from the two time series. The time lag between drifts gives the lag between time series. We test for Granger causality under consideration of a kernel function[5].

**Definition 13** (Concept Drift Correlation) *Let* $\Phi, \Phi' : \mathbb{R} \rightarrow \mathbb{R}$ *be kernel function and* $s = \langle s_1, \dots s_n \rangle \in \mathbb{R}^*$ *and* $s' = \langle s'_1, \dots s'_n \rangle \in \mathbb{R}^*$ *with* $len(s) = len(s')$ *be two time series with change points* $cp(s), cp(s')$*. For a concept drift pair* $(d, d') \in cp(s) \times cp(s')$ *the concept drift correlation is retrieved by* $\textsc{Granger}(\Phi(s), \Phi'(s'), d - d')$*.*

This test can be applied to all pairs of concept drifts between any two time series to find all correlated concept drifts.

[5]Using $\Phi(x) = x$ we can still test for linear relationships

## 7 Framework for Explainable Concept Drift

The previously introduced notations allow us to define our general framework to correlate significant changes in object-centric event data. Several parameters can be chosen, Table 5 depicts an overview of the parameters for different framework steps. We extract the correlated concept drifts called explainable concept drifts based on these parameters. The combinations of each pair of change points in both times series are considered for each pair of features. We test for Granger causality for each change point pair after transforming the time series with the kernel functions. If the p-value of this test is lower than the threshold $p$, the features together with the two change points are added to the set of correlated drift.

**Definition 14** (Explainable Concept Drifts) *Let* $L = (E, O, \pi_{time}, trace, \pi_{attr})$ *be an event log,* $ex \in \mathbb{EX}$ *an extraction technique, and* $P = ex(L)$ *a set of process executions,* $w \in \mathcal{T}$ *be a window size,* $f^{in} \in \mathbb{F}^{in}$ *be an inclusion function,* $f \subseteq \mathcal{F}$ *be a feature set,* $cp$ *be a change point detection algorithm,* $\Phi, \Phi' \in \mathbb{R} \rightarrow \mathbb{R}$ *be kernel functions and* $p \in [0, 1]$ *a probability threshold.* $EXC = \{(feat_i, feat_j, d, d') \in \mathcal{F} \times \mathcal{F} \times \mathbb{N} \times \mathbb{N} \mid feat_i \neq feat_j \wedge s_i = \overrightarrow{s}(P, w, f^{in}, feat_i) \wedge s_j = \overrightarrow{s}(P, w, f^{in}, feat_j) \wedge d \in cp(s_i) \wedge d' \in cp(s_j) \wedge d' < d \wedge \textsc{Granger}(\Phi(s_i), \Phi(s_j), d - d') \leq p\}$ *is the set of explainable concept drifts.*

Table 5: Summary of parameter choices in our framework.

| Main Framework Step | Substep | Parameters | Symbol |
|---|---|---|---|
| **Time Series Extraction** | Execution Extraction | Process Executions | $ex$ |
| | Log Segmentation | Window size | $w$ |
| | | Inclusion function | $f^{in}$ |
| | Feature Calculation | Feature set | $F$ |
| **Drift Detection** | Detection | Detection algorithm | $cp$ |
| **Drift Correlation** | Granger causality | Kernel functions | $\Phi, \Phi'$ |
| | | Probability threshold | $p$ |

### *7.1 Implementation*

We implement our framework in Python. The framework as well as the experiments of Section 8 are publicly available on GitHub[6]. Every figure can be reproduced using the *expriments.py* script. For further instructions, please visit the repository.

The framework can be extended with new features or different concept drift detection techniques. The user can freely choose the kernel function. Our implementation is based on the OCPA[7] library [60].

## 8 Evaluation

This section evaluates our proposed framework. We provide a quantitative evaluation in terms of sensitivity and scalability and, subsequently, showcase our framework in a case study. First, we investigate the sensitivity of the time series extraction to the chosen inclusion function and the window size. Second, we evaluate the scalability of the whole framework depending on the number of features. We will not evaluate the performance of different concept drift detection techniques. The corresponding papers cover these aspects. We employ the Pruned Exact Linear Time (PELT) [61] algorithm for our experiments. The PELT algorithm minimizes a cost-function that strikes a balance between the number of assigned change points and the variance in each resulting segment. The PELT algorithm was already employed for concept drift detection in process mining [9] and has low computation times. Furthermore, we do not evaluate the influence of the $p$-value threshold on the detected explainable concept drifts since data sets providing ground truths for these do not exist yet. Furthermore, we provide a case study by applying our framework and uncovering an explainable concept drift.

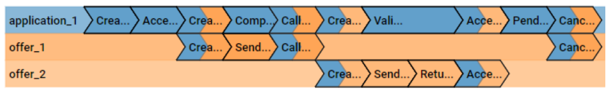

Figure 6: Visualization of a process execution with one application and two offers retrieved from the OC$\pi$ tool.

We use an object-centric event log describing a loan application process [62]. This event log has two object types: *application* and *offer*. A customer submits an application for a loan. Subsequently, this application is answered with one or multiple offers. An offer can be canceled, accepted, or refused. We use the process executions defined by the connected components of the objects graph: Each process execution describes exactly one application and the corresponding offer objects created throughout the application. Figure 6 depicts a visualization for the activities of an exemplary process execution with two offers. The visualization is retrieved from the OC$\pi$ tool [63]. The chosen extraction technique for this visualization and the remainder of this evaluation is $ex_{comp}$.

### *8.1 Time Series Extraction Sensitivity*

In this section, we discuss the influence of different parameters on the extracted time series. First, we investigate the impact of the inclusion function for features calculated on the execution and event level. Subsequently, we depict the influence of the window

[6]`https://github.com/niklasadams/ex_concept_drift`.
[7]`https://github.com/ocpm/ocpa`

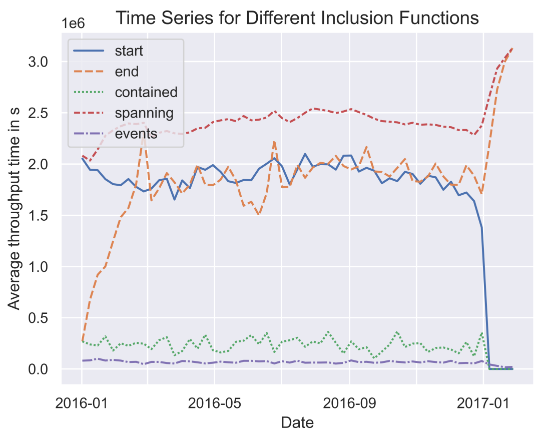

Figure 7: Time series of the average throughput time of a process execution for different inclusion functions. The underlying window size is one week, i.e., 7 days.

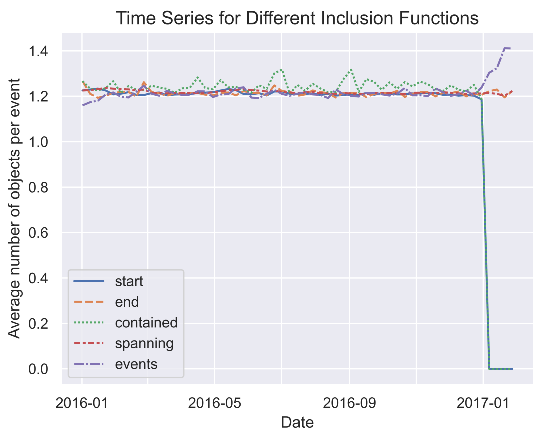

Figure 8: Time series of the average number of objects per event for different inclusion functions. The underlying window size is 7 days.

size on the extracted time series.

*Inclusion Function.* We extracted time series for the same features and the same window size for different inclusion functions to depict the influence of the inclusion function on the retrieved time series. We can generally calculate features on the process execution level, e.g., the throughput time, and on the event level, e.g., the number of objects per event. We take one feature of both categories and extract time series with all inclusion functions.

Figure 7 depicts the time series for the average execution time of a process execution, i.e., the time difference between the first and last event. We can observe significant differences between the retrieved time series. Using the event-based inclusion function (cf. Definition 6, Figure 3), process executions are split into smaller ones, significantly impacting features calculated based on process executions, e.g., the throughput time. If possible, the event-based inclusion function should not be used to calculate process execution-based features. The remaining four inclusion functions can be grouped into two groups: Exact decompositions (start and end) and subset generators (contained and spanning). Exact decomposition inclusion functions assign each execution to exactly one window. Subset generators can assign executions to an arbitrary amount of windows, including zero.

The time series for the contained inclusion function (cf. Definition 6, Figure 3) exhibits low values, as only executions with lower throughput times than the window size can be included. Others are discarded. The time series of the spanning inclusion function exhibits high levels as process executions spanning relatively many windows increase the average of each of those.

The start and end inclusion functions show similar behavior. However, both have effects on the resulting time series. Start leads to empty windows if no process execution starts within this time frame. For demonstration purposes, these empty windows are depicted by zero values in Figure 7. Only long-lasting process executions are completed in the latest time frame, shown by the increase in the time series given by the end inclusion function. In conclusion, when interpreting results, one should be aware of the chosen inclusion function. Furthermore, the event-based inclusion function should not be used on process execution features as it might lead to a misleading time series.

Figure 8 depicts the time series retrieved for the average number of objects per event. Two observa-

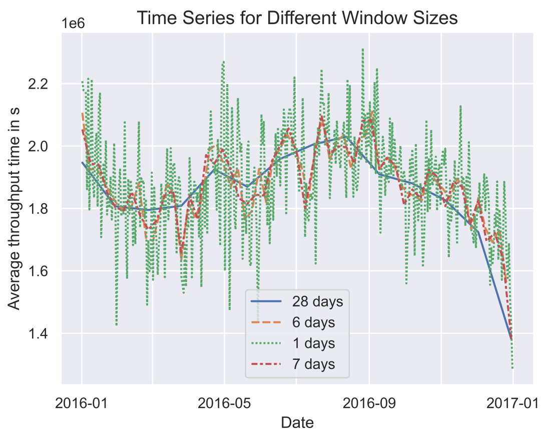

Figure 9: Time series of the average throughput time retrieved for different window sizes. The inclusion function is start. Empty windows are discarded.

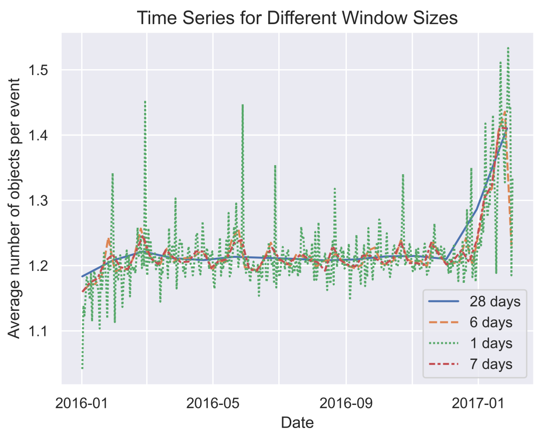

Figure 10: Time Series of the average number of objects per event retrieved for different window sizes. The inclusion function is events.

tions can be made: Two inclusion functions (start and contained) can not obtain any event values for windows where no process execution starts. Furthermore, the event-based inclusion function shows an increase in the average number of objects towards the end, a dynamic that the remaining two inclusion functions cannot capture. Therefore, the default inclusion function for event-based features should be the event-based inclusion function.

In general, the event-based inclusion function should be used for event-based features, and process execution-based inclusion functions should be used for process execution-based features. The start and end inclusion functions seem to give the most accurate measures. However, some postprocessing to remove empty parts might be necessary. When removing empty parts, it might be necessary to post-process time series of other features to ensure equal time series length.

*Window Size.* In this section, we investigate the influence of the window size choice on the retrieved time series. Again, we use the average throughput time and the average number of objects per event as features. We use the start inclusion function.

Figure 9 depicts the retrieved time series for the average throughput time of process executions starting within that time frame; Figure 10 depicts the retrieved time series for the average number of objects per event for all events within a time frame. We can observe a high variance for time series extracted with window size one day. A daily window size calculates the feature for every day of the year, including weekends and holidays. This leads to highly variable time series, not due to concept drifts but due to the nature of working weeks and holidays. Choosing a different window size can correct some of these problems. Choosing a windows size of a multiple of seven days will lead to an equal distribution of weekends over the windows. Choosing higher multiples of one week, e.g., 28 days, leads to a smoother time series averaging out some effects. However, it also reduces the number of observations and eliminates short-term dynamics. This trade-off that has to be determined by the user.

### *8.2 Framework Scalability*

In this section, we investigate the scalability of our framework. The user can select an arbitrary set of features to be investigated for correlated concept drifts. Each of the time series of these features is investigated for concept drifts, leading to a linear dependency of the concept drift detection time to the

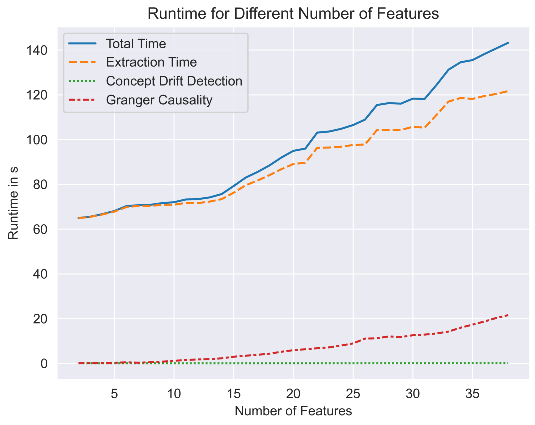

Figure 11: Decomposition of the framework's running time for different numbers of features. With an increasing number of features the Granger causality check takes a greater relative share of computation time.

number of features. However, all of the time series are tested for correlated concept drift to *all* other time series. The Granger causality testing converges to a quadratic runtime with an increasing number of features. For this reason, we inspect the scalability of our framework depending on the number of features in this section. We segment the log based on a window size of seven days and the start inclusion function. To simulate the worst-case scenario of a large number of concept drifts demanding a large number of tests, we use the PELT algorithm with a low penalty value of 0.1 to detect concept drifts. We test for linear relationships. We calculate 38 trivial features for the feature space. Figure 11 depicts the results. We can observe increasing running times of the overall framework with an increasing number of features. The increasing running times are primarily due to the time series extraction and the Granger causality checking. Time series extraction takes up a majority of the computation time for multiple reasons: First, the extraction includes subdividing the event log into sublogs. This includes copying data, and recomputing concepts such as process executions. Second, the features have to be computed on the basis of each event log. Since the event logs contain many events, these are expensive operations. While the time series extraction deals with hundreds of thousands of data points, it boils these data points to only a few. Therefore, subsequent steps have much lower input sizes and therefore much lower computation times. The concept drift detection does not show any notable share of the computation time. The Granger causality checking takes an increasingly large share of the computation time for an increasing number of features. However, for the number of features tested, Granger causality testing never accounts for the majority of computation time. Therefore it does not pose a threat to the applicability of our framework for feature spaces of this size.

### *8.3 Case Study*

We apply our framework to the object-centric event log of the loan application event log to test for the presence of explainable concept drifts. Resource workload and service times (i.e. a resource's time spent actively working on one activity) have shown to frequently exhibit relationships [25]. Therefore, we choose the features of workload (active process executions) and service times to extract time series. According to the previously discussed experiments, a window size of one week provides some smoothing of day-to-day dynamics but captures long-term dynamics and cancels out weekends. Therefore, we extract the time series with weekly windows. The number of ongoing process executions is captured by counting the number of different process executions that events within one week are associated with. Furthermore, we are interested in retrieving the average service times of an activity over the course of a specific week. For these two reasons, we choose the event-based inclusion function. If we would choose any process execution based inclusion function, events actually occurring in other windows might be counted for another window if the corresponding process execution starts/end within this window.

We depict the retrieved result in Table 6. Applying the PELT concept drift detection algorithm we retrieve the provided set of concept drifts. Each pair is checked for Granger-causality given the lag provided by the time difference between concept drifts. If the time difference between drifts is too large,

Table 6: Detected concept drifts, checked concept drift pairs and the results of applying our framework to the loan application event log. When the drift lag is too large there are not sufficient observations to test for Granger-causality (NA). We uncover one explainable concept drift between a workload drift in week 20 and a service times drift in week 25.

| Feature | Concept Drifts (Week) | Checked Pairs | Results |
|---|---|---|---|
| Workload | 24 | (24,29) | $p = 0.0075$ |
| | | (24,49) | NA |
| | 49 | $\emptyset$ | $\emptyset$ |
| Service times (W_Validate application) | 19 | (19,24) | $p = 0.7517$ |
| | | (19,49) | NA |
| | 24 | (24,49) | NA |
| | 25 | (24,49) | NA |
| | 49 | $\emptyset$ | $\emptyset$ |

there might not be sufficient observations to test for Granger-causality. Therefore, some pairs are ruled out. Two pairs are tested for Granger-causality: Whether the drift in workload is Granger-causal to drift in service times 5 weeks later and vice versa. While the p-value for a Granger-causality for a service times drift causing a workload drift is very high (0.7517, i.e., very unlikely), the Granger-causality for a workload drift causing a service times drift 5 weeks later is very low (0.0075, i.e., very likely). Therefore, we uncovered an explainable concept drift from workload to service times. This is confirming the existence of the explainable concept already uncovered in [12], where an increase in workload led to a decrease in service times.

## 9 Conclusion

In this paper, we introduced a framework to uncover explainable concept drifts in object-centric event data. Our framework is split into three steps. First, time series for different features are extracted from an objct-centric event log and, second, investigated for concept drifts. Third, we test for correlations between concept drifts using Granger causality. Through the use of kernel functions, we can test for non-linear relationships. In our evaluation, we investigated the choice of different parameters to the retrieved time series and the framework's scalability. The inclusion function and window size significantly affect the retrieved time series. The inclusion function should be chosen in accordance with the feature. Selecting a window size of multiples of seven days corrects the time series for irregularities like weekends and provides a smoothing of the time series. Furthermore, we provide a case study uncovering an explainable concept drift where increased workload led to reduced service times. In conclusion, we provide a general framework to extract time series and correlate concept drifts to retrieve insights about explanations for significant changes in a process.

The general approach of correlating concept drifts in different perspectives with each other might also be of interest for other application areas that experience concept drifts, e.g., data streams. Furthermore, by dropping the concept drift detection step and exhaustively testing through all lags for all feature pairs, one can find arbitrary linear and non-linear relationships between different process features. It is possible to extend the framework into multiple directions is possible: First, one could incorporate the support of multivariate time series for checking Granger causality. Second, $n$-wise causality relationships should be supported as some drifts may only be enabled by $n$ preoccuring changes. Third, learning kernel functions to provide an automated kernel function choice could be a significant support for applying the framework in large scale.

## Funding

We thank the Alexander von Humboldt (AvH) Stiftung for supporting our research (grant no. 1191945).

## References

[1] M. Dumas, M. L. Rosa, J. Mendling, H. A. Reijers, Fundamentals of Business Process Management, Second Edition, Springer, 2018. doi:10.1007/978-3-662-56509-4.

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

[3] S. J. J. Leemans, D. Fahland, W. M. P. van der Aalst, Discovering block-structured process models from event logs - A constructive approach, in: PETRI NETS, Springer, 2013. doi:10.1007/978-3-642-38697-8_17.

[4] A. Adriansyah, B. F. van Dongen, W. M. P. van der Aalst, Conformance checking using cost-based fitness analysis, in: EDOC, IEEE, 2011, pp. 55–64. doi:10.1109/EDOC.2011.12.

[5] M. Camargo, M. Dumas, O. González, Automated discovery of business process simulation models from event logs, Decis. Support Syst. 134 (2020).

[6] N. Tax, I. Verenich, M. L. Rosa, M. Dumas, Predictive business process monitoring with LSTM neural networks, in: CAiSE, Springer, 2017, pp. 477–492. doi:10.1007/978-3-319-59536-8_30.

[7] R. P. J. C. Bose, W. M. P. van der Aalst, I. Žliobaitė, M. Pechenizkiy, Dealing with concept drifts in process mining, IEEE Trans. Neural Networks Learn. Syst. 25 (2014) 154–171.

[8] T. Brockhoff, M. S. Uysal, W. M. P. van der Aalst, Time-aware concept drift detection using the earth mover's distance, in: ICPM, 2020, pp. 33–40. doi:10.1109/ICPM49681.2020.00016.

[9] A. Yeshchenko, C. Di Ciccio, J. Mendling, A. Polyvyanyy, Visual drift detection for sequence data analysis of business processes, IEEE Transactions on Visualization and Computer Graphics (2021).

[10] A. E. M. Chamorro, I. A. Nepomuceno-Chamorro, M. Resinas, A. Ruiz-Cortés, Updating prediction models for predictive process monitoring, in: CAiSE, volume 13295, Springer, 2022, pp. 304–318. doi:10.1007/978-3-031-07472-1_18.

[11] A. Yeshchenko, C. D. Ciccio, J. Mendling, A. Polyvyanyy, Comprehensive process drift detection with visual analytics, in: ER, 2019, pp. 119–135. doi:10.1007/978-3-030-33223-5_11.

[12] J. N. Adams, S. J. van Zelst, L. Quack, K. Hausmann, W. M. P. van der Aalst, T. Rose, A framework for explainable concept drift detection in process mining, in: BPM, Springer, 2021, pp. 400–416. doi:10.1007/978-3-030-85469-0_25.

[13] D. M. V. Sato, S. C. D. Freitas, J. P. Barddal, E. E. Scalabrin, A survey on concept drift in process mining, ACM Comput. Surv. 54 (2022) 189:1–189:38.

[14] S. Aminikhanghahi, D. Cook, A survey of methods for time series change point detection, Knowl. Inf. Syst. 51 (2017) 339–367.

[15] A. Ostovar, A. Maaradji, M. L. Rosa, A. H. M. ter Hofstede, Characterizing drift from event streams of business processes, in: CAiSE, Springer, 2017, pp. 210–228. doi:10.1007/978-3-319-59536-8_14.

[16] W. M. P. van der Aalst, et al., Process mining manifesto, in: BPM Workshops, Springer, 2011, pp. 169–194. doi:10.1007/978-3-642-28108-2_19.

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

[18] W. M. P. van der Aalst, A. Berti, Discovering object-centric Petri nets, Fundam. Informaticae 175 (2020) 1–40.

[19] S. Esser, D. Fahland, Multi-dimensional event data in graph databases, J. Data Semant. 10 (2021) 109–141.

[20] P. Waibel, L. Pfahlsberger, K. Revoredo, J. Mendling, Causal process mining from relational databases with domain knowledge, CoRR abs/2202.08314 (2022).

[21] J. N. Adams, W. M. P. van der Aalst, Precision and fitness in object-centric process mining, in: ICPM, IEEE, 2021, pp. 128–135. doi:10.1109/ICPM53251.2021.9576886.

[22] D. Fahland, Process mining over multiple behavioral dimensions with event knowledge graphs, in: Process Mining Handbook, volume 448 of *Lecture Notes in Business Information Processing*, Springer, 2022, pp. 274–319. doi:10.1007/978-3-031-08848-3_9.

[23] J. N. Adams, D. Schuster, S. Schmitz, G. Schuh, W. M. P. van der Aalst, Defining Cases and Variants for Object-Centric Event Data, IEEE, 2022, pp. 128–135. doi:10.1109/ICPM57379.2022.9980730.

[24] G. Park, J. N. Adams, W. M. P. van der Aalst, OPerA: Object-centric performance analysis, in: ER, Springer, 2022, pp. 281–292. doi:10.1007/978-3-031-17995-2_20.

[25] J. Nakatumba, W. M. P. van der Aalst, Analyzing resource behavior using process mining, in: BPM, 2009, pp. 69–80. doi:10.1007/978-3-642-12186-9_8.

[26] C. W. Granger, Investigating causal relations by econometric models and cross-spectral methods, Econometrica: Journal of the Econometric Society (1969) 424–438.

[27] R. P. J. C. Bose, W. M. P. van der Aalst, I. Zliobaite, M. Pechenizkiy, Handling concept drift in process mining, in: CAiSE, Springer, 2011, pp. 391–405. doi:10.1007/978-3-642-21640-4_30.

[28] J. Martjushev, R. P. J. C. Bose, W. M. P. van der Aalst, Change point detection and dealing with gradual and multi-order dynamics in process mining, in: BIR, Springer, 2015, pp. 161–178. doi:10.1007/978-3-319-21915-8_11.

[29] A. Maaradji, M. Dumas, M. L. Rosa, A. Ostovar, Fast and accurate business process drift detection, in: BPM, 2015, pp. 406–422. doi:10.1007/978-3-319-23063-4_27.

[30] R. Accorsi, T. Stocker, Discovering workflow changes with time-based trace clustering, in: SIMPDA, Springer, 2011, pp. 154–168.

[31] P. Weber, B. Bordbar, P. Tiño, Real-time detection of process change using process mining, in: ICCSW, Imperial College London, 2011, pp. 108–114.

[32] J. Carmona, R. Gavaldà, Online techniques for dealing with concept drift in process mining, in: IDA, 2012, pp. 90–102. doi:10.1007/978-3-642-34156-4_10.

[33] C. Zheng, L. Wen, J. Wang, Detecting process concept drifts from event logs, in: OTM, Springer, 2017, pp. 524–542. doi:10.1007/978-3-319-69462-7_33.

[34] A. Maaradji, M. Dumas, M. L. Rosa, A. Ostovar, Detecting sudden and gradual drifts in business processes from execution traces, IEEE Trans. Knowl. Data Eng. 29 (2017) 2140–2154.

[35] S. B. Junior, G. M. Tavares, V. G. T. da Costa, P. Ceravolo, E. Damiani, A framework for human-in-the-loop monitoring of concept-drift detection in event log stream, in: WWW, ACM, 2018, pp. 319–326. doi:10.1145/3184558.3186343.

[36] M. Hassani, Concept drift detection of event streams using an adaptive window, in: ECMS, European Council for Modeling and Simulation, 2019, pp. 230–239. doi:10.7148/2019-0230.

[37] L. Lin, L. Wen, L. Lin, J. Pei, H. Yang, LCDD: Detecting business process drifts based on local completeness, IEEE Transactions on Services Computing (2020).

[38] B. Hompes, J. C. A. M. Buijs, W. M. P. van der Aalst, P. M. Dixit, H. Buurman, Detecting change in processes using comparative trace clustering, in: SIMPDA, 2015, pp. 95–108.

[39] A. Seeliger, T. Nolle, M. Mühlhäuser, Detecting concept drift in processes using graph metrics on process graphs, in: S-BPM ONE, 2017, p. 6.

[40] A. Koschmider, D. S. V. Moreira, Change detection in event logs by clustering, in: OTM, Springer, 2018, pp. 643–660. doi:`10.1007/978-3-030-02610-3\_36`.

[41] A. F. Ghahfarokhi, G. Park, A. Berti, W. M. P. van der Aalst, OCEL: A standard for object-centric event logs, in: ADBIS, Springer, 2021, pp. 169–175. doi:`10.1007/978-3-030-85082-1\_16`.

[42] M. Ding, Y. Chen, S. L. Bressler, Granger causality: basic theory and application to neuroscience, Handbook of time series analysis: recent theoretical developments and applications (2006) 437–460.

[43] C. Hiemstra, J. D. Jones, Testing for linear and nonlinear granger causality in the stock price-volume relation, The Journal of Finance 49 (1994) 1639–1664.

[44] J. R. Freeman, Granger causality and the times series analysis of political relationships, American Journal of Political Science (1983) 327–358.

[45] K. J. Blinowska, R. Kuś, M. Kamiński, Granger causality and information flow in multivariate processes, Physical Review E 70 (2004) 050902.

[46] Y. Chen, G. Rangarajan, J. Feng, M. Ding, Analyzing multiple nonlinear time series with extended granger causality, Physics letters A 324 (2004) 26–35.

[47] A. Wismüller, A. M. Dsouza, M. A. Vosoughi, A. Abidin, Large-scale nonlinear granger causality for inferring directed dependence from short multivariate time-series data, Scientific reports 11 (2021) 1–11.

[48] B. Schölkopf, A. J. Smola, Learning with Kernels: support vector machines, regularization, optimization, and beyond, Adaptive computation and machine learning series, MIT Press, 2002.

[49] D. Marinazzo, W. Liao, H. Chen, S. Stramaglia, Nonlinear connectivity by granger causality, Neuroimage 58 (2011) 330–338.

[50] W. M. P. van der Aalst, T. Weijters, L. Maruster, Workflow mining: Discovering process models from event logs, IEEE Trans. Knowl. Data Eng. 16 (2004) 1128–1142.

[51] S. Suriadi, C. Ouyang, W. M. P. van der Aalst, A. H. M. ter Hofstede, Root cause analysis with enriched process logs, in: BPM, Springer, 2012, pp. 174–186. doi:`10.1007/978-3-642-36285-9\_18`.

[52] A. K. A. de Medeiros, B. F. van Dongen, W. M. P. van der Aalst, A. J. M. M. Weijters, Process mining : Extending the $\alpha$-algorithm to mine short loops, in: BETA Working Paper Series, volume WP 113, 2004.

[53] B. F. A. Hompes, A. Maaradji, M. L. Rosa, M. Dumas, J. C. A. M. Buijs, W. M. P. van der Aalst, Discovering causal factors explaining business process performance variation, in: CAiSE 2017, Springer, 2017, pp. 177–192. doi:`10.1007/978-3-319-59536-8\_12`.

[54] A. Weijters, W. M. P. van der Aalst, A. Medeiros, Process mining with the heuristics miner-algorithm, CIRP Annals-Manufacturing Technology 166 (2006).

[55] W. M. P. van der Aalst, H. A. Reijers, M. Song, Discovering social networks from event logs, Comput. Support. Cooperative Work. 14 (2005) 549–593.

[56] J. N. Adams, G. Park, S. Levich, D. Schuster, W. M. P. van der Aalst, A Framework for Extracting and Encoding Features from Object-Centric Event Data, Springer, 2022, pp. 36–53. doi:10.1007/978-3-031-20984-0_3.

[57] J. Munoz-Gama, J. Carmona, A fresh look at precision in process conformance, in: BPM, Springer, 2010, pp. 211–226.

[58] M. de Leoni, W. M. P. van der Aalst, M. Dees, A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs, Inf. Syst. 56 (2016) 235–257.

[59] R. Galanti, M. de Leoni, N. Navarin, A. Marazzi, Object-centric process predictive analytics, Expert Syst. Appl. 213 (2023) 119173.

[60] J. N. Adams, G. Park, W. M. van der Aalst, ocpa: A python library for object-centric process analysis, Software Impacts 14 (2022) 100438.

[61] D. Gachomo, The power of the pruned exact linear time (PELT) test in multiple changepoint detection, American Journal of Theoretical and Applied Statistics 4 (2015) 581.

[62] B. van Dongen, BPI Challenge 2017 (2017).

[63] J. N. Adams, W. M. P. van der Aalst, Oc$\pi$: Object-centric process insights, in: PETRI NETS, volume 13288, Springer, 2022, pp. 139–150. doi:10.1007/978-3-031-06653-5_8.