Complex Event Processing Methods for Process Querying

Han van der Aa, Alexander Artikis, and Matthias Weidlich

Abstract Business Process Management targets the design, execution, and optimization of business operations. This includes techniques for process querying, i.e., methods to filter and transform business process representations. Some of these representations may assume the form of event data, with an event denoting an execution of an activity as part of a specific instance of a process. In this chapter, we argue that models and methods developed in the general field of Complex Event Processing (CEP) may be exploited for process querying. Specifically, if event data is generated continuously during process execution, CEP techniques may help to filter and transform process-related information by evaluating queries over event streams. Against this background, this chapter first outlines how CEP fits into common use cases and frameworks for process querying. We then review design choices of CEP models that are of importance when adopting the respective techniques. Finally, we discuss techniques for the application of CEP for process querying, namely those for event-process correlation, model-based query generation, automated discovery of event queries, and diagnostics for event query matches.

Han van der Aa
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
e-mail: han.van.der.aa@hu-berlin.de

Alexander Artikis
Department of Maritime Studies, University of Piraeus, Greece
Institute of Informatics & Telecommunications, NCSR Demokritos, Greece
e-mail: a.artikis@iit.demokritos.gr

Matthias Weidlich
Department of Computer Science, Humboldt-Universität zu Berlin, Germany
e-mail: matthias.weidlich@hu-berlin.de

1 Introduction

The field of Business Process Management (BPM) assumes a process-oriented view on how organizations are structured [30]. In order to analyze the operations of an organization, its business processes are assessed. Such a process is defined by a set of activities, which denote atomic units of work, along with causal and temporal dependencies for their execution. An example would be a Lead-to-Quote process, which comprises activities such as *loading contact data from customer-relationship-management (CRM) system*, *estimating the project effort*, and *preparing a quote template*. Causal dependencies would then define that the loading of contact data precedes the other two activities, which may then be executed concurrently.

Process querying is concerned with models and methods to filter and transform representations of business processes [57]. As such, it supports manifold use cases, reaching from process modeling support, through variation management and performance simulation, to compliance verification. Process representations that are subject to querying may take various forms: A *process model* captures the designed behavior of a process. It specifies the activities and execution dependencies of a process, thereby serving as a blueprint for the execution of a specific instance [45]. A process model may be constructed for various purposes, such as process automation, staff training, or performance simulation. Hence, even for a single process, there may exist various models, each capturing the aspects of the process that are important in light of the purpose of the model [65]. The notion of *event data*, in turn, relates to process representations that capture the recorded behavior of a process, such that an event denotes that a certain state has been reached (e.g., an order request has been received) or that an activity has been executed as part of a specific process instance [4]. Event data is often formalized as an *event log*, a set of traces, each trace being a finite sequence of events that denotes past behavior for a particular process instance. Process-related data may also be available as an *event stream*, a potentially infinite sequence of events that represent the current behavior of a process.

Complex Event Processing (CEP) defines models and methods to make sense of streams of event data [14, 26]. It defines languages to express queries, which are then evaluated over an event stream, thereby implementing continuous filtering, transformation, and pattern detection. It therefore suggests itself to adopt event-based process querying through CEP, once process-related information is represented by event streams.

While CEP is developed for such *online* event processing, event-based process querying also enables various use cases for *offline* event analysis. This is achieved by replaying event logs, which encode temporal event orders [4], thereby rendering online event-based techniques applicable to static event data.

In this chapter, we outline how CEP methods can be used in the context of process querying. Specifically, this chapter delivers a contextualization and overview of essential techniques in this area, as follows:

- We embed event-based querying by means of CEP methods in the larger process querying context (Section 2). That is, we discuss CEP methods with respect to use cases and frameworks for process querying.

- We review common design choices of CEP models (Section 3). We highlight which aspects to consider when choosing among available event models, query languages, and system infrastructures when using CEP for process querying.
- We discuss essential techniques for the application of CEP for process querying (Section 4). This includes techniques to correlate events with other process representations, to derive queries from process models for control-flow monitoring, to discover such queries automatically from event logs, and to obtain diagnostic information on specific process behavior that is identified by event queries.

The chapter closes with a discussion of open research issues in Section 5.

2 The Context of Event-based Process Querying

Event-based process querying can be seen as a special variant of process querying, where the filtered and transformed process representations assume the form of event data. Below, we first elaborate on use cases for event-based querying. Subsequently, we discuss how event-based techniques fit into the broader picture of process querying and provide an illustrative example.

2.1 Use Cases

Methods for event-based querying enable the analysis of the recorded behavior of a process. In general, different types of analysis are distinguished, along several dimensions. Analysis questions may relate to a qualitative as well as a quantitative property of a process [30]. The former relates to recorded execution dependencies [24], e.g., whether two activities have been executed in a specific order or a particular number of times. The latter may be defined in terms of execution and wait times, or costs assigned to activity executions [29, 61]. In either case, however, not only the control-flow dimension may be considered. In addition to recorded activity executions, event data often also contains information on processed artifacts or involved resources [51], which can also be subject to event-based querying.

To illustrate the spectrum of applications for event-based querying, we consider two specific use cases: compliance verification and performance monitoring.

Compliance verification. Today, the execution of processes is widely supported by information systems. For processes in domains such as logistics [12] or healthcare [52], however, the execution of the actual activities is often conducted manually by diverse stakeholders. Yet, there are expectations on how the process is conducted, which, depending on the domain, originate, for instance, from reference models [33] or legal frameworks [49]. When a system guides the execution of a process, but does not enforce a specification of its expected behavior, compliance (aka conformance) between expected and recorded behavior needs to be verified explicitly [5, 24, 49]. Based on a formalization of compliance requirements, event-based process querying

helps to identify cases of non-compliant process execution [68]. Such mechanisms are particularly useful if applied to event streams representing the most recent behavior of a process: Detecting a compliance violation shortly after it occurred enables the immediate implementation of mitigation and compensation schemes.

In order to fully exploit the potential of event-based querying for compliance verification, several challenges should be addressed. First and foremost, compliance requirements need to be linked to events. For instance, if a particular ordering of activities is required, events must be correlated unambiguously with activity executions as part of a specific process instance to enable conclusions on the compliance of process execution. Moreover, the actual translation of compliance requirements into event queries is cumbersome, since it requires the formalization of the requirements in a (commonly declarative) query model. Hence, the construction of event queries for compliance verification must be supported, e.g., based on models that capture the expected behavior of a business process or event data that is annotated with compliance violations. Furthermore, the interpretation of compliance issues that have been detected by event-based querying is challenging. From a practical point of view, understanding the cause for a compliance violation is important to be effective in the mitigation and compensation of the issue.

Performance monitoring. For many processes, ensuring efficient process execution is a core requirement. Whether it is the cycle time of an order process at an e-commerce platform, the wait time of patients as part of clinical pathways, or the cost spent for claim handling processes at an insurance company—a good share of the success of process management depends on quantitative properties of how the process is conducted [30]. Event-based process querying helps to measure these properties: It selects events that are used as input for the computation of performance indicators [27], e.g., the average activity execution time, the delay with which a particular activity is executed after activation, or the accumulated costs induced by a specific type of process instance. At the same time, outliers that represent process execution with anomalous performance can be extracted [60]. Beyond the sheer assessment of the performance, event-based process querying further enables the detection of respective trends, such as continuous deterioration of process performance as well as abrupt drifts [21]. Again, immediate detection of performance issues is a prerequisite for effective countermeasures, which can be achieved through querying of event streams, but not through post-hoc analysis of event logs.

As for the previous use case, however, event-based querying for performance monitoring faces the challenge of event correlation. That is, while performance indicators are defined in terms of the business semantics of a process (e.g., based on the time needed to reach a milestone), the task to correlate recorded events with notions of process progress may be non-trivial. Also, the challenges on the construction of event queries and the interpretation of their results, mentioned above for compliance verification, are faced as well in performance monitoring. For instance, a delay in the execution of a business process may lead to ripple effects, so that a large number of performance requirements are not met. Upon detecting all these performance issues, an understanding of the initial deviation that caused further delays is important.

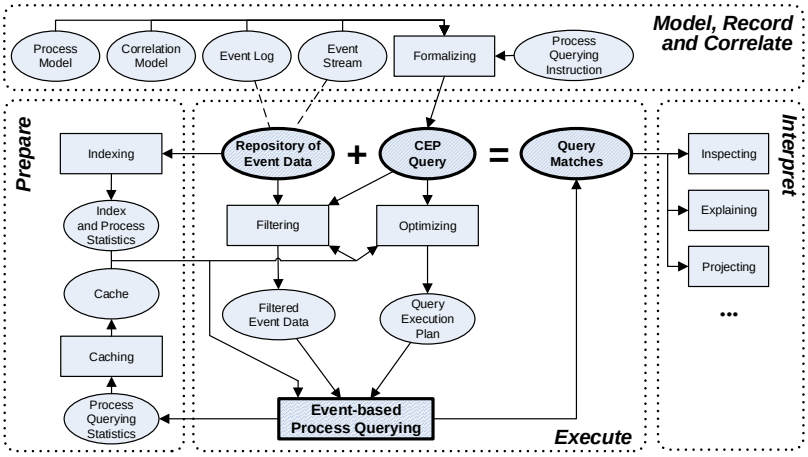

Fig. 1: Instantiation of a fragment of the Process Querying Framework [57] for event-based process querying based on Complex Event Processing.

2.2 The Context of Event-based Process Querying

Querying techniques that are grounded in event handling fit into established frameworks for process querying. We illustrate that through the Process Querying Framework (PQF) as introduced by Polyvyanyy et al. [57]. It defines active and passive components, which jointly realize the functionality needed for process querying.

Figure 1 shows a fragment of the PQF, which organizes the respective functionality into four parts, targeting (i) the design of process repositories and queries through modeling, recording, and correlation; (ii) the preparation and (iii) execution of queries; and (iv) result interpretation. In the context of event-based process querying, components of this framework are instantiated as follows:

(i) Model, record, and correlate. A repository, in general, may comprise various types of artifacts, also referred to as behavior models. For event-based process querying, some types of such models are of particular importance. First and foremost, event logs and event streams represent the event data that is the actual target of any event-based query mechanism. As discussed above, an event log is a set of recorded traces of events that denote past process execution [4], whereas an event stream is a potentially infinite sequence of events that indicates how the process is currently conducted [23, 37]. Event streams, therefore, cannot be stored in a repository in their entirety. However, most use cases for process querying inherently justify the consideration of a bounded subsequence of an event stream. For instance, queries commonly analyze individual or specific groups of process instances, so that, assuming that process instances eventually terminate, the finiteness of event data relevant to a particular query is ensured.

Moreover, behavior models that assume the form of process models and correlation models play a role for event-based process querying. Neither type of artifact is used as a query target, i.e., they are not part of the repository in the sense of the PQF, but may guide the definition of event queries. This is achieved by formalizing a process querying instruction. In part, this instruction may automatically be generated from a process model [16, 68], which we detail later in this chapter. For instance, for either of the above use cases, the expectations regarding compliant and high-performance process execution may have been materialized as a process model. As such, queries that are derived from it can be used to ensure that these expectations are met in the process behavior as recorded in event data. To this end, however, it may be required to first establish the relation between process model elements and event data using a correlation model [18]. Again, we will highlight essential techniques for this step.

In the absence of artifacts that can be used to formalize process querying instructions, the event data itself may also form a starting point for query development. As we will detail later, based on event data that is annotated with relevant situations that shall be matched, a respective query may actually be learned (semi-)automatically.

(ii) Prepare. Given a repository of event data and an event query, the evaluation of the query may be prepared. The PQF defines indexing and caching schemes as the major means for such preparation. Those also apply to event-based process querying, whereas their implementation depends on the specifics of the respective type of event data and formalism adopted for event queries. Basic techniques include indexing of specific types of events, their frequencies, or constraints on the occurrence or absence of events in an event log or an event stream.

(iii) Execute. As part of the execution stage, the information resulting from the preparation is exploited to filter the event data and optimize the execution plan of an event query. An example for the former is the projection of all events that cannot be of relevance for the query at hand. Examples for the latter are satisfiability checking [28], rewriting of event queries [67], or sub-pattern sharing [59] based on information on the event data. In any case, the execution of the query yields a set of query matches. As those are given as event data, they may be thought of yet another process repository as put forward by the PQF.

(iv) Interpret. Obtained query matches represent the input to the last part of the framework. Matches may be interpreted in light of the respective application scenarios. For the aforementioned use cases, for instance, query matches may denote compliance violations or aggregated performance measurements. By putting query matches into context, e.g., comparing them to those obtained by other queries, for other traces, or at other times, they may also enable the explanation of the queried phenomena. As an example, we later review a technique that provides diagnostic insights into non-compliance by assessing the interplay of the matches obtained with event queries.

Query matches may also be used for further interpretation. In Figure 1, we included the option to project the results on behavior models for analysis purposes. For instance, projecting query matches on the originating event data may be useful to quantify the relative amount of observed matches. However, in practice, many further

types of interpretation of query matches are possible, so that this list is not supposed to be understood as an exhaustive overview of how to make sense of query matches.

2.3 An Example Scenario

We now turn to an exemplary scenario related to a *Lead-to-Quote* process, which will serve as a running example throughout the remainder.

Key event types in such a Lead-to-Quote scenario will capture process milestones including *received a lead*, *project details entered into ERP (Enterprise Resource Planning) system*, and *quote sent*. Events of these types may be derived from the relational data stored in the system, e.g., the insertion of a tuple into a relation that captures leads can be interpreted as a signal that an instance of an activity *received a lead* had been executed. Given these events, managers can establish monitoring requirements that track behavioral properties, such as the *qualitative requirement* that project details must always be entered into an ERP system before a quote can be sent out, or a *quantitative requirement* used to ensure that a quote is sent out within two weeks after receiving the lead. The automatic and continuous monitoring of these requirements is then achieved by translating them into event queries. In the presence of models or annotated data, this translation may even be supported using automated techniques. The resulting queries can use both pattern recognition (e.g., events occurring in a particular order for a specific case) and data attribute analysis (e.g., timestamps and other payload data associated with events) to monitor both qualitative and quantitative monitoring requirements.

Finally, we introduce a process model for the Lead-to-Quote scenario, depicted in Figure 2. The modeled process starts with an import of contact data (activity a or b) or with the receipt of a request for quote (c). In the former case (following a or b), the contact details are updated (d). This step may be repeated if data integrity constraints are not met (e). In the latter case (after c), the request details are checked (f). For both cases, the quote is then prepared by first entering prospective project details (g). This is followed by conducting an effort estimation (h and k), updating the requirements (i), and preparing the quote template (j). These latter steps are done concurrently. Finally, the quote is approved and sent (l). A process model like this may have been created in order to capture the expected or required behavior of the process. It can serve as a basis to (semi-)automatically derive monitoring queries that allow for the continuous comparison of expected to actual process behavior, as described later in this chapter.

3 Complex Event Processing

Complex Event Processing (CEP) emerged as a computational paradigm to handle streams of event data [14]. This paradigm has to be seen in the context of the broader areas of data stream processing [34] and stream reasoning [11], which are all

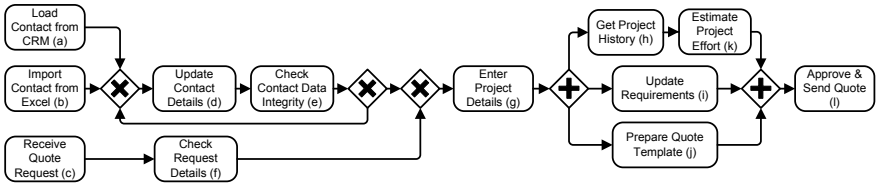

Fig. 2: Example Lead-to-Quote process modelled in BPMN.

concerned with online processing of data that is continuously generated [26]. More specifically, the focus of common CEP models and methods is the detection of event patterns, sets of events that are correlated in terms of their ordering, their payload data, and the context in which they occur.

Even when considering only the scope of CEP, however, we note that a plethora of different event processing models have been proposed in literature. We therefore refrain from adopting one particular model and rather highlight important aspects of formalisms for event streams and event queries, as well as query evaluation infrastructures. As such, our focus is to highlight the spectrum of models and methods for CEP that may be exploited for process querying.

3.1 Event Streams

An *event* is commonly seen as a an 'occurrence within a particular system or domain' [32]. It is a recording of some state change that is considered to be of relevance. In the context of process querying, such state changes typically refer to the progress of process execution as, for instance, indicated by the execution of an activity as part of an instance of the process.

Events may be defined following different formalisms and conceptual models. However, most models share the requirement to capture some essential information about an event, as follows. First and foremost, an event e has an *identifier*, which we denote by $e.id$. It renders the event uniquely distinguishable. Second, each event is assigned a *timestamp*, denoted by $e.t$. Such event timestamps enable (relative) ordering of events. In addition, they may be exploited in event queries to assess the absolute time difference between events. Third, each event is of a specific type, denoted by $e.type$. A type refers to a specification that serves as a meta-model for a set of events, i.e., events are instances of their type specification. In many application scenarios, event types define both a specific syntax to capture respective events as well as their semantics.

An event stream is defined by a potentially infinite set of events E and an order relation $\prec \subseteq E \times E$ (either partial or total). The fact that a stream is potentially infinite means that, in practice, processing is based on the stream at a specific point in time, i.e., the prefix of the stream up to this time.

Despite its simplicity, the above event stream model incorporates assumptions and design choices along several dimensions:

Timestamp semantics: The exact meaning of the timestamp of an event may vary. The timestamp can, for instance, denote the time of event occurrence, the time of event recording (which is potentially delayed with respect to its occurrence), or the time of event arrival at a CEP infrastructure [20].

Event atomicity: Events may have point-based or interval-based semantics, meaning that an event is either atomic or has a duration [46]. Conceptually, an even more fine-grained structure may be considered, in which events comprise multiple intervals that represent suspension and resumption.

Stream ordering: Event stream models differ in their assumptions on the ordering of events. Specifically, the order relation $\prec$ can be assumed to be a total order or a partial order [69]. The former may keep synchronization issues out of the event processing model, whereas the latter enables to incorporate event generation by distributed, independent sources [9]. Also, note that incorporating more than one timestamp semantics leads to multiple notions of order for the events of a stream, e.g., an occurrence order and an arrival order, so that inconsistencies need to be handled [48].

Payload data: While the type of an event defines its syntactic structure, this structure may assume various forms. For instance, events may be structured according to a relational model (a type defines a relational schema) [13], adopt an object-oriented model (a type defines a set of concepts and their relations), or use tree-based formalisms to define the data carried by an event (a type defines, for instance, an XML schema) [72]. Note that relational models are often employed already in the context of process querying [57].

Event semantics: In a simple model, semantics of events are directly induced by their type definition. However, it has been argued that these semantics shall also be specified explicitly, adopting appropriate formalisms for knowledge representation [36, 63].

For illustration, we take up the scenario introduced in Section 2.3. Adopting a relational model for the payload of events, Table 1 lists three exemplary events. This illustrates some of the aforementioned aspects of event models: Instead of having separate types per reached state or executed activity, the example events are of a unified type `Act` (representing that an activity has been executed), while an attribute `name` references milestones (e.g., `QR` for quote request received) and activities (e.g., `ED` for entering details). Moreover, events are atomic, while the timestamp semantics is assumed to denote the end of activity execution. The identical timestamps of events `11` and `12` illustrate that events may happen concurrently (e.g., a batch of quote requests is received), inducing a partial order over the stream.

Table 1: Three example events for the introduced scenario.

id	timestamp	type	order_id	name	client	price
11	21.09.18,15:12:36	Act	O23	OR	Franklin	745,00
12	21.09.18,15:12:36	Act	O67	OR	Meyers	282,00
42	24.09.18,09:72:10	Act	O23	ED	Franklin	745,00

3.2 Event Query Languages

Numerous models and languages for the definition of event queries have been proposed in recent years [14,26,40]. While there have been recent initiatives to converge on a model, notably the match-recognize operator for row-based pattern matching as included in SQL:2016 [42], as of today, there is no common standard for event query languages. Rather, CEP systems define their own languages, differing in syntax and semantics. Since the respective languages have been developed in various communities, any comparative assessment is further hindered by the differences in the adopted terminologies and underlying event model [26]. In this section, we therefore focus on an overview of generic types of query operators, exemplify the syntax of query languages, and point to different ways to achieve a formal grounding of queries.

Common operators types. It has been noted that many languages for the specification of event queries, despite all their differences, share at least a set of common operator types [73]. While the specific definitions of query operators may still vary in syntax and semantics, these types describe rather abstract functionality that is typically supported by a query language. Specifically, these types are:

Disjunction and Conjunction: Query operators define that an event pattern is characterized by the occurrence of either of a set of events (disjunction), or their joint occurrence (conjunction).

Sequencing: A sequence operator defines a list of events that have to occur in the respective temporal order for a query to match.

Kleene Closure: An operator defines a pattern as a recurring occurrence of a specific event, with the number of occurrences being finite, but unbounded.

Negation: A query operator checks for the absence of a specific event, generating query matches only if the respective event is not part of the processed stream.

Data Predicates: A query operator specifies conditions for events to be part of a match, based on their data payload.

Windowing: A window operator specifies time-based or ordering-based conditions for events to become part of a query match.

Event Construction: A query operator specifies how a new event, emitted as part of an output stream generated by a query, is constructed from matched events.

Again, it has to be stressed that the precise definition of operators of these types are typically language-specific. For instance, the semantics of operators such as sequencing and Kleene closure needs to be further disambiguated through processing policies that clarify, for instance, how often a single event can be part of a match;

which further events may occur between matched events; and how to select among multiple candidate events for a query match, see [7, 39].

Moreover, query languages differ significantly in terms of their compositionality, i.e., the support to build complex queries by combining operators of the aforementioned types. Several existing languages restrict compositionality, e.g., in terms of nesting Kleene closure operators.

Exemplary languages. To illustrate how the above operators are actually represented in common query languages, we now take up the example scenario introduced in Section 2.3. Specifically, we focus on compliance verification and the requirement that project details for a received quote request must be entered *before* a quote can be sent out. As an additional condition, we require that this rule applies only for quotes with a value of more than 10.000 Euro. Let us assume that the underlying event model defines a single type for activity executions (with an attribute capturing the activity), incorporates atomic events that are totally ordered and denote the occurrence of activity execution, while the event payload is always defined by a relational schema.

Figure 3 exemplifies how such a query would be defined in two languages. Figure 3a defines the query in the SASE language [73]. After specifying the input stream, the query comprises a pattern definition. The latter includes a sequence (`SEQ`) of three event variables (`a1-a3`), each of the same type `Act`, while the second variable is negated (`!Act a2`). The `Where` clause then specifies conditions for the pattern to detect. The clause captures processing policies (`skip_till_next_match` means that irrelevant events may be skipped as long as no relevant event occurs) and data predicates, such as `[request_id]` correlating all events by the identifier of the quote request and statements that refer to the names of executed activities. Finally, a time-based window is defined for the query (`Within`).

Figure 3b illustrates a related (due to subtle differences in semantics not equivalent) query in the Esper Pattern Language (EPL) [31]. Here, the partition of the input event stream is realized through the definition of a `context` over the attribute `request_id`. Under this context, the actual query then selects data from a pattern that defines that an `Act` event, for which the name indicates a quote request, shall be followed by an event related to the quote submission, without an event capturing that the details have been entered. While adopting a different syntax, this query also contains the respective data predicates as well as the window definition.

Formal grounding. For query languages such as those illustrated above, different models have been proposed to use as a formal basis. Unfortunately, we note that most of the proposed formalisms suffer from two problems: They are incomplete, i.e., they capture only a subset of the aforementioned query operators; and they are not based on well-established formalisms, so that existing theoretical results and reasoning methods cannot be exploited in the context of query analysis.

As reviewed in [14], the most common proposals to formalize event queries include automata-based, tree-based, and logic-based models. Systems as Cayuga [22], SASE [73], and TESLA [25] adopt automata, in which the states represent the progress in query processing, while state transitions carry guards that encode the conditions for the evolution of a partial match. As these models operate on an infinite

```
From act_stream
Pattern
  SEQ(Act a1, !Act a2, Act a3)
  Where skip_till_next_match(a1,a3)
  And [request_id]
  And a1.name='QR'
  And a2.name='ED'
  And a3.name='SQ'
  And a3.price > 10000
  Within 10 days
```

(a) Example query in SASE [73]

```
Create Context qr_context
  Partition By request_id
  From act_stream;
Context qr_context
Select * From
Pattern [
  Every Act(name='QR') -> (
    Act(name='SQ' and price>10000)
    And Not Act(name='ED'))
  Where timer:within(10 days)];
```

(b) Example query in EQL [31]

Fig. 3: Example queries.

universe of events, evaluate predicates over data, and produce output explicitly, they incorporate ideas of symbolic and register automata as well as transducers. However, the models proposed for CEP languages are typically not directly based on such established notions of formal language theory.

Queries may also be formalized as trees, where non-leaf nodes are event operators that construct partial matches from the events matched by their children. Leaf nodes, in turn, define data predicates for the single events to be matched. An example for this approach is the model of ZStream [54].

Finally, logic-based formalisms can serve as a basis for event queries. Examples include chronicle recognition and the event calculus. The former relies on temporal logic. It encodes the occurrence of events by logic predicates that define the time of occurrence and the event payload; see the example of TESLA [25]. Contextual and temporal constraints then define event operators, time windows, and data predicates of an event query. The event calculus [15] relies on fluents as essential building blocks. A fluent is a property that may assume different values over time, while changes in this property are encoded by logic predicates. Queries in the event calculus are then defined as rules over the fluents.

3.3 Event Query Evaluation

Various systems and infrastructures have been proposed for the evaluation of event queries. In most cases, the formal models mentioned above, which serve as a basis for the definition of event queries, directly give rise to an execution model. An automata-based formalization of a query is used as follows: A CEP system keeps track of partial runs of the automaton. Processing an event then requires to assess whether a new run shall be instantiated according to the initial state of the automaton, and whether the existing runs shall be extended, duplicated, or terminated [22]. Similarly, adopting a tree-based model for evaluation, a CEP system maintains buffers for all leaf-nodes. Upon filling them with a batch of new events, buffers for partial matches at non-leaf nodes are filled or emptied [54]. For logic-based models, a CEP system

conducts logical inference to see whether query matches materialize. In a streaming setting, this is done whenever facts representing new events have been inserted into the knowledge base [15].

The continuous evaluation of queries over high-velocity event streams is a common performance bottleneck, so that a plethora of optimization strategies have been developed. A recent survey [41], focusing not only on CEP but the broader area of stream processing, classifies these optimization strategies based on whether they change the topology of an operator graph, whether they change the semantics of queries, and whether they are applicable at design-time or run-time. Specific examples of optimization techniques include load shedding for CEP that limits processing to a subset of the arriving events [38]; delayed construction of partial matches during run-time [73]; semantic query rewriting based on constraints on the event stream [67]; and sharing of partial matches among several queries [59].

4 Methods for Process Querying

The application of CEP models and methods as discussed above in the context of event-based process querying has to cope with several challenges, partially mentioned for two exemplary use cases in Section 2.1. In this section, we take up these challenges and discuss four essential techniques to address them: event-activity correlation (Section 4.1), model-based query derivation (Section 4.2), discovery of event queries (Section 4.3), and diagnostics for event query matches (Section 4.4).

4.1 Event-Activity Correlation

A fundamental requirement for analysis techniques involving event data alongside other representations of a process, i.e., process models, is that observed event types can be linked to process model elements, such as activities or decision points. In terms of the Process Querying Framework [57], these links are captured in a correlation model that establishes the relation between elements of different process representations. For instance, in compliance verification, the expected behavior of a process may be formalized by a process model, against which the recorded behavior, i.e., events that denote activity executions, is assessed. Typically, however, such required event-activity correlation is not readily available [18, 55]. Furthermore, *manually* establishing correlation is often unfeasible because analysts rarely possess the necessary knowledge on the details of a process implementation [67]. Consequently, it is highly beneficial to establish event-activity correlation in an *automated* fashion. However, to reliably achieve this, challenges including cryptic data values in the definition of events, noisy and non-compliant behavior, as well as complex event-activity relations must be taken into account [3]. A variety of automated techniques have been developed that aim to overcome such challenges and

to identify correspondences between recorded events and process model elements, for convenience referred to as *activities* in the remainder. In this sense, the goal of event-activity correlation can be framed as a matching problem. To address this problem, automated correlation techniques can consider various types of information:

1. Event and activity label information: The labels (or values) assigned to attributes that are associated with recorded events and process model activities represent valuable information for the establishment of event-activity correlation. In optimal scenarios, events and activities can be correlated when they have equal or highly similar names, e.g. an event carries an attribute with the label *project information submitted*, while an activity is labeled *enter project details*. A plethora of *similarity measures* exist that can be used to compute the degree of similarity between two text fragments. These measures can be generally divided into streams of syntactic and semantic similarity measures [10].

Syntactic similarity measures, such as the *string edit distance* and N-gram distance, compute the degree of similarity between two text fragments by comparing their character sequences. By contrast, *semantic* similarity measures, such as measures based on Wordnet or on Distributional semantics, consider word similarity based on the meaning of words [8]. For example, the words "*contract*" and "*agreement*" have a high semantic similarity, because they both describe an *exchange of promises*. Both types of similarity measures have their benefits and disadvantages. This is illustrated in Table 2 (derived from [1]), which compares the values of syntactic and semantic similarity measures, respectively obtained using the *Levenshtein distance* [71] and *Lin similarity* [47]. Note that both measures range between 0.0 and 1.0, where 1.0 denotes perfect similarity. The table, for instance, shows that syntactic similarity measures can recognize similarity in spite of typographical errors (indicated by the high $sim_{syntatic}$ value for *agreement* vs *argeement*), whereas semantic measures are able to differentiate among words with similar syntax but different meanings (contract vs contact). Therefore, both types of similarity measures are often used in conjunction.

Table 2: Comparison of syntactic and semantic similarity scores

t_1	t_2	$sim_{syntactic}$	$sim_{semantic}$
agreement	*argeement*	0.88	n/a
contract	*contact*	0.88	0.10
contract	*agreement*	0.11	0.96

However, in practice, recorded events are often associated with far less useful labels. For examples, attributes of an event may merely associated with cryptic database fields such as *CDHDR* or *I_SM_E* [19]. In these cases, not even advanced linguistic analysis tools are able to reliably identify a correlation with process model elements, if only label information is considered.

2. Structural information: Information on the behavioral relations that exist among events and among process model activities can be highly relevant when establishing event-activity correlation. As an illustration, consider the situation in which two event types, e.g., E_1 and E_2, appear to be mutually exclusive, i.e., where for every observed process instance at most one of them occurs. Such an event relation can be a relevant indicator to determine that these event types correspond to, for instance, the *Import contact from Excel* and *Receive quote request* activities from the model in Figure 2, which also exclude each other. Similarly, the position at which events occur in process instances, e.g., towards the beginning of an instance, can be a strong indicator that these event correlate with activities that appear at comparable positions in the process model.

Event-activity correlation techniques consider structural properties in different ways. Van der Aa et al. [2] propose a measure for *positional similarity* quantifying similarity based on the average position in which events or activities occur in process instances. Baier et al. [17] analyze more complex behavioral relations by deriving *declarative process constraints* for events and activities. These constraints can capture various structural relations, such as those depicted in Table 3. For example, the `init` constraint can be used to specify that an event or activity occurs first in a process instance, whereas constraints such as `coexistence` and `response` identify inter-relations among events or activities. Given such structural relations, the approach from [17] then identifies a 1:1 correlation between events and activities for which the structural relations are most similar, through constraint-based optimization.

Table 3: Exemplary declarative process constraints

Constraint	**Explanation**
`init(a)`	`a` is executed first in a process instance.
`coexistence(a, b)`	If `a` occurs in a process instance, then `b` also occurs, and vice-versa.
`response(a, b)`	If `a` occurs in an instance, then `b` will eventually occur afterwards.
`chainResponse(a, b)`	If `a` occurs in an instance, then `b` will immediately occur afterwards.

A challenge regarding the derivation of process-event correlation based on structural information is that this implicitly assumes that the observed events follow the process in accordance with the specified process model. However, in practice, factors such as noise and non-compliant behavior can lead to deviations between the recorded events and the specified process model. As a result, correlation techniques can, for instance, not blindly assume that when a process model specifies that activity *A* occurs before activity *B*, the corresponding events will always be observed in this order as well. To deal with this, techniques such as the one from Baier et al. [17] use probabilistic means to identify the most likely correlation, taking into account potential process deviations.

3. Pattern matching: A key challenge when establishing event-activity correlation is that there can exist complex relations among events and activities. Often, recorded events correspond to more fine-granular process steps than process model activities, which can lead to many-to-one or many-to-many relations [18]. Recognizing such complex correlation generally requires the consideration of information beyond event and activity labels or structural relations.

A technique by Baier et al. [18] aims to recognize one-to-many and many-to-many correlations by considering natural language texts (i.e., *work instructions*) that may be associated with process model activities. The premise of their approach is that such work instructions contain detailed information regarding the execution of process activities, which may correspond better to the level of detail of observed events.

Still, event-activity correlation can be even more complex, in situations where process model activities correspond to particular event patterns. In these situations, a single model activity may correspond to the occurrence of an event pattern [50]. If these complex correlations are known, they can be manually established, though such a task heavily depends on domain knowledge from analysts.

However, it is also possible to automatically correlate complex event patterns to process model activities. For instance, Senderovich et al. [62] discover correlations between sensor readings (i.e., events) and process model activities by analyzing sensor readings that indicate employee interactions in healthcare environments. Their approach considers factors such as the entities involved in the interaction, its location and duration, and the preceding and succeeding interactions. Given these factors, the approach then aims to identify event patterns that correspond to particular activities in a process model, ultimately yielding an event-activity correlation.

4.2 Model-based Query Generation

As discussed in Section 2.1, compliance verification is an important use case for event-based process querying. Assume that a specification of the expected behavior of a process is available in terms of a process model and that its elements are linked to event types by a correlation model. If the respective specification is not enforced during execution, event queries may be formulated to detect any deviation of the recorded from the modeled behavior. In this section, we therefore consider how to generate such monitoring queries from a process model. For this section, we follow a generic architecture, as introduced next.

4.2.1 Overview

Figure 4 provides a generic architecture for systems that facilitate the automatic generation of monitoring queries as well as the preprocessing of their results to provide diagnostic insights. The system takes two kinds of input: process models and process events. The *process models* are used to define the normative behavior

of processes. We do not impose any assumptions on the process model notation or language used to define them. Regarding the *process events*, we assume that these events reflect the completion of activities as recorded by some information system. If recorded events do not directly capture this, correlation techniques such as discussed in Section 4.1 can be used as a preprocessing step.

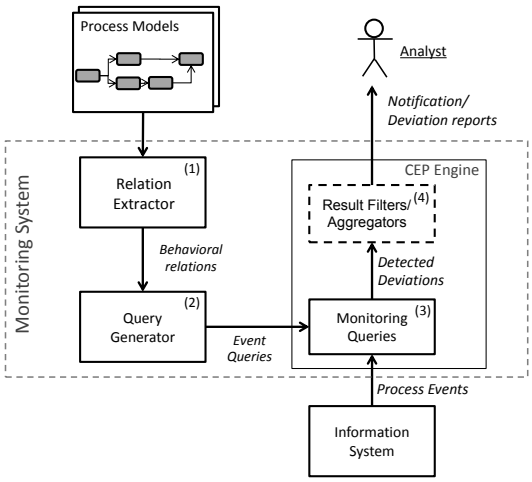

Fig. 4: Overview of a generic architecture for the generation of monitoring queries

As depicted in the figure, the generation and preprocessing of monitoring queries consists of four main steps: (1) the extraction of behavioral relations from process models, (2) the generation of monitoring queries, (3) the evaluation of these queries over process events, and (4) filtering and aggregating the detected deviations. In this section, we focus on Steps 1 and 2, which result in the generation of monitoring queries from normative process models. The evaluation of these queries (Step 3) is performed by CEP engines, as discussed in Section 3.3. In the light of the different sets of query operators provided by common query languages, here, we merely assume that event queries may comprise binary operators for conjunctive (*and*), ordered (*ord*), and subsequent occurrence (*sub*) of events of a particular type, whereas a negation operator supports testing for the absence of events (*not*).

Finally, the post-processing of the identified deviations using filtering and aggregation (Step 4) to provide diagnostic insights is discussed in Section 4.4.

4.2.2 Query Derivation

We derive monitoring queries from a process model by first computing behavioral relations for the model. Behavioral relations [56], such as the sets of alpha relations [6], (causal) behavioral profile relations [66], and the relations of the 4C spectrum [58],

define constraints that should hold between process activities according to a process model. Given a certain set of behavioral relations, we can then establish a monitoring query that identifies when the behavioral relation is not satisfied, i.e., when a compliance violation occurs.

As an illustration, we will discuss the derivation of monitoring queries for four behavioral relations: *exclusiveness*, *co-occurrence*, *strict order*, and *directly follows*.

Exclusiveness: The exclusiveness relation +, part of the behavioral profile relations, denotes that two activities should not occur in the same process instance. For instance, activities a and c should not occur in the same instance of the running example from Figure 2. To identify compliance violations of these constraints, we define queries that match the joint execution of two exclusive activities, e.g., we define a query that recognizes instances where both a and c are executed.

Using + to denote all pairs of process model activities that are in an exclusiveness relation (including self-relations), we define the *exclusiveness query set* as follows.

$$Q_{+} = \bigcup_{(a_1,a_2)\in +} \{and(a_1,a_2)\}.$$

For the model in Figure 2, monitoring exclusiveness comprises the following queries, $Q_{+} = \{and(a,a), and(a,b), and(b,a), and(a,c), \ldots, and(e,f)\}$. Mirrored queries such as $and(a,b)$ and $and(b,a)$ have the same semantics and can, therefore, be filtered through optimization techniques of a CEP implementation.

Co-occurrence: The co-occurrence relation $\gg$, part of the causal behavioral profile relations, denotes that two activities should always occur together in a completed process instance. For example, for the model in Figure 2, there is no trace that can contain only activity c but not f, and vice versa. Contrary to the query patterns derived for exclusiveness, co-occurrence constraints are violated not by the *presence* of a certain activity execution, but by its *absence*. Therefore, a constraint violation materializes only at the completion of a process instance. Only then it becomes visible which activities are missing in the observed execution sequence even though they should have been executed. Using $\gg$ to denote the set of activities in a co-occurrence relation for the behavioral profile of a model, we construct a corresponding query set as follows:

$$Q_{\gg} = \bigcup_{(a_1,a_2)\in \gg} \{and(and(a_1,END), not(a_2))\}.$$

For a running process instance, the question whether an activity is missing cannot be answered definitely, because the activity may still occur. Nevertheless, the strict order relation can be exploited to identify states of a process instance, which may evolve into a co-occurrence constraint violation. In particular, it is possible to query for activities for which we deduced from the observed execution sequence that they should have been executed already. That is, their execution is implied by a co-occurrence constraint for one of the observed activities and they are in strict order with one of

the observed activities, see [68] for a detailed description.

Strict order: The strict order relation $\rightsquigarrow$, part of the behavioral profile relations, indicates that two activities should occur in a particular order, if they both occur in the same process instance. For example, for the running example, the relation $e \rightsquigarrow g$ holds, since g can never occur before e. By contrast, the relation $d \rightsquigarrow e$ does not hold, due to the loop surrounding the two activities. We query for violations of the strict order of activities with an order query set that matches pairs of activities for which the order of execution is not in line with the behavioral profile relation $\rightsquigarrow$. Using $\rightsquigarrow$ to denote the set of activities of a model in strict order, we define these queries as:

$$Q_{\rightsquigarrow} = \bigcup_{(a_1,a_2)\in\rightsquigarrow} \{ord(a_2,a_1)\}.$$

For the model in Figure 2, the order query set contains the following queries, $Q_{\rightsquigarrow} = \{ord(d,a), ord(e,a), ord(g,a), \ldots, ord(l,k)\}$.

Directly follows: The directly follows relation $<$ from the set of alpha relations denotes that two activities only directly succeed each other in a particular order. For instance, activities h and k are in the relation $h < k$, since k never directly precedes h, while it can directly follow h. We query for violations of the respective relation with a query set that matches pairs of activities with subsequent executions, which are not captured by relation $<$. Let A be the set of all activities in a process model and let $<$ denote the set of activities in the directly follows relation. Then, we define the respective set of queries as:

$$Q_{<} = \bigcup_{(a_1,a_2)\in(A\times A)\setminus<} \{sub(a_1,a_2)\}.$$

Again, we use the model in Figure 2 for illustration. The query set contains the following queries, $Q_{<} = \{sub(a,b), sub(a,c), sub(a,e), \ldots, sub(h,l)\}$.

4.3 Discovery of Event Queries

The previous section showed how to construct event queries based on a specification of the expected behavior of a process, under the assumption that events have been correlated with process model elements. However, such an approach may not be feasible. It may be impossible to link events to a process model, e.g., due to severe differences in the assumed level of abstraction, or a process model specifying the expected behavior may not be available at all. If so, however, historic event data that is annotated with the situation of interest, e.g., a compliance violation or the attainment of a milestone, may serve as the basis for process querying. In this section, we first outline how such annotated event data leads to a supervised learning problem, before discussing a specific algorithm to address this problem.

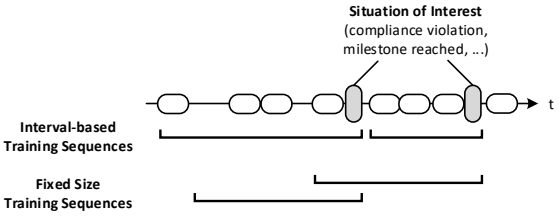

Fig. 5: The setting of event query discovery.

4.3.1 The Problem of Event Query Discovery

The setting of event query discovery is illustrated in Figure 5. Some event data (assumed to be totally ordered in the figure) is assigned annotations that indicate that a specific situation materialized. Such a situation may, for example, be an instance of the process violating a compliance rule or reaching a milestone during processing. Either way, the respective situation may be identified in retrospect in order to obtain such annotations. While the annotations identify the point in time at which the situation occurred, the actual event pattern indicating it is not, or only partially, known.

The problem of event query discovery is the construction of a query that matches whenever an annotation indicates that the situation of interest occurred. That is, the annotated event data serves as a training dataset with respect to some data without annotations, presumably generated by the same process. Under the assumption that the materialization of the situation in the annotated event data can be projected to the one without annotations, analysis may then rely on the derived query to detect formerly unknown occurrences of the situation. Put differently, query learning postulates that the event pattern signaling the situation is the same in the annotated and plain event data.

When aiming at the construction of a query that matches whenever there is an annotation, the scope of potential matches has to be limited, though. As the query shall generalize over the multiple occurrences of an event pattern (one per annotation), a *set* of training sequences has to be derived from the annotated event data. As illustrated in Figure 5, different approaches may be followed for this purpose. For example, all events between two subsequent annotations may be considered as a separate, *interval-based* training sequence, so that no two sequences overlap. This implicitly encodes an assumption on how occurrences of the situation of interest materialize—on each occurrence the state of processing is entirely reset. Depending on the process at hand, however, this assumption may not hold true. In that case, *fixed size* training sequences may be derived from a fixed amount of the event data preceding each annotation. Here, the amount may be determined based on volume (a fixed number of events) or time (a fixed temporal period).

4.3.2 Discovery Algorithms

Any attempt to address the problem of event query discovery has to be tailored to the models assumed for event streams and event queries, respectively. Intuitively, the more expressive these models, the larger the space of candidate queries to be considered as solutions for the discovery problem. A simple case would be the one of an event stream that comprises a total order of symbols and a query model that defines queries solely as a sequence of such symbols. Then, query discovery may simply be traced back to frequent sequence mining [64, 70], detecting the subsequences that are shared among all training sequences.

However, as described in Section 3.1 and Section 3.2, common models for event streams and query languages, in particular in the context of event data generated by processes, are much more complex. Events are not simple symbols, but typed and carry a structured payload. Queries are not limited to sequences of symbols, but may comprise data predicates, negation operators, and time windows. Against this background, a few tailored algorithms have recently been proposed for event query discovery, specifically iCEP [53] and the IL-Miner [35]. While these algorithms cannot cope with the full expressiveness of CEP languages (e.g., neglecting Kleene closure and negation operators), they discover queries built of sequence operators, data predicates, and time windows. Below, we summarize the main ideas of the IL-Miner [35], as it supports a more expressive query model compared to iCEP [53]. Note that there are also techniques for complex event pattern discovery supporting logic-based approaches, such as OLED [43, 44] for event calculus.

In essence, one first tries to identify data predicates that refer to single events and appear to be relevant for query discovery. To this end, it is assessed, for which atomic predicates, an event that satisfies it can be found in any of the training sequences. Subsequently, the identified predicates, coined in relevant event templates, are used to abstract the training sequences, yielding sequences of templates. Based thereon, standard frequent sequence mining is conducted, which yields sequences of templates. Intuitively, each such sequence provides a skeleton for the construction of a set of event queries. In a next step, to construct a query, each sequence of templates is linked to the events of the original training sequences. From the obtained sequences of events, further data predicates (e.g., those referring to more than one event in a pattern) and a time window are extracted.

Running such a discovery algorithm in practice may lead to a very large number of discovered event queries, even when ignoring obvious redundancies such as inclusion dependencies between queries. To cope with that phenomenon and to enable manual inspection of the discovered queries, the result may be filtered to obtain a representative set of queries. To this end, one may leverage clustering techniques based on syntactic and semantic similarity measures for event queries.

Finally, it should be highlighted that various kinds of domain knowledge, if available, may be incorporated to increase the effectiveness and efficiency of event query discovery. For instance, knowing that particular attributes distinguish instances of a process, then any discovered event query should contain the respective data predicates that correlate events based on these attributes.

4.4 Diagnostics for Event Query Matches

This section discusses how to gain diagnostic insights into specific process behavior. If a query, derived using the methods described in Section 4.2 and 4.3, matches, the result needs to be interpreted as a violation of some normative behavior. Specifically, diagnostics for these matches are important, as fine-granular queries may lead to an overload of triggered alerts for certain deviations from the expected process behavior. For example, the occurrence of a single out-of-order event may trigger a large amount of *strict order* violations, even though these violations all stem from the same source.

To avoid such an overload, monitoring alerts can be filtered by identifying the earliest indicator of non-compliant behavior in a set of compliance violations. This identification requires a different approach depending on the type of violation. Here, we illustrate a diagnostics technique for the violation of two constraint types for which the respective queries have been introduced in Section 4.2: exclusiveness and ordering constraints. Afterwards, we discuss how these techniques may be lifted beyond the control-flow perspective.

4.4.1 Diagnostics for Exclusiveness Violations

We first consider violations that stem from the exclusiveness monitoring query set Q_+. Let $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ be the sequence of recorded events in a process instance and $+$ be the exclusiveness relation, as introduced in Section 4.2, of the respective process model. Then, we derive the set of violations V_+^n at the time event a_n is recorded as follows.

$$V_+^n = \{(a_x, a_y) \in + \mid a_y = a_n \ \wedge \ a_x \in \sigma\}.$$

Trigger Violation. A single event may cause multiple exclusiveness violations. Given such a set of violations, V_+^n, the trigger of these violations is the violation that relates to the earliest event in the sequence of recorded events $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$. Therefore, the trigger refers to the earliest event that implies that the event a_n was not allowed. We define a function *trigger* to extract the trigger for the most recent violations with respect to a_n as follows.

$$\mathit{trigger}(V_+^n) = (a_x, a_y) \in V_+^n \ \text{ such that } \ \forall\, (a_k, a_l) \in V_+^n \, [\, x \leq k \,].$$

We illustrate the introduced concept for the process from Figure 2. Assume that the activities a and d have been executed, when we observe an event that signals the completion of activity c, i.e., we recorded $\sigma = \langle a, d, c \rangle$. The exclusiveness monitoring query set Q_+ matches and identifies two violations, $V_+^3 = \{(a,c),(d,c)\}$. The violation of exclusiveness for a and c is the trigger, since a was the first activity to complete in the recorded event sequence, i.e., $\mathit{trigger}(V_+^3) = (a,c)$.

Consecutive Violation. The identification of a trigger for a set of violations triggered by a single event is the first step to structure the feedback on violations. Once a violation is identified, subsequent events may result in violations that logically follow

from the violations already observed. For a trigger violation (a_x, a_y), consecutive violations (a_p, a_q) are characterized by the fact that (1) either a_x with a_p and a_y with a_q are not conflicting or (2) this non-conflicting property is observed for a_x with a_q and a_y with a_p. Further, we have to consider the case that potentially it holds $a_p = a_x$ or $a_p = a_y$. Consecutive violations are recorded, but explicitly marked once they are observed. Given a trigger (a_x, a_y) of exclusiveness violations, we define the set of consecutive violations by a function *consec*.

$$\begin{aligned} \mathit{consec}_+(a_x, a_y) = \{(a_p, a_q) \in + \mid ((a_x = a_p \vee a_x \not\nless a_p) \wedge (a_q = a_y \vee a_y \not\nless a_q)) \\ \vee\ ((a_y = a_p \vee a_y \not\nless a_p) \wedge (a_x = a_q \vee a_x \not\nless a_q))\}. \end{aligned}$$

Reconsider the process from Figure 2 and the recorded event sequence $\sigma = \langle a, d, c \rangle$. Now, assume a subsequent recording of event e. The exclusiveness monitoring query set Q_+ matches and we extract a set of violations $V_+^4 = \{(c, e)\}$ (in addition to V_+^3) with the trigger $\mathit{trigger}(V_+^4) = (c, e)$. Apparently, this violation follows directly from the violations identified when event c has been recorded, because e is expected to occur subsequent to d. This is captured by our notion of consecutive violations for the previously identified trigger (a, c). Since c and e are expected to be exclusive and it holds $a_y = c = a_p$ and $a \not\nless e$, we observe that $(c, e) \in \mathit{conseq}(a, c)$. Hence, the exclusiveness violation (c, e) would be reported as a consecutive violation of the previous violations identified by their trigger $\mathit{trigger}(V_+^3) = (a, c)$.

Further, assume that the next recorded event is b, so that $\sigma = \langle a, d, c, e, b \rangle$ and $V_+^5 = \{(a, b), (c, b)\}$. Then, both violations represent a situation that does not follow logically from violations observed so far, i.e., they are non-consecutive and reported as independent violations to the analyst. Still, the feedback is structured as we identify a trigger as $\mathit{trigger}(V_+^5) = (a, b)$, since event a has occurred before event c.

4.4.2 Diagnostics for Order Violations

Now, we consider violations that stem from the order monitoring query set $Q_{\rightsquigarrow}$. Let $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ be a sequence of recorded events in a process instance and $\rightsquigarrow$ be the strict order relation, as defined in Section 4.2, of the respective process model. Let $\rightsquigarrow^{-1}$ be the inverse relation of the strict order relation, $(a_x \rightsquigarrow a_y) \Leftrightarrow (a_y \rightsquigarrow^{-1} a_x)$. Then, we derive the set of violations $V_{\rightsquigarrow}^n$ at the time event a_n is recorded as follows.

$$V_{\rightsquigarrow}^n = \{(a_x, a_y) \in \rightsquigarrow^{-1} \mid a_y = a_n \wedge a_x \in \sigma\}.$$

Trigger Violation. For this set of violations, it may be the case that a single event causes multiple order violations. Given a set of order violations $V_{\rightsquigarrow}^n$, the trigger is the violation that relates to the earliest event in the sequence of recorded events $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$. Again, we define a function to extract this trigger.

$$\mathit{trigger}(V_{\rightsquigarrow}^n) = (a_x, a_y) \in V_{\rightsquigarrow}^n \text{ such that } \forall\, (a_k, a_l) \in V_{\rightsquigarrow}^n\ [\, x \leq k \,].$$

For illustration, consider the example of Figure 2 and a sequence of recorded events $\sigma = \langle a,d,h,k \rangle$. Now, e is completed, which points to a violation of the order constraint between e and h as well as between e and k. The idea is to report the earliest event in the execution sequence, which was supposed to be executed after e. Then, the violation $trigger(V^5_{\rightsquigarrow}) = (h,e)$ is identified as the trigger, since h has been the first event in this case.

Consecutive Violation. As for exclusiveness constraint violations, we also define consecutive violations. These include violations from subsequent events that logically follow from violations observed earlier. For a trigger (a_x,a_y), consecutive violations (a_p,a_q) are characterized by the fact that either a_x with a_p and a_y with a_q are in strict order, or a_x with a_q and a_y with a_p, respectively. Taking into account that it may hold $a_p = a_x$ or $a_p = a_y$, we lift the function *consec* to triggers of strict order violations. Given such a trigger (a_x,a_y), it is defined as follows.

$$\begin{aligned} consec(a_x,a_y) = \{(a_p,a_q) \in {\rightsquigarrow}^{-1} \mid & ((a_x = a_p \vee a_x \rightsquigarrow a_p) \wedge (a_q = a_y \vee a_y \rightsquigarrow a_q)) \\ \vee & \ ((a_y = a_p \vee a_y \rightsquigarrow a_p) \wedge (a_x = a_q \vee a_x \rightsquigarrow a_q))\}. \end{aligned}$$

Consider the example of Figure 2 and the sequence $\sigma = \langle a,d,h,k,e \rangle$, which resulted in $trigger(V^5_{\rightsquigarrow}) = (h,e)$. Now, g is observed, which violates the order with h and k as monitored by the order monitoring query $Q_{\rightsquigarrow}$. Apparently, this violation follows from the earlier violations. It is identified as a consecutive violation for the previous trigger (h,e), since h is in both violations and e and g are not conflicting in terms of order, i.e., $(h,g) \in consec(h,e)$. Since $k \rightsquigarrow^{-1} g$, $h \rightsquigarrow k$, and $e \rightsquigarrow g$, this also holds for the second violation. Hence, violations (h,g) and (h,k) are reported as consecutive violations.

4.4.3 Diagnostics on the Violation Context

Next to the identification of particular events that resulted in compliance violations, it is possible to assess whether there are connections between violations and their occurrence context as reflected in data attributes associated with process instances. That is, the goal is to check for attribute values that differentiate cases with the violation from cases without the violation.

As an illustration, consider that an organization monitors the time it takes from receiving a lead to sending out a quote in their Lead-to-Quote process, particularly, they want to identify where this takes more than two weeks. By considering the data values associated with events, the organization may discover that these violations occur for a specific context, e.g., for orders stemming from quote requests originating from a particular *country* and which are related to a specific *order type*. If such information can be identified, this provides valuable diagnostic insights into the factors that are correlated with the observed delays, which may guide the efforts to resolve the issue.

The automated identification of the context of specific violations can be achieved by applying *classification* techniques on an annotated set of process instances, where

two classes are used to distinguish among instances with violations and instances without. Classifiers that produce human interpretable output, such as *decision trees*, are particularly useful for this setting. These techniques can produce clear rules indicating in which contexts monitoring violations have been observed.

5 Discussion

This chapter outlined how Complex Event Processing methods can be leveraged for process querying that works on event-based representations of processes. In particular, we discussed how CEP can be embedded in the process querying context, by relating CEP methods to process querying use cases and the Process Querying Framework. In this way, we showed that event-based process querying can be used both for online querying, through the analysis of event streams, as well as for offline querying, by replaying event logs containing static event data. Moreover, we highlighted design choices of CEP models, as those govern which of these models may be appropriate for process querying in a specific application context.

We argued that using CEP methods for event-based process querying faces several challenges, which may be addressed by four essential techniques: First, event-activity correlation is used to link elements from a process specification (e.g., process model activities) to types of observed events, which represents a fundamental requirement for a broad range of analysis techniques. Second, model-based query generation can be used to automatically derive monitoring queries from process models, which enables the identification of deviations between modeled and recorded behavior. Third, we also discussed how such monitoring queries can be based on historic event data, in case a suitable process model is not available. Fourth, we discussed techniques to analyze the matches of event queries, which can be used to gain diagnostic insights into specific process behavior. Most importantly, this includes the identification of trigger violations that represent the earliest signal of deviating process behavior.

While the aforementioned shows that CEP methods have a range of applications in the context of process querying, open research questions remain. In relation to the techniques discussed in Section 4, we identify the main open directions, as follows:

- Although different techniques have been developed for event-activity correlation, it has been recognized that this problem in practice has only been resolved through the development of probabilistic techniques, which are unable to provide correlations in a deterministic manner. As a result, analysis techniques should be adapted to this so-called mapping uncertainty [3].
- Model-based query generation allows for the automatic establishment of monitoring queries from process models. However, given a sufficiently complex model, the amount of generated queries can quickly become unmanageable. Recognizing that some process violations will have a greater impact on organizations than others, the question of how to identify the semantically most relevant monitoring queries remains open.

- The problem of specifically weighting the relevance of particular queries is also present when discovering queries automatically from labeled event data. Moreover, common discovery techniques do show scalability issues. Hence, optimizations of discovery algorithms that incorporate domain knowledge on the process are a promising direction for future research.
- Having exemplified the potential of diagnostics for the matches of event queries, future work should focus on recognizing the complimentary nature of control-flow and data-aware techniques. A combination of these types of techniques would be able, for instance, to identify particular control flow-related deviations that only occur if the throughput time or other data-based values are above a particular threshold. This is currently not achieved, because the control flow and data perspectives are only considered in isolation.

References

1. Van der Aa, H.: Comparing and Aligning Process Representations: Foundations and Technical Solutions, vol. 323. Springer (2018)
2. Van der Aa, H., Gal, A., Leopold, H., Reijers, H.A., Sagi, T., Shraga, R.: Instance-based process matching using event-log information. In: International Conference on Advanced Information Systems Engineering, pp. 283–297. Springer (2017)
3. Van der Aa, H., Leopold, H., Reijers, H.A.: Checking process compliance on the basis of uncertain event-to-activity mappings. In: International Conference on Advanced Information Systems Engineering, pp. 79–93. Springer (2017)
4. Van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016). DOI 10.1007/978-3-662-49851-4. URL https://doi.org/10.1007/978-3-662-49851-4
5. Van der Aalst, W.M.P., van Hee, K.M., van der Werf, J.M.E.M., Kumar, A., Verdonk, M.: Conceptual model for online auditing. Decision Support Systems **50**(3), 636–647 (2011). DOI 10.1016/j.dss.2010.08.014. URL https://doi.org/10.1016/j.dss.2010.08.014
6. Van der Aalst, W.M.P., Weijters, T., Maruster, L.: Workflow mining: Discovering process models from event logs. IEEE Transactions on Knowledge & Data Engineering (9), 1128–1142 (2004)
7. Adi, A., Etzion, O.: Amit - the situation manager. VLDB J. **13**(2), 177–203 (2004). DOI 10.1007/s00778-003-0108-y. URL https://doi.org/10.1007/s00778-003-0108-y
8. Agirre, E., Alfonseca, E., Hall, K., Kravalova, J., Paşca, M., Soroa, A.: A study on similarity and relatedness using distributional and wordnet-based approaches. In: Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp. 19–27. Association for Computational Linguistics (2009)
9. Akidau, T., Bradshaw, R., Chambers, C., Chernyak, S., Fernández-Moctezuma, R., Lax, R., McVeety, S., Mills, D., Perry, F., Schmidt, E., Whittle, S.: The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. PVLDB **8**(12), 1792–1803 (2015). DOI 10.14778/2824032.2824076. URL http://www.vldb.org/pvldb/vol8/p1792-Akidau.pdf
10. Algergawy, A., Nayak, R., Saake, G.: Element similarity measures in xml schema matching. Information Sciences **180**(24), 4975–4998 (2010)

11. Anicic, D., Rudolph, S., Fodor, P., Stojanovic, N.: Stream reasoning and complex event processing in ETALIS. Semantic Web **3**(4), 397–407 (2012). DOI 10.3233/SW-2011-0053. URL https://doi.org/10.3233/SW-2011-0053
12. Appel, S., Kleber, P., Frischbier, S., Freudenreich, T., Buchmann, A.P.: Modeling and execution of event stream processing in business processes. Inf. Syst. **46**, 140–156 (2014). DOI 10.1016/j.is.2014.04.002. URL https://doi.org/10.1016/j.is.2014.04.002
13. Arasu, A., Babcock, B., Babu, S., Cieslewicz, J., Datar, M., Ito, K., Motwani, R., Srivastava, U., Widom, J.: STREAM: the stanford data stream management system. In: Garofalakis et al. [34], pp. 317–336. DOI 10.1007/978-3-540-28608-0_16. URL https://doi.org/10.1007/978-3-540-28608-0_16
14. Artikis, A., Margara, A., Ugarte, M., Vansummeren, S., Weidlich, M.: Complex event recognition languages: Tutorial. In: Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, DEBS 2017, Barcelona, Spain, June 19-23, 2017, pp. 7–10. ACM (2017). DOI 10.1145/3093742.3095106. URL http://doi.acm.org/10.1145/3093742.3095106
15. Artikis, A., Sergot, M.J., Paliouras, G.: An event calculus for event recognition. IEEE Trans. Knowl. Data Eng. **27**(4), 895–908 (2015). DOI 10.1109/TKDE.2014.2356476. URL https://doi.org/10.1109/TKDE.2014.2356476
16. Backmann, M., Baumgrass, A., Herzberg, N., Meyer, A., Weske, M.: Model-driven event query generation for business process monitoring. In: A. Lomuscio, S. Nepal, F. Patrizi, B. Benatallah, I. Brandic (eds.) Service-Oriented Computing - ICSOC 2013 Workshops - CCSA, CSB, PASCEB, SWESE, WESOA, and PhD Symposium, Berlin, Germany, December 2-5, 2013. Revised Selected Papers, *Lecture Notes in Computer Science*, vol. 8377, pp. 406–418. Springer (2013). DOI 10.1007/978-3-319-06859-6_36. URL https://doi.org/10.1007/978-3-319-06859-6_36
17. Baier, T., Di Ciccio, C., Mendling, J., Weske, M.: Matching events and activities by integrating behavioral aspects and label analysis. Software & Systems Modeling pp. 1–26 (2017)
18. Baier, T., Mendling, J., Weske, M.: Bridging abstraction layers in process mining. Information Systems **46**, 123–139 (2014)
19. Baier, T., Rogge-Solti, A., Weske, M., Mendling, J.: Matching of events and activities-an approach based on constraint satisfaction. In: IFIP Working Conference on The Practice of Enterprise Modeling, pp. 58–72. Springer (2014)
20. Barga, R.S., Goldstein, J., Ali, M.H., Hong, M.: Consistent streaming through time: A vision for event stream processing. In: CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings, pp. 363–374. www.cidrdb.org (2007). URL http://cidrdb.org/cidr2007/papers/cidr07p42.pdf
21. Bose, R.J.C., Van Der Aalst, W.M., Zliobaite, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE transactions on neural networks and learning systems **25**(1), 154–171 (2014)
22. Brenna, L., Demers, A.J., Gehrke, J., Hong, M., Ossher, J., Panda, B., Riedewald, M., Thatte, M., White, W.M.: Cayuga: a high-performance event processing engine. In: C.Y. Chan, B.C. Ooi, A. Zhou (eds.) Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007, pp. 1100–1102. ACM (2007). DOI 10.1145/1247480.1247620. URL http://doi.acm.org/10.1145/1247480.1247620
23. Burattin, A., Sperduti, A., van der Aalst, W.M.P.: Control-flow discovery from event streams. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2014, Beijing, China, July 6-11, 2014, pp. 2420–2427. IEEE (2014). DOI 10.1109/CEC.2014.6900341. URL https://doi.org/10.1109/CEC.2014.6900341
24. Carmona, J., van Dongen, B.F., Solti, A., Weidlich, M.: Conformance Checking - Relating Processes and Models. Springer (2018). DOI 10.1007/978-3-319-99414-7. URL https://doi.org/10.1007/978-3-319-99414-7
25. Cugola, G., Margara, A.: TESLA: a formally defined event specification language. In: J. Bacon, P.R. Pietzuch, J. Sventek, U. Çetintemel (eds.) Proceedings of the Fourth ACM International

Conference on Distributed Event-Based Systems, DEBS 2010, Cambridge, United Kingdom, July 12-15, 2010, pp. 50–61. ACM (2010). DOI 10.1145/1827418.1827427. URL http://doi.acm.org/10.1145/1827418.1827427

26. Cugola, G., Margara, A.: Processing flows of information: From data stream to complex event processing. ACM Comput. Surv. **44**(3), 15:1–15:62 (2012). DOI 10.1145/2187671.2187677. URL http://doi.acm.org/10.1145/2187671.2187677
27. del-Río-Ortega, A., Resinas, M., Cabanillas, C., Cortés, A.R.: On the definition and design-time analysis of process performance indicators. Inf. Syst. **38**(4), 470–490 (2013). DOI 10.1016/j.is.2012.11.004. URL https://doi.org/10.1016/j.is.2012.11.004
28. Ding, L., Chen, S., Rundensteiner, E.A., Tatemura, J., Hsiung, W., Candan, K.S.: Runtime semantic query optimization for event stream processing. In: G. Alonso, J.A. Blakeley, A.L.P. Chen (eds.) Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, Mexico, pp. 676–685. IEEE Computer Society (2008). DOI 10.1109/ICDE.2008.4497476. URL https://doi.org/10.1109/ICDE.2008.4497476
29. Dumas, M., García-Bañuelos, L., Polyvyanyy, A., Yang, Y., Zhang, L.: Aggregate quality of service computation for composite services. In: P.P. Maglio, M. Weske, J. Yang, M. Fantinato (eds.) Service-Oriented Computing - 8th International Conference, ICSOC 2010, San Francisco, CA, USA, December 7-10, 2010. Proceedings, *Lecture Notes in Computer Science*, vol. 6470, pp. 213–227 (2010). DOI 10.1007/978-3-642-17358-5_15. URL https://doi.org/10.1007/978-3-642-17358-5_15
30. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018). DOI 10.1007/978-3-662-56509-4. URL https://doi.org/10.1007/978-3-662-56509-4
31. EsperTech: Esper documentation (2018). URL http://www.espertech.com/esper/esper-documentation/
32. Etzion, O., Niblett, P.: Event Processing in Action. Manning Publications Company (2010). URL http://www.manning.com/etzion/
33. Fettke, P., Loos, P., Zwicker, J.: Business process reference models: Survey and classification. In: C. Bussler, A. Haller (eds.) Business Process Management Workshops, BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS, Nancy, France, September 5, 2005, Revised Selected Papers, vol. 3812, pp. 469–483 (2005). DOI 10.1007/11678564_44. URL https://doi.org/10.1007/11678564_44
34. Garofalakis, M.N., Gehrke, J., Rastogi, R. (eds.): Data Stream Management - Processing High-Speed Data Streams. Data-Centric Systems and Applications. Springer (2016). DOI 10.1007/978-3-540-28608-0. URL https://doi.org/10.1007/978-3-540-28608-0
35. George, L., Cadonna, B., Weidlich, M.: Il-miner: Instance-level discovery of complex event patterns. PVLDB **10**(1), 25–36 (2016). DOI 10.14778/3015270.3015273. URL http://www.vldb.org/pvldb/vol10/p25-weidlich.pdf
36. Grez, A., Riveros, C., Ugarte, M.: A Formal Framework for Complex Event Processing. In: P. Barcelo, M. Calautti (eds.) 22nd International Conference on Database Theory (ICDT 2019), *Leibniz International Proceedings in Informatics (LIPIcs)*, vol. 127, pp. 5:1–5:18. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2019). DOI 10.4230/LIPIcs.ICDT.2019.5. URL http://drops.dagstuhl.de/opus/volltexte/2019/10307
37. Hassani, M., Siccha, S., Richter, F., Seidl, T.: Efficient process discovery from event streams using sequential pattern mining. In: IEEE Symposium Series on Computational Intelligence, SSCI 2015, Cape Town, South Africa, December 7-10, 2015, pp. 1366–1373. IEEE (2015). DOI 10.1109/SSCI.2015.195. URL https://doi.org/10.1109/SSCI.2015.195
38. He, Y., Barman, S., Naughton, J.F.: On load shedding in complex event processing. In: N. Schweikardt, V. Christophides, V. Leroy (eds.) Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014., pp. 213–224. OpenProceedings.org (2014). DOI 10.5441/002/icdt.2014.23. URL https://doi.org/10.5441/002/icdt.2014.23

39. Hinze, A., Voisard, A.: EVA: an event algebra supporting complex event specification. Inf. Syst. **48**, 1–25 (2015). DOI 10.1016/j.is.2014.07.003. URL https://doi.org/10.1016/j.is.2014.07.003
40. Hirzel, M., Baudart, G.: Stream Processing Languages and Abstractions, pp. 1–8. Springer International Publishing, Cham (2018). DOI 10.1007/978-3-319-63962-8_260-1. URL https://doi.org/10.1007/978-3-319-63962-8_260-1
41. Hirzel, M., Soulé, R., Schneider, S., Gedik, B., Grimm, R.: A catalog of stream processing optimizations. ACM Comput. Surv. **46**(4), 46:1–46:34 (2013). DOI 10.1145/2528412. URL http://doi.acm.org/10.1145/2528412
42. ISO/IEC: Information technology - Database languages - SQL Technical Reports - Part 5: Row Pattern Recognition in SQL. TR 19075-5
43. Katzouris, N., Artikis, A., Paliouras, G.: Online learning of event definitions. TPLP **16**(5-6), 817–833 (2016). DOI 10.1017/S1471068416000260. URL https://doi.org/10.1017/S1471068416000260
44. Katzouris, N., Artikis, A., Paliouras, G.: Parallel online event calculus learning for complex event recognition. Future Generation Comp. Syst. **94**, 468–478 (2019). DOI 10.1016/j.future.2018.11.033. URL https://doi.org/10.1016/j.future.2018.11.033
45. Kunze, M., Weske, M.: Behavioural Models - From Modelling Finite Automata to Analysing Business Processes. Springer (2016). DOI 10.1007/978-3-319-44960-9. URL https://doi.org/10.1007/978-3-319-44960-9
46. Li, M., Mani, M., Rundensteiner, E.A., Lin, T.: Complex event pattern detection over streams with interval-based temporal semantics. In: D.M. Eyers, O. Etzion, A. Gal, S.B. Zdonik, P. Vincent (eds.) Proceedings of the Fifth ACM International Conference on Distributed Event-Based Systems, DEBS 2011, New York, NY, USA, July 11-15, 2011, pp. 291–302. ACM (2011). DOI 10.1145/2002259.2002297. URL http://doi.acm.org/10.1145/2002259.2002297
47. Lin, D.: An information-theoretic definition of similarity. In: ICML, vol. 98, pp. 296–304 (1998)
48. Liu, M., Li, M., Golovnya, D., Rundensteiner, E.A., Claypool, K.T.: Sequence pattern query processing over out-of-order event streams. In: Y.E. Ioannidis, D.L. Lee, R.T. Ng (eds.) Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China, pp. 784–795. IEEE Computer Society (2009). DOI 10.1109/ICDE.2009.95. URL https://doi.org/10.1109/ICDE.2009.95
49. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.P.: Compliance monitoring in business processes: Functionalities, application, and tool-support. Inf. Syst. **54**, 209–234 (2015). DOI 10.1016/j.is.2015.02.007. URL https://doi.org/10.1016/j.is.2015.02.007
50. Mannhardt, F., De Leoni, M., Reijers, H.A., Van Der Aalst, W.M., Toussaint, P.J.: From low-level events to activities-a pattern-based approach. In: International Conference on Business Process Management, pp. 125–141. Springer (2016)
51. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. Computing **98**(4), 407–437 (2016). DOI 10.1007/s00607-015-0441-1. URL https://doi.org/10.1007/s00607-015-0441-1
52. Mans, R., van der Aalst, W.M.P., Vanwersch, R.J.B.: Process Mining in Healthcare - Evaluating and Exploiting Operational Healthcare Processes. Springer Briefs in Business Process Management. Springer (2015). DOI 10.1007/978-3-319-16071-9. URL https://doi.org/10.1007/978-3-319-16071-9
53. Margara, A., Cugola, G., Tamburrelli, G.: Learning from the past: automated rule generation for complex event processing. In: U. Bellur, R. Kothari (eds.) The 8th ACM International Conference on Distributed Event-Based Systems, DEBS '14, Mumbai, India, May 26-29, 2014, pp. 47–58. ACM (2014). DOI 10.1145/2611286.2611289. URL http://doi.acm.org/10.1145/2611286.2611289
54. Mei, Y., Madden, S.: Zstream: a cost-based query processor for adaptively detecting composite events. In: U. Çetintemel, S.B. Zdonik, D. Kossmann, N. Tatbul (eds.) Proceedings of the

ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009, pp. 193–206. ACM (2009). DOI 10.1145/1559845.1559867. URL http://doi.acm.org/10.1145/1559845.1559867
55. Oliner, A., Ganapathi, A., Xu, W.: Advances and challenges in log analysis. Communications of the ACM **55**(2), 55–61 (2012)
56. Polyvyanyy, A., Armas-Cervantes, A., Dumas, M., García-Bañuelos, L.: On the expressive power of behavioral profiles. Formal Asp. Comput. **28**(4), 597–613 (2016). DOI 10.1007/s00165-016-0372-4. URL https://doi.org/10.1007/s00165-016-0372-4
57. Polyvyanyy, A., Ouyang, C., Barros, A., van der Aalst, W.M.P.: Process querying: Enabling business intelligence through query-based process analytics. Decision Support Systems **100**, 41–56 (2017). DOI 10.1016/j.dss.2017.04.011. URL https://doi.org/10.1016/j.dss.2017.04.011
58. Polyvyanyy, A., Weidlich, M., Conforti, R., Rosa, M.L., ter Hofstede, A.H.M.: The 4c spectrum of fundamental behavioral relations for concurrent systems. In: G. Ciardo, E. Kindler (eds.) Application and Theory of Petri Nets and Concurrency - 35th International Conference, PETRI NETS 2014, Tunis, Tunisia, June 23-27, 2014. Proceedings, *Lecture Notes in Computer Science*, vol. 8489, pp. 210–232. Springer (2014). DOI 10.1007/978-3-319-07734-5_12. URL https://doi.org/10.1007/978-3-319-07734-5_12
59. Ray, M., Lei, C., Rundensteiner, E.A.: Scalable pattern sharing on event streams. In: F. Özcan, G. Koutrika, S. Madden (eds.) Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016, pp. 495–510. ACM (2016). DOI 10.1145/2882903.2882947. URL http://doi.acm.org/10.1145/2882903.2882947
60. Rogge-Solti, A., Kasneci, G.: Temporal anomaly detection in business processes. In: S.W. Sadiq, P. Soffer, H. Völzer (eds.) Business Process Management - 12th International Conference, BPM 2014, Haifa, Israel, September 7-11, 2014. Proceedings, *Lecture Notes in Computer Science*, vol. 8659, pp. 234–249. Springer (2014). DOI 10.1007/978-3-319-10172-9_15. URL https://doi.org/10.1007/978-3-319-10172-9_15
61. Sadiq, S., Governatori, G., Namiri, K.: Modeling control objectives for business process compliance. In: International conference on business process management, pp. 149–164. Springer (2007)
62. Senderovich, A., Rogge-Solti, A., Gal, A., Mendling, J., Mandelbaum, A.: The road from sensor data to process instances via interaction mining. In: International Conference on Advanced Information Systems Engineering, pp. 257–273. Springer (2016)
63. Teymourian, K., Rohde, M., Paschke, A.: Fusion of background knowledge and streams of events. In: F. Bry, A. Paschke, P.T. Eugster, C. Fetzer, A. Behrend (eds.) Proceedings of the Sixth ACM International Conference on Distributed Event-Based Systems, DEBS 2012, Berlin, Germany, July 16-20, 2012, pp. 302–313. ACM (2012). DOI 10.1145/2335484.2335517. URL http://doi.acm.org/10.1145/2335484.2335517
64. Wang, J., Han, J.: BIDE: efficient mining of frequent closed sequences. In: Z.M. Özsoyoglu, S.B. Zdonik (eds.) Proceedings of the 20th International Conference on Data Engineering, ICDE 2004, 30 March - 2 April 2004, Boston, MA, USA, pp. 79–90. IEEE Computer Society (2004). DOI 10.1109/ICDE.2004.1319986. URL https://doi.org/10.1109/ICDE.2004.1319986
65. Weidlich, M., Mendling, J., Weske, M.: Efficient consistency measurement based on behavioral profiles of process models. IEEE Trans. Software Eng. **37**(3), 410–429 (2011). DOI 10.1109/TSE.2010.96. URL https://doi.org/10.1109/TSE.2010.96
66. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Causal behavioural profiles–efficient computation, applications, and evaluation. Fundamenta Informaticae **113**(3-4), 399–435 (2011)
67. Weidlich, M., Ziekow, H., Gal, A., Mendling, J., Weske, M.: Optimizing event pattern matching using business process models. IEEE Trans. Knowl. Data Eng. **26**(11), 2759–2773 (2014). DOI 10.1109/TKDE.2014.2302306. URL https://doi.org/10.1109/TKDE.2014.2302306
68. Weidlich, M., Ziekow, H., Mendling, J., Günther, O., Weske, M., Desai, N.: Event-based monitoring of process execution violations. In: S. Rinderle-Ma, F. Toumani, K. Wolf (eds.)

Business Process Management - 9th International Conference, BPM 2011, Clermont-Ferrand, France, August 30 - September 2, 2011. Proceedings, *Lecture Notes in Computer Science*, vol. 6896, pp. 182–198. Springer (2011). DOI 10.1007/978-3-642-23059-2_16. URL https://doi.org/10.1007/978-3-642-23059-2_16

69. White, W.M., Riedewald, M., Gehrke, J., Demers, A.J.: What is "next" in event processing? In: L. Libkin (ed.) Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China, pp. 263–272. ACM (2007). DOI 10.1145/1265530.1265567. URL http://doi.acm.org/10.1145/1265530.1265567
70. Yan, X., Han, J., Afshar, R.: Clospan: Mining closed sequential patterns in large datasets. In: D. Barbará, C. Kamath (eds.) Proceedings of the Third SIAM International Conference on Data Mining, San Francisco, CA, USA, May 1-3, 2003, pp. 166–177. SIAM (2003). DOI 10.1137/1.9781611972733.15. URL https://doi.org/10.1137/1.9781611972733.15
71. Yujian, L., Bo, L.: A normalized levenshtein distance metric. IEEE transactions on pattern analysis and machine intelligence **29**(6), 1091–1095 (2007)
72. Zeng, K., Yang, M., Mozafari, B., Zaniolo, C.: Complex pattern matching in complex structures: The xseq approach. In: C.S. Jensen, C.M. Jermaine, X. Zhou (eds.) 29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013, pp. 1328–1331. IEEE Computer Society (2013). DOI 10.1109/ICDE.2013.6544936. URL https://doi.org/10.1109/ICDE.2013.6544936
73. Zhang, H., Diao, Y., Immerman, N.: On complexity and optimization of expensive queries in complex event processing. In: C.E. Dyreson, F. Li, M.T. Özsu (eds.) International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014, pp. 217–228. ACM (2014). DOI 10.1145/2588555.2593671. URL http://doi.acm.org/10.1145/2588555.2593671

Acronyms

BPM Business Process Management
CEP Complex Event Processing
CQL Continuous Query Language
CRM Customer Relationship Management
EPL Esper Pattern Language
ERP Enterprise Resource Planning
PQF Process Querying Framework

Index