Detect & Conquer: Template-Based Analysis of Processes using Complex Event Processing

Christian Imenkamp[1][0009-0007-4295-1268], Samira Akili[2][0000-0002-9852-7489], Matthias Weidlich[2][0000-0003-3325-7227], and Agnes Koschmider[1][0000-0001-8206-7636]

[1] Business Informatics and Process Analytics, University of Bayreuth, Germany
`{christian.imenkamp|agnes.koschmider}@uni-bayreuth.de`
`https://www.pa.uni-bayreuth.de/en/`

[2] Department of Computer Science, Humboldt-Universität zu Berlin, Germany
`{akilsami|matthias.weidlich}@hu-berlin.de`

Abstract. Online process analysis aims at identifying behavioral regularities or abnormalities in processes in near-real-time from continuous event streams. Yet, its realization is challenging, due to the requirements in terms of scalability and accuracy imposed by processes in Internet-of-Things environments. Against this background, this paper presents an approach for online process analysis that is based on standard models and systems for complex event processing (CEP). We present the "Detect and Conquer" approach that includes generic process templates to accurately capture behavioral regularities or deviations, which are then mapped to CEP queries to achieve their efficient evaluation. We evaluated our approach against synthetic and real-world datasets. The results demonstrate the feasibility and efficiency of our approach.

Keywords: Process Querying · Complex Event Processing · Control-Flow Patterns · Event Stream Processing

1 Introduction

With the rise of the Internet-of-Things (IoT), the volume of data that can be exploited for process analysis has increased significantly. Specifically, IoT environments include sensor-based systems that produce continuous streams of data [11]. While such a setting provides unique opportunities for online process analysis [9], it also imposes challenges in terms of accuracy, i.e., how to identify behavioral regularities or abnormalities, and in terms of scalability, i.e., how to scale the analysis to high-velocity data streams.

Most existing techniques for online process analysis are based on approaches that have originally been developed for static data and subsequently been lifted to online settings [17, 16]. However, these techniques are tailored for a specific analysis task, and do not provide a generic mechanism that may be instantiated for a wide range of analysis needs. In addition, their execution over streams of data requires dedicated optimizations to achieve scalability, instead of employing technical infrastructures for efficient stream processing.

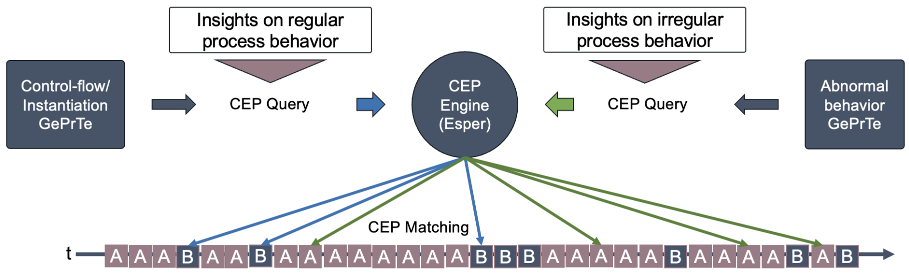

Insights on regular process behavior
Insights on irregular process behavior
Control-flow/ Instantiation GePrTe
CEP Query
CEP Engine (Esper)
CEP Query
Abnormal behavior GePrTe
CEP Matching
t

Fig. 1. Conceptual visualization of the "Detect and Conquer" approach.

In this paper, we outline the "Detect and Conquer" approach as a generic mechanism for accurate and scalable online process analysis. In essence, our idea is to facilitate accurate online process analysis through a collection of *generic process templates* that capture behavioral dependencies that signal both, the regular progress of process execution as well as abnormal behavior. Once these templates have been instantiated for a specific process, they are translated into queries for complex event processing (CEP). This way, standard models and systems for CEP, which have been designed for high-throughput, low-latency processing of event streams, can be leveraged for online process analysis.

Figure 1 provides a conceptual overview of our approach. It operates over a stream of events that are emitted continuously (bottom). Then, user-defined patterns that capture behavioral dependencies between activities as well as abnormal behavior are mapped onto a set of generic process templates (GePrTe). The instantiated templates are then translated into CEP queries, which are processed by a CEP engine.

To realize this vision, this paper introduces 17 generic process templates and their mapping to CEP queries. The templates correspond to common behavioral dependencies of process activities, e.g., related to process instantiation, basic control-flow patterns, and behavioral anomalies. To map these templates onto CEP queries, we introduce a conceptual model that aligns the most important concepts of either area. Based thereon, we operationalize the mapping and formulate the CEP queries in the Esper Query Language (EPL), i.e., the query language of Esper, a generic, open-source CEP engine.

We evaluated our approach using synthetic data and the Sepsis event log. The results highlight the general feasibility of our approach and illustrate its runtime efficiency in terms of latency.

The remainder is structured as follows. Section 2 introduces our template-based approach for online process analysis. The evaluation results are given in Section 3. Section 4 reviews related work, before Section 5 concludes the paper.

2 The Detect and Conquer Approach

This section presents our "Detect and Conquer" approach to online process analysis. We first introduce our model for generic process templates, before turning to their translation into CEP queries.

2.1 The Notion of a Generic Process Template (GePrTe)

A generic process template (GePrTe) captures the essence of common behavioral dependencies between process activities. It is wrapped in a query that is formulated using the Esper Query Language (EPL), as follows. The `SELECT` clause captures the input, while the `FROM` clause contains the actual template definition. In addition, a template must be assigned a name.

```
<independent_templates>

@Name(<template_name>)
SELECT *
FROM PATTERN [
    EVERY ( <event_definition> )
]
.WIN:TIME(<time_window>)
```

The `<template_name>` corresponds to one of the 17 generic process templates (e.g., SingleEventTrigger), as extracted from the literature and summarized in Table 1. The `<time_window>` defines the temporal context, e.g., in terms of the duration of a sliding window (e.g., '60 sec', '2 hours', or '2 days 3 hours').

A query might require the output of another query as input. Therefore, it must be possible to define an order of execution. Unless mentioned explicitly, by a `<independent_templates>` statement, multiple queries may be defined and their execution order follows from the order of their definition.

The actual template definition, captured as `<event_definition>` is given in terms of the following scheme:

```
(
<event_id> = <event_class_ref>(<event_attributes>)
<operator> <negation> <predicate>

<operator> <event_definition>
)
```

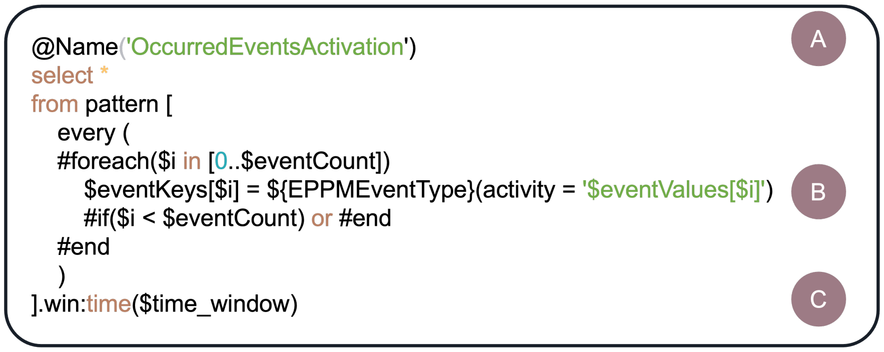
Here, the `<event_id>` is a unique identifier for the respective event. Moreover, `<event_class_ref>` is a reference to the class that defines a single event in the stream, thereby linking the template to a particular schema of the stream. `<event_attributes>`, in turn, defines the specific event that should be matched in the query. `<operator>` captures the behavioral relation between the events/predicates, with examples being conjunction ('AND'), disjunction ('OR'), and sequencing ('->'), while `<negation>` potentially negates a statement (i.e., using 'NOT'). In addition, a `<predicate>` is defined through the following scheme:

```
(
    <event_id>.<event_attributes>
        <comparison_operator>
    <event_id>.<event_attributes>
) <operator>
<predicate>
```

Here, the `<comparison_operator>` is used to compare the attribute values of two events, e.g., using '<', '>', '=', or '!='.

Table 1 shows the set of generic process templates that we implemented as part of our approach. They cover various behavioral dependencies that capture regular process execution in terms of its basic control-flow and instantiation, as well as abnormal process execution.

Next, exemplify `occurred event`, one of the generic process templates. It is shown in Fig. 2 and, once instantiated, can be used to discover specific events that trigger the instantiation of a process.



```
@Name('OccurredEventsActivation')
select *
from pattern [
    every (
    #foreach($i in [0..$eventCount])
        $eventKeys[$i] = ${EPPMEventType}(activity = '$eventValues[$i]')
        #if($i < $eventCount) or #end
    #end
    )
].win:time($time_window)
```

Fig. 2. Example template that triggers when a specific event is observed

The template consists of three parts: (A) the pattern name and core EPL logic; (B) the template engine syntax, which converts the GePrTe into a CEP query; and (C) the time window, which allows the user to define the duration for the sliding window. In (B), the template engine uses a *#foreach* loop to create assignments for the events, based on user input. Additionally, the *#if* condition appends an 'or' operator between event assignments, except for the last one.

Next, we define how to map the generic process templates onto CEP queries.

2.2 Mapping GePrTe onto CEP queries

The mapping relies on the Esper Query Language (EPL). The CEP queries are written in EPL syntax accordingly. The following query illustrates the EPL syntax definition for the pattern that an event of type "A" is followed by event of type "B" within 5 minutes:

Table 1. List of generic process templates

Template Name	**Description**
	Basic Control-Flow
Exclusive Choice [1]	One of several possible paths is selected
Parallel Split [1]	A single path branches into multiple parallel paths
Simple Merge [1]	Merges multiple branches into a single branch
Synchronization [1]	Synchronize multiple parallel branches into one
	Process Instantiation
Occurred Events (conditional) [6]	Triggers a process when specific events happen
Single Event Trigger [6]	Starts a process upon a single event.
Multi Event Trigger [6]	Requires multiple events to start a process, using all events at once
All Subscriptions [6]	Creates event subscriptions for all non-triggered start events
No Subscriptions [6]	No event subscriptions are created for the process instance
Reachable Subscription [6]	Activates event subscriptions only for necessary process completion
Until Consumption [6]	Subscriptions remain active until the relevant event is consumed
Event-based Unsubscription [6]	Cancels remaining subscriptions after one of several events occurs
	Abnormal Behavior
Deadlock [8]	An exclusive choice followed by a parallel merge
Infinite Loop [8]	A loop begins with an and join and ends with a XOR split
Missing Events [12]	Triggering an event that requires follow-up actions but no subsequent event occurs
Unexpected Sequence [12]	A sequence of events that should occur
Unattended Decision Points [12]	Only positive outcomes are logged, and failures are unaddressed

```
SELECT * FROM PATTERN [every a=A -> b=B(a.end + 5 min >= b.start)]
```

We defined EPL queries that correspond to the discovery of process tasks like finding behavior dependencies. Table 2 illustrates this. For example, the first row defines the EPL syntax to find events that follow each other. This corresponds to the control-flow pattern sequence. The second row defines the EPL syntax to detect parallel events corresponding to control-flow patterns Parallel Split or Synchronization.

Table 2. Mapping of Esper EPL Syntax to Process Discovery Tasks

(Esper) EPL Syntax	**Task**
`PATTERN [every a=EventA -> b=EventB]`	Discovery of sequence activities [5]
`PATTERN [every (a=EventA AND b=EventB)]`	Discovery of parallel activities [5]
`PATTERN [every (EventA and not EventB or EventB and not EventA)]`	Discovery of alternative activities [5]
`SELECT COUNT(*) FROM EventA`	Discovery of frequent activities [5]
`WHERE EventA.timestamp < EventB.timestamp`	Discovery of temporal dependencies [5]
`WHERE EventA.activity = 'Approval'`	Filtering of relevant cases [5]
`PATTERN [every a=EventA -> (b=EventB AND c=EventC)]`	Discovery of complex control-flow patterns [5]

To clarify the correspondence of concepts from the domain of CEP and process mining, we defined a conceptual model. Each entity in the model is either attributed to the CEP, PM or has a meaning in both contexts. While the notion of an *event* is central to our model, its meaning is defined on a different level of abstraction in the context of CEP and PM. That is, for CEP an event stands for the change of a state or the occurrence of a situation of interest within some system. For PM, an event denotes a recorded activity of a business process. As such, it comes with an *activity* and *case* identifiers, as well as a *timestamp*. The former attributes define a certain event class from the perspective of CEP based on which arbitrary *patterns* can be constructed, based on CEP *operators*. Those patterns again, can be used to encode temporal behavior among certain activities. Matches of such patterns over some *event stream* are referred to as *complex events* and consequently, correspond to the instantiation of control flows within process models.

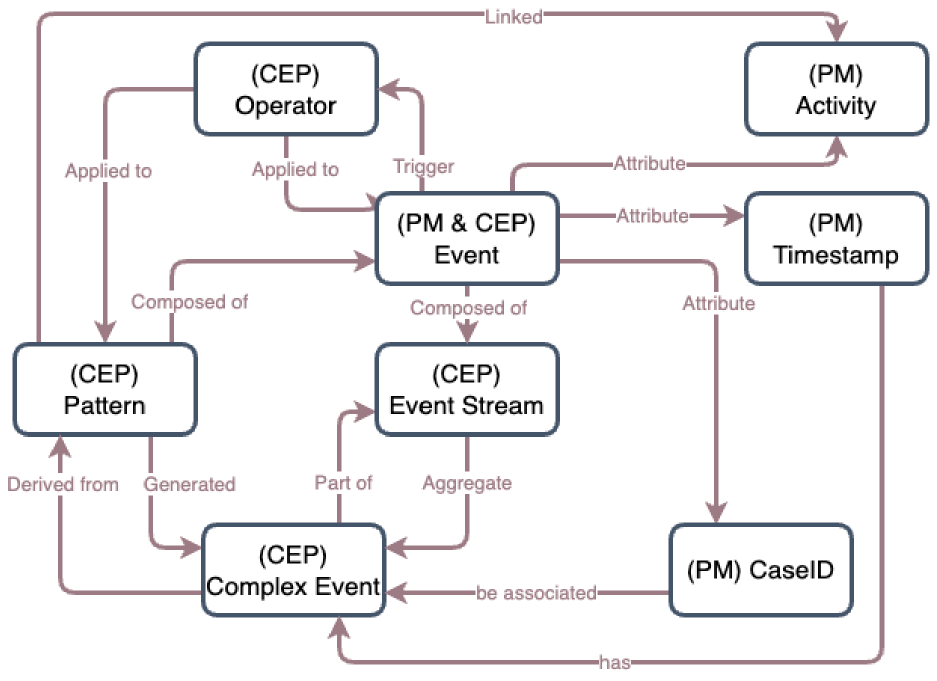


Fig. 3. Conceptual model of CEP query constructs and the inclusion of event log attributes.

3 Evaluation

This section summarizes the evaluation results. We implemented a prototype in Java [3]. The implementation integrates Esper for event processing and analysis [4]. We applied our approach on publicly accessible event logs and synthetically generated event logs of different variety in terms of variability, length of traces. Table 3 summarizes the properties of the event logs that we used for evaluation. The implementation can be found in a publicly available repository [5].

The evaluation results and queries for the synthetic event logs are summarized in (Section 3.1). Next, we applied our approach on the Sepsis event log and used the online heuristics miner to evaluate the discovery result, see (Section 3.2). Due to page limitations, we only present an excerpt of results. However, all results (e.g., queries) can be found in the repository along the prototype implementation. Finally, we evaluated the runtime efficiency (Section 3.3)

[3] `https://www.java.com/en/`
[4] `https://github.com/espertechinc/esper`
[5] `https://github.com/chimenkamp/detect-and-conquer---Esper`

Table 3. Synthetic and real-world event logs

Log Name	Mean Trace Length	Number of Activities	Number of Events
Deadlock, Loop, and Conditions (Synthetic)	7.12	19	71212
Subscriptions and Choices (Synthetic)	24.55	29	1227635
Sepsis Cases [15]	14.48	16	15214

3.1 Generation of Different Queries and the Application to an Event Stream

This section summarizes the results of the generated queries and shows how we matched them on the event stream. The following query aims to identify a deadlock (i.e., and AND join is used for synchronization although a XOR has been used for branching). The query consists of three sub-queries for control-flow detection (i.e., "ExclusiveChoiceAsStreamactivity_2", "ExclusiveChoiceAsStreamactivity_1" and "ParallelMergeAsStream") and one query to detect the deadlock (i.e., see subquery "DeadlockDetection"). "DeadlockDetection" provides the output stream:

Listing 1.1. Generated Query to detect deadlocks

```
@Name('ExclusiveChoiceAsStreamactivity_2')
SELECT * FROM PATTERN 2
INSERT INTO ExclusiveChoiceStream
SELECT *
FROM EventRef.win:time(50 sec) as event
WHERE event.activity = 'Electronic␣invoice␣received'
HAVING NOT EXISTS (
    SELECT *
    FROM EventRef.win:time(50 sec) as subEvent
    WHERE subEvent.caseID = event.caseID
        AND subEvent.activity = 'Paper␣invoice␣received'
);

@Name('ExclusiveChoiceAsStreamactivity_1')
...

@Name('ParallelMergeAsStream')
INSERT INTO ParallelMergeStream
SELECT *
FROM EventRef.win:time(50 sec) as event
WHERE event.activity IN (
    'Order␣Amendment␣Confirmation'
,     'Clarification␣Sent␣to␣Supplier'
    )
GROUP BY event.caseID
```

```
HAVING COUNT(DISTINCT event.activity) = 2;

@Name('DeadlockDetection')
INSERT INTO DeadlockStream
SELECT ex.caseID, 'DeadlockDetected' AS DeadlockType
FROM ExclusiveChoiceStream.win:time(50 sec) AS ex
INNER JOIN ParallelMergeStream.win:time(50 sec) AS pm
ON ex.caseID = pm.caseID
    OUTPUT LAST EVERY 5 SECONDS;
```

We applied, i.e., matched, our generated queries against the considered event streams using Esper. The exemplary event stream, over which the above query was matched, can be found in our repository.

3.2 Application of the Online Heuristics Miner for Query Generation

Our CEP pattern generation relies on a predefined set of control-flow patterns. To construct such patterns from a given event stream, we implement the online heuristic miner, a state-of-the-art approach for online control flow discovery. As the online heuristic miner has limited performance on high-rate event streams, we ran it on a 20 second's long snippet from the Sepsis event log. Given that, the heuristic's miner identifies *84* control flow patterns (i.e., sequential: *28*, or: *28*, and: *28*). 1.2 shows one of the simplified queries generated from control-flow fragments discovered by the heuristic miner with ($\varepsilon = 0.01$). *<consistency check>* is a placeholder to turn the inclusive OR into an exclusive OR. The query is derived by the following control-flow fragment: *"OR(ER Triage, IV Liquid)"*

Listing 1.2. Exemplary query generated from the sepsis dataset

```
SELECT * FROM PATTERN [
EVERY(
    (a1 = EPPMEventType('IV␣Liquid') -> <consistency check>)
    OR
    (a2 = EPPMEventType('IV␣Liquid') -> <consistency check>)
    OR
    (a1 = EPPMEventType('ER␣Triage') -> <consistency check>))
)].win:time(50 sec)
```

3.3 Performance Evaluation

This section presents the evaluation results in terms of runtime efficiency. Following [10], we evaluated information latency, which refers to the time between where an event occurred and its processing by the system. The experiments were run on a MacBook Pro (Apple M2 and 16 GB of RAM). The system processed a stream of over two million events. Figure 4 illustrates the latency during the experiment. The plots were smoothed with (1) a rolling mean and (2) a rolling maximum approach (with window size of 5000 events). Please note that 14 queries have (on average) lower latencies than two queries, which might be counterintuitive, but can be explained due optimization efficiency and parallelism across the queries. However, the 14 query configurations demonstrate more frequently and higher latency spikes. This can be attributed to occasional resource contention or synchronization overhead. Additionally, the triggering of complex events may temporarily increase processing time.

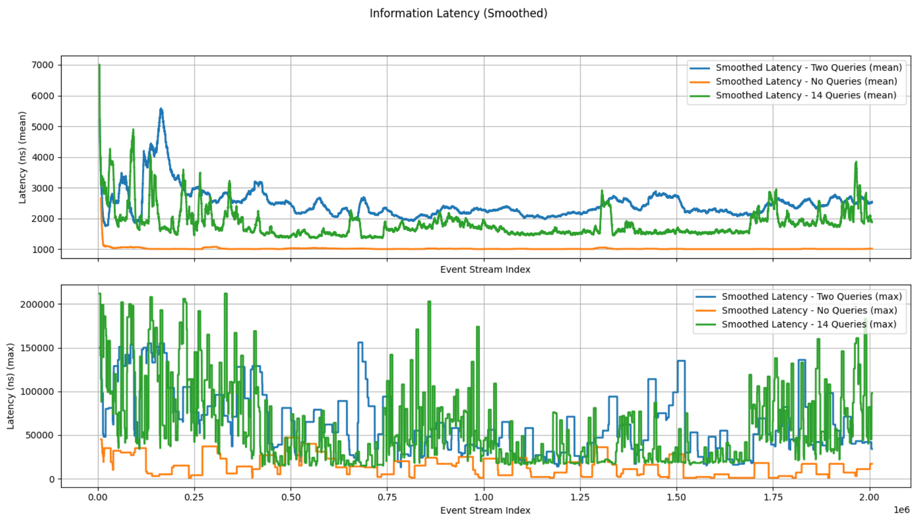


Fig. 4. Latency plots for the average and maximal latency inside a window size of 5000 events

4 Related Work

Several approaches exist that aim to identifying behavioral regularities or abnormalities in an offline setting. For instance, [13] introduced an approach to discovering behavioral models from software execution data.

[7] propose a method to repair outlier behaviors in event logs by removing infrequent events. Furthermore, [16] addressed the detection of structural behavioral flaws (e.g., deadlocks and lack of synchronization) from business process event logs. Additionally, approaches have been proposed, which address certain properties unique to the online setting. For instance, [4] discuss techniques for discovering processes that change over time. [2] introduced an online framework for process mining over unordered event streams. In the same vein, [14] introduces a framework for online concept drift detection in processes based on event streams. Moreover, [3] adapts the heuristic miner that incrementally mines processes from event streams. However, as also observed in our experiments, the

respective approach does not meet the scalability requirements of online data processing in the long term.

Despite these advances, there is a notable gap between classical process mining algorithms adapted for online settings and native online approaches. Classical algorithms, even when adapted, often struggle with efficiency and flexibility when confronted with the highly dynamic nature of real-time processing. Native online approaches, while designed to handle real-time data, sometimes lack the robustness and explainable pattern recognition capabilities of traditional methods.

5 Conclusion

Online process analysis seeks to identify behavioral regularities or abnormalities in processes in near-real-time from continuous event streams, a task that remains challenging. This paper introduced an approach for an online behavioral analysis. Through the "Detect and Conquer" approach, generic process templates are defined that capture most behavioral dependencies of process activities and are subsequently translated into CEP queries. These queries are then applied to streams of events. The approach has been evaluated using both synthetic and real-world datasets, and the results demonstrate its feasibility and efficiency.

Acknowledgements. This work received funding by the Deutsche Forschungsgemeinschaft (DFG), grant 496119880. The responsibility for the content of this publication remains with the authors.

References

[1] Wil Aalst, Arthur Ter, Bartosz Kiepuszewski, and Alistair Barros. “Workflow Patterns”. In: *Distributed and Parallel Databases* (Jan. 2003).

[2] Ahmed Awad, Matthias Weidlich, and Sherif Sakr. “Process Mining over Unordered Event Streams”. In: *2020 2nd International Conference on Process Mining (ICPM)*. Oct. 2020, pp. 81–88. URL: https://ieeexplore.ieee.org/document/9230157/?arnumber=9230157 (visited on 08/13/2024).

[3] Andrea Burattin, Alessandro Sperduti, and Wil M. P. van der Aalst. “Heuristics Miners for Streaming Event Data”. In: *CoRR* abs/1212.6383 (2012). arXiv: 1212.6383. URL: http://arxiv.org/abs/1212.6383.

[4] Josep Carmona and Ricard Gavaldà. “Online Techniques for Dealing with Concept Drift in Process Mining”. In: *Advances in Intelligent Data Analysis XI*. Ed. by Jaakko Hollmén, Frank Klawonn, and Allan Tucker. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 90–102.

[5] David Chapela-Campa, Manuel Mucientes, and Manuel Lama. “Mining frequent patterns in process models”. In: *Information Sciences* 472 (2019), pp. 235–257. URL: https://www.sciencedirect.com/science/article/pii/S0020025517304875.

[6] Gero Decker and Jan Mendling. "Instantiation Semantics for Process Models". In: *Business Process Management*. Ed. by Marlon Dumas, Manfred Reichert, and Ming-Chien Shan. Vol. 5240. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 164–179. URL: http://link.springer.com/10.1007/978-3-540-85758-7_14 (visited on 05/02/2024).

[7] Mohammadreza Fani Sani, Sebastiaan J. van Zelst, and Wil M. P. van der Aalst. "Repairing Outlier Behaviour in Event Logs". In: *Business Information Systems*. Ed. by Witold Abramowicz and Adrian Paschke. Cham: Springer International Publishing, 2018, pp. 115–131.

[8] Zhaogang Han et al. "Definition and Detection of Control-Flow Anti-patterns in Process Models". In: *2013 IEEE 37th Annual Computer Software and Applications Conference Workshops*. 2013, pp. 433–438.

[9] Christian Janiesch and Koschmider et al. "The Internet of Things Meets Business Process Management: A Manifesto". In: *IEEE Systems, Man, and Cybernetics Magazine* 6.4 (Oct. 2020), pp. 34–44. URL: http://dx.doi.org/10.1109/MSMC.2020.3003135.

[10] Jeyhun Karimov et al. "Benchmarking Distributed Stream Data Processing Systems". In: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 2018, pp. 1507–1518.

[11] Agnes Koschmider et al. "Process Mining for Unstructured Data: Challenges and Research Directions". In: *Modellierung 2024, Potsdam, Germany, March 12-15, 2024*. Ed. by Judith Michael and Mathias Weske. Vol. P-348. LNI. Gesellschaft für Informatik e.V., 2024, pp. 119–136. URL: https://doi.org/10.18420/modellierung2024%5C_012.

[12] Ralf Laue, Wilhelm Koop, and Volker Gruhn. "Indicators for Open Issues in Business Process Models". In: *Requirements Engineering: Foundation for Software Quality*. Ed. by Maya Daneva and Oscar Pastor. Cham: Springer International Publishing, 2016, pp. 102–116.

[13] Cong Liu. "Automatic Discovery of Behavioral Models From Software Execution Data". In: *IEEE Transactions on Automation Science and Engineering* 15.4 (2018), pp. 1897–1908.

[14] Na Liu, Jiwei Huang, and Lizhen Cui. "A Framework for Online Process Concept Drift Detection from Event Streams". In: July 2018, pp. 105–112.

[15] Felix Mannhardt. *Sepsis Cases - Event Log*. 2016. URL: https://data.4tu.nl/articles/dataset/Sepsis_Cases_-_Event_Log/12707639/1.

[16] Wei Song, Zhen Chang, Hans-Arno Jacobsen, and Pengcheng Zhang. "Discovering Structural Errors From Business Process Event Logs". In: *IEEE Transactions on Knowledge and Data Engineering* 34.11 (2022), pp. 5293–5306.

[17] Gabriel Marques Tavares et al. "Overlapping Analytic Stages in Online Process Mining". In: *2019 IEEE International Conference on Services Computing (SCC)*. ISSN: 2474-2473. July 2019, pp. 167–175. URL: https://ieeexplore.ieee.org/document/8813959/footnotes#footnotes (visited on 08/02/2024).