Event-based Monitoring of Process Execution Violations

M. Weidlich[1], H. Ziekow[2⋆], J. Mendling[3], O. Günther[3], M. Weske[1], N. Desai[4]

[1] Hasso-Plattner-Institute, University of Potsdam, Germany
{matthias.weidlich,weske}@hpi.uni-potsdam.de
[2] AGT Germany, Germany
hziekow@agtgermany.com
[3] Humboldt-Universität zu Berlin, Germany
{jan.mendling,guenther}@wiwi.hu-berlin.de
[4] IBM India Research Labs, India
Nirmit123@in.ibm.com

Abstract. Process-aware information systems support business operations as they are typically defined in a normative process model. Often these systems do not directly execute the process model, but provide the flexibility to deviate from the normative model. This paper proposes a method for monitoring control-flow deviations during process execution. Our contribution is a formal technique to derive monitoring queries from a process model, such that they can be directly used in a complex event processing environment. Furthermore, we also introduce an approach to filter and aggregate query results to provide compact feedback on deviations. Our techniques is applied in a case study within the IT service industry.

1 Introduction

Process-aware information systems are increasingly used to execute and control business processes [1]. Such systems provide a more general support to process execution in comparison to classical workflow systems as they do not necessarily need to enforce a normative process model. This notion of process-awareness rather relates to guiding the execution of an individual case instead of restricting the behaviour to a narrow set of sequences. In this way, process-aware information systems often provide the flexibility to deviate from the normative process model [2].

While the flexibility provided by process-aware information systems is often a crucial benefit, it also raises the question in how far deviations from the normative behaviour can be efficiently identified. For instance, a process analyst may explicitly want to be notified in a timely manner when actions are taken on a particular case that deviate from the standard procedure. Such reactive mechanisms are typically referred to as process monitoring. They build on the identification of specific types of events, which are analysed and processed to yield business-relevant insights.

In recent years, techniques for complex event processing have been introduced for identifying non-trivial patterns in event sequences and for triggering appropriate reactions. Such patterns are formulated as queries over event sequences. While certain non-temporal requirements upon a process can be easily formulated as complex event

⋆ The main part of the work was done while at Humboldt-Universität zu Berlin.

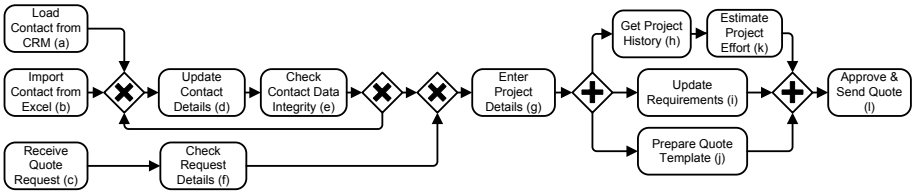

Fig. 1. Example Lead-to-Quote process modelled in BPMN

queries, e.g., a four-eyes principle on a pair of activities, it is a non-trivial problem to encode the control flow of a normative process model as a set of event queries that pinpoint violations. However, such queries are highly relevant, e.g., for monitoring IT incident resolution for which best practice process models exist, but are not enforced by a workflow system. In this paper, we address this problem from two angles. First, we define a technique to generate monitoring queries from a process model. To this end, we leverage the concept of a behavioural profile. Second, we support the presentation of the query results in such a way that root causes of a rule violation are directly visible to the process analyst. To achieve this, we develop an approach to filter and aggregate query results. In this way, we aim to contribute towards a better integration of process monitoring and complex event processing techniques.

The paper is structured as follows. Section 2 introduces the formal foundations of our approach including complex event processing and behavioural profiles. Section 3 defines a transformation technique to generate queries from process models. Section 4 identifies three classes of execution violations and how they are handled to provide meaningful feedback to the process analyst. Section 5 applies our techniques to a case study in the IT service domain. Section 6 discusses related work, before Section 7 concludes the paper.

2 Preliminaries

This section presents preliminaries for our investigations. Section 2.1 discusses process models. Then, Section 2.2 gives background information on complex event processing. Finally, Section 2.3 introduces casual behavioural profiles as an abstraction of the behaviour of a process model.

2.1 Process Models

In large organisations, process models are used to describe business operations. As such, they formalise requirements that have to be considered during the development of process-aware information systems [1]. Process models are also directly used to foster process execution when a workflow engine utilises a model to enforce the specified behaviour. In both cases, the process model has a normative role. It precisely defines the intended behaviour of the process.

Abstracting from differences in expressiveness and notation of common process description languages, a process model is a graph consisting of nodes and edges. The former represent activities and control flow routing nodes – split and merge nodes that

implement execution logic beyond simple sequencing of activities. The latter implement the control flow structure of the process. Figure 1 depicts an example process model in BPMN, which we use in this paper to illustrate our concepts. It depicts a Lead-to-Quote process. The process starts with an import of contact data or with the reception of a request for quote. In the former case, the contact details are updated. This step may be repeated if data integrity constraints are not met. Then, the quote is prepared by first entering prospective project details and then conducting an effort estimation, updating the requirements, and preparing the quote template. These steps are done concurrently. Finally, the quote is approved and submitted.

2.2 Complex Event Processing

Complex event processing (CEP) refers to technologies that support processing of real time data that occur as events. The goal of CEP is to detect situations of interest in real time as well as to initiate corresponding actions. The notion of an event slightly differs throughout literature. Etzion and Niblett refer to an event as "occurrence within a particular system or domain" [3]. Examples for such events are manifold. They can be as simple as a sensor reading or as complex as the prediction of an supply bottleneck.

Data models for events vary between systems and application contexts. For instance, some systems encode events as tuples while others follow an object oriented model. Further, events may be modelled explicitly with a duration or only with an occurrence time stamp. We assume events to be represented as tuples with the following structure:

$$event = (eventID, caseID, activity, timeStamp, < e_{val} >)$$

Here, $eventID$ is a unique identifier of an event, $caseID$ a unique identifier for the process instance, $activity$ the observed activity in the process, $timeStamp$ the time of the completion time of the activity, and $< e_{val} >$ a collection of attribute-value pairs. Our approach is concerned with monitoring single instances. Still, the unique identifier for process instances is required to correlate events to instances. If this information is not available in a system, techniques to correlate events [4, 5] may be applied to discover events related to a single instance.

Intuitively, a CEP system acts like an inverted database. A database stores data persistently and processes events in an ad-hoc manner. In contrast, a CEP system stores standing queries and evaluates these queries as new event data arrives. This principle is implemented in different systems to support real time queries over event data, e.g., [6–9].

Queries are formulated in terms of dedicated query languages. Rule based languages following an ECA structure and SQL-extensions are two main categories for query language styles. A key element in many languages is a pattern, which defines relations between events. Typical operations for pattern definition are conjunction, disjunction, negation, and a sequence operator that defines a specific order of event occurrences. Temporal constraints are supported by constructs for defining time windows. Below we show a query example written in the SQL-like language of ESPER [9]. The query matches if the activity 'Update Contact Details' is followed by the activity 'Enter Project Details' in the same process instance.

```
select * from pattern
[every a=ObservationEvent(a.Observation='Update Contact Details') ->
b=ObservationEvent(b.Observation='Enter Project Details',b.caseID=a.caseID)]
```

In this paper, we use a more abstract notation for queries. The notation focuses on the query parts that are relevant to our approach and is intended to support an intuitive understanding. With a small letter, we denote the atomic query for events reflecting the execution of the corresponding activity. A query a evaluates true, if activity a was completed. For simplicity, we say that 'an event a occurred'. With an expression in capital letters we denote sets of atomic event queries. Such an expression evaluates true, if any atomic query in the set evaluates true. We build more complex queries using the operators and, not, seq, and $within$. The and operator represents Boolean conjunction. The term $and(a, b)$ evaluates true if the two atomic queries a and b evaluate true, i.e., if events a and b occurred. The not operator represents Boolean negation. The seq operator implements sequencing, i.e., $seq(a, b)$ is true if query a is true first and at a later time query b evaluates true. Constraints on event attributes are not explicitly modelled. However, we implicitly assume that all queries are limited to events with the same $caseID$. This ensures that all matched events belong to the same process instance. To constrain time, we use the operator $within$. It takes a query q and a time window t as input and evaluates to true if q is true when constrained to events that are not more than t apart in time. Since the operators and, not, and seq are defined over Boolean values, we can nest them, e.g., $and(a, seq(b, c))$ with a, b, and c being atomic queries.

As CEP engines typically can support these operations, our approach is not limited to any specific implementation. For instance, the defined operators can be evaluated by an extended automaton, in which state transitions correspond to the occurrence of events [10].

2.3 Causal Behavioural Profiles

Our approach to monitoring of process instances exploits the behavioural constraints that are imposed by a normative process model. We use the notion of a causal behavioural profile [11] to capture these constraints. A causal behavioural profile provides an abstraction of the behaviour defined by a process model. It captures behavioural characteristics by relations on the level of activity pairs. The order of potential execution of activities is captured by three relations. Two activities are either in strict order, exclusive to each other, or in interleaving order. These relations follow from the possible execution sequences, alias traces, of the process model.

- The *strict order* relation, denoted by $\rightsquigarrow$, holds between two activities x and y, if x may happen before y, but not vice versa. That is, in all traces comprising both activities, x will occur before y.
- The *exclusiveness* relation, denoted by $+$, holds for two activities, if they never occur together in any process trace.
- The *interleaving order* relation, denoted by $||$, holds for two activities x and y, if x may happen before y and y may also happen before x. Interleaving order can be interpreted as the absence of any specific order constraint for two activities.

The causal behavioural profile also comprises a *co-occurrence* relation, denoted by $\gg$, to capture occurrence dependencies between activities. Co-occurrence holds between two activities x and y, if any complete trace, from the initial to the final state of the process, that contains activity x contains also activity y. For the activities of the model in Fig. 1, for instance, activities a and b are exclusive. Activities e and g are in strict order. The

concurrent execution of activities h and i yields interleaving order in the behavioural profile. As a self-relation, activities show either exclusiveness (if they are executed at most once, e.g., $a + a$) or interleaving order (if they may be repeated, e.g., $d||d$). As an example for co-occurrence, we observe that every trace containing a also contains d, but not vice versa, $a \gg d$ and $d \not\gg a$. Causal behavioural profiles are computed efficiently for process models as the one shown in Fig. 1, see [11, 12].

The causal behavioural profile is a behavioural abstraction. For instance, cardinalities of activity execution are not captured. Still, findings from experiments with process variants [12] suggest that these profiles capture a significant share of the behavioural characteristics of a process model. Although our approach builds on the causal behavioural profile, for brevity, we use the term behavioural profile in the remainder of this paper.

3 Monitoring based on Event Queries

We describe the architecture of our approach to run-time monitoring of process execution in Section 3.1. Then, we focus on the derivation of monitoring queries in Section 3.2.

3.1 Architecture

Our approach facilitates the automatic generation of monitoring queries from process models as well as preprocessing of the query results for presentation. Conceptually this involves four main steps:

1. Extraction of behavioural profiles from process models
2. Generation of complex event queries
3. Running queries over process events
4. Applying filters and aggregates on detected deviations

Figure 2 provides an overview of the system architecture. We assume that the monitoring system is provided with events that reflect the completion of process activities. This might require some preprocessing to deduce the completion of an activity from available input data, e.g. as in RFID based process monitoring [13]. However, we make no assumptions on the specific event sources. Instead, events can have multiple origins, such as an enterprise service bus, a manufacturing execution system, or sensor middleware.

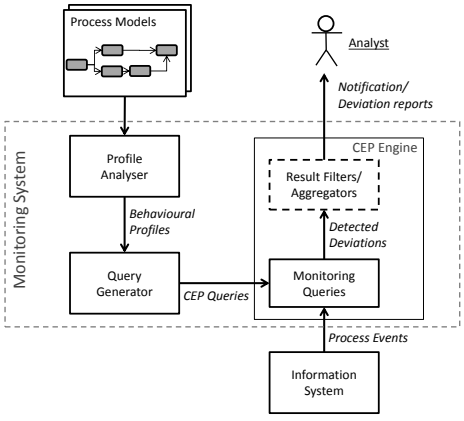

Fig. 2. Architecture overview

Further, we assume the availability of process models as input. From these process models we extract behavioural profiles to obtain behavioural constraints on event occurrences in model conformant process instances. Given the behavioural profiles, we generate complex event queries that target violations on a fine grained level and reveal where a process instance deviates from the model.

To run the queries, we assume the utilisation of a CEP engine such as provided in [6–9]. However, our approach abstracts from specific implementation details. Instead, we describe our solution using general concepts that can be realised in different systems. The CEP engine continuously receives the process events. It then matches the events against the monitoring queries to detect deviations from the model. The system can create an alert for each detected deviation. We enhance this mechanism with filters and aggregations to condense reports and avoid information overload.

3.2 Derivation of Event Queries from Process Models

To leverage the information provided by the behavioural profile of a process model for monitoring, we generate a set of event queries from the profile. These queries follow directly from the relations of the behavioural profile and relate to *exclusiveness*, *order*, and *co-occurrence* constraints. As discussed in Section 2.3, interleaving order between two activities can be seen as the absence of an ordering constraint. Therefore, there is no need to consider this relation when monitoring the accuracy of process execution.

Exclusiveness: Exclusiveness between activities as defined by the relation of the behavioural profile has to be respected in the events signalling activity execution. To identify violations to these constraints, a query matches joint execution of two exclusive activities. Note that the exclusiveness relation also indicates whether an activity is executed at most once (exclusiveness as a self-relation). Given the behavioural profile $\mathcal{B} = \{\rightsquigarrow, +, ||, \gg\}$ of a process model, we define the *exclusiveness query set* as follows.

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

For the model depicted in Fig. 1, 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. We assume those to be filtered by optimisation techniques of the query processing.

The described queries are sufficient for detecting exclusiveness violations. However, they provide no means to discard partial query results if no violations occur. Such a mechanism is crucial for the applicability of our approach as an overload of the event processor has to be avoided. To optimise resource utilisation, we present modified versions of the query generation that incorporate termination in correct process instances. We consider two options for terminating non matching queries. One option is using timeouts. Setting an appropriate timeout t requires background knowledge on the monitored processes and should be defined by a domain expert. Setting the timeout too low may result in missing constraint violations. To avoid this risk, the set of terminating activities

END of the process may be leveraged. This option solely relies on the process model. We construct a set of optimised queries as follows.

$$Q_{+optimised} = \bigcup_{(a_1,a_2)\in+} \{within(and(and(a_1,a_2), not(END \setminus \{a_1,a_2\})), t)\}.$$

Here, $not(END)$ evaluates true if no event in the set END occurred with END being the set of events representing terminating activities. The operator $within(a,t)$ limits the evaluation of the query a to the relative time frame t. That is, query a is only evaluated for events that are not more than t apart in time.

Order: Violations of the order of activity execution is queried for with an *order query set*. It matches pairs of activities for which the order of execution is not in line with the behavioural profile. Given the behavioural profile $\mathcal{B} = \{\rightsquigarrow, +, ||, \gg\}$ of a process model, we define this set of queries as

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

For the model in Fig. 1, the order query set contains the following queries, $Q_{\rightsquigarrow} = \{seq(d,a), seq(e,a), seq(g,a), \ldots, seq(l,k)\}$.

Similar to queries for exclusiveness, queries for order violation do not match in correct process instances. For optimisation we again incorporate constructs for termination based on timeouts t or terminating activities END of the process.

$$Q_{\rightsquigarrow optimised} = \bigcup_{(a_1,a_2)\in\rightsquigarrow} \{within(and(seq(a_2,a_1), not(END \setminus \{a_1,a_2\})), t)\}.$$

Co-occurrence: For exclusiveness and order of activity execution, the respective queries follow directly from the relations of the behavioural profile. Co-occurrence constraints, in turn, cannot be verified in this way. Co-occurrence constraints are violated not by the *presence* of a certain activity execution, but by its *absence*. Therefore, a constraint violation materialises only at the completion of a process instance. Only then it becomes visible which activities are missing the observed execution sequence even though they should have been executed. We construct a corresponding query set as follows:

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

For a running process instance, the question whether an activity is missing cannot be answered definitely. Nevertheless, the strict order relation can be exploited to identify states of a process instance, which may evolve into a co-occurrence constraint violation. Taking up the idea of conformance measuring based on behavioural profiles [14], we 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. As the co-occurrence constraint is not yet violated definitely,

we refer to these queries as co-occurrence warning queries. Note that, although the co-occurrence constraint is not violated definitely when the query matches, the respective instance is fraudulent. Even if the co-occurrence constraint is satisfied at a later stage, the instance will show an order violation. In other words, if the co-occurrence warning query matches, a constraint violation is detected, but it is still unclear whether the order or the co-occurrence constraint is violated. Given the behavioural profile $\mathcal{B} = \{\rightsquigarrow, +, ||, \gg\}$ of a process model, the *co-occurrence warning query set* is defined as follows.

$$Q_{\gg warning} = \bigcup_{((a_1,a_2)\in(\gg\cap\rightsquigarrow)) \,\wedge\, ((a_2,a_3)\in\rightsquigarrow)} \{and(and(a_1, a_3), not(a_2))\}.$$

For the example model depicted in Fig. 1, there exist the following co-occurrence warning queries, $Q_{\gg warning} = \{and(and(a, g), not(d)), \ldots, and(and(h, l), not(k))\}$. The first term refers to the co-occurrence constraint between activities a and d, i. e., if the contact is loaded from the CRM, the contact details have to be updated. The execution of activity g, entering the project details, is used to judge on the progress of the processing. The query matches, if we observe the execution of activities a and g, but there has not been any execution of activity d. If this activity is executed at a later stage, the co-occurrence constraint would be satisfied, whereas the order constraint between activities d and g would be violated.

Du to the lack of space we do not provide optimised versions of the queries for co-occurrence violations. However, optimisation can be done along the lines of the optimisations for queries for exclusiveness and order violations.

4 Feedback on Behavioural Deviations

The queries which we derived in the previous section allow us to monitor the behaviour of a process instance on a fine-granular level. On the downside, fine-granular queries may result in multiple alerts when an activity is performed at an unexpected stage of the business process. For instance, consider a sequence of activities a, d, e, which are all exclusive to c, as in our example in Fig. 3. When now c is executed after the other activities have already been completed, we obtain three alerts. The idea of the concepts presented in this section is to filter these alerts. First, we identify the root cause for the set of violations. A root cause refers to the responsible event occurrence, which implies that we observe a violation at a later stage. Second, we check whether a certain violation is an implication of an earlier violation. Sections 4.1 to 4.3 define according filtering and reporting strategies for the constraint violations detected by the queries introduced in the previous section.

4.1 Exclusiveness Violations

We first consider violations that stem from the exclusiveness monitoring query Q_+. Let $\sigma = \langle a_1, a_2, \ldots, a_n\rangle$ be the sequence of recorded events for a process instance and $\mathcal{B} = \{\rightsquigarrow, +, ||, \gg\}$ the behavioural profile of the respective process model. Then, we derive the set of violations V_+^n at the time event a_n is recorded as follows.

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

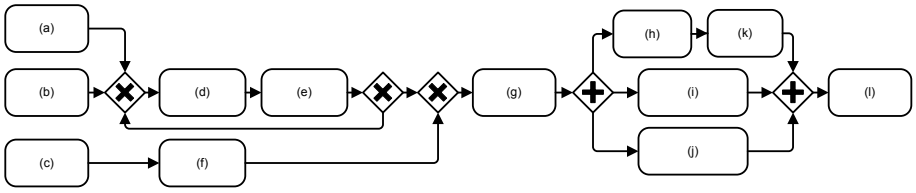

Fig. 3. Example process revisited

Root Cause. A single event may cause multiple exclusiveness violations. Given a set of exclusiveness violations V_+^n, the root cause is the violation that relates to the earliest event in the sequence of recorded events $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$. In this way, the root cause refers to the earliest event that implies that the latest event would not be allowed anymore. We define a function $root$ to extract the root cause for the most recent violations with respect to a_n as follows.

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

We illustrate the introduced concept for the process from Fig. 3. 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 Q_+ matches and identifies two violations, $V_+^3 = \{(a, c), (d, c)\}$. The violation of exclusiveness for a and c is the root cause, since a was the first activity to complete in the recorded event sequence, i.e., $root(V_+^3) = (a, c)$.

Consecutive Violation. The identification of a root cause 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 observed already. For a root cause (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 root cause (a_x, a_y) of exclusiveness violations, we define the set of consecutive violations by a function $consec$.

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

Consider the process from Fig. 3 and the recorded event sequence $\sigma = \langle a, d, c \rangle$. Now assume a subsequent recording of event e. The exclusiveness monitoring query Q_+ matches and we extract a set of violations $V_+^4 = \{(c, e)\}$ with the root cause $root(V_+^4) = (c, e)$. 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 root cause (a, c). Since c and e are expected to be exclusive and it holds $a_y = c = a_p$ and $a \not= e$, we observe that $(c, e) \in conseq(a, c)$. Hence, the exclusiveness violation (c, e) would

be reported as a consecutive violation of the previous violations identified by their root cause $root(V_+^3) = (a, c)$.

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 root cause as $root(V_+^5) = (a, b)$, since event a has occurred before event c.

4.2 Order Violations

Now, we consider violations that stem from order monitoring query $Q_{\rightsquigarrow}$. Let $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ be the sequence of recorded events for a process instance and $\mathcal{B} = \{\rightsquigarrow, +, ||, \gg\}$ the behavioural profile 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.

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

Root Cause. Also 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 root cause 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 $root$ to extract the root cause.

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

For illustration, consider the example of Fig. 3 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 $root(V_{\rightsquigarrow}^5) = (h, e)$ is the root cause, 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 root cause (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 root causes of strict order violations. Given such a root cause (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 Fig. 3 and the sequence $\sigma = \langle a, d, h, k, e \rangle$, which resulted in $root(V_{\rightsquigarrow}^5) = (h, e)$. 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 root cause (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.

4.3 Occurrence Violations

Finally, we consider violations that relate to occurrence dependencies between executions of activities. As discussed in Section 3.2, the absence of an required activity execution materialises only once the process instance finished execution (query $Q_{\gg detection}$). To indicate problematic processing in terms of potential occurrence violations, we introduced a set of co-occurrence warning queries $Q_{\gg warning}$. These queries may match repeatedly during execution of a process instance, so that we focus on filtering violations that relate to these queries. Once again, note that the queries $Q_{\gg warning}$ detect violations and only the type of violation (co-occurrence constraint violation or order violation) cannot be determined definitely.

Let $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ be the sequence of recorded events for a process instance and $\mathcal{B} = \{\rightsquigarrow, +, ||, \gg\}$ the behavioural profile of the respective process model. Further, let A denote the set of events related to activities of the process model. Then, we derive the set of violations $V^n_{\gg}$ at the time event a_n is recorded as follows.

$$V^n_{\gg} = \{(a_x, a_y) \in (A \times A) \mid a_y = a_n \ \wedge \ a_x \in \sigma \\ \wedge \ \exists\, a \in A \,[\, a \notin \sigma \ \wedge \ a_x \gg a \ \wedge \ a_x \rightsquigarrow a \ \wedge \ a \rightsquigarrow a_y \,]\}.$$

Such a violation comprises the event that requests the presence of the missing event along with the event signalling that the missing event should have been observed already. This information on a violation is needed to structure the feedback for the analyst. In contrast to exclusiveness and order violations, the actual violated constraint is not part of the violation captured in $V^n_{\gg}$. We define an auxiliary function $miss$ to characterise the actual missing events for a violation $(a_x, a_y) \in V^n_{\gg}$ at the time event a_n is recorded.

$$miss(a_x, a_y) = \{a \in A \mid a \notin \sigma \ \wedge \ a_x \gg a \ \wedge \ a \rightsquigarrow a_y\}.$$

Root Cause. Also here, a single event may cause multiple violations. Given a set of violations $V^n_{\gg}$, the root cause is the violation that relates to latest event in the sequence of recorded events $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$. By reporting the latest event, we identify the smallest region in which an event is missing. We define $root$ as follows.

$$root(V^n_{\gg}) = (a_x, a_y) \in V^n_{\gg} \ \text{ such that } \ \forall\, (a_k, a_l) \in V^n_{\gg} \,[\, x \geq k \,].$$

For illustration, consider the example of Fig. 3 and assume that a sequence of events $\sigma = \langle a, d \rangle$ has been recorded. Now, h is completed, which points to a missing execution of e and g, $V^3_{\gg} = \{(a, h), (d, h)\}$ and $miss(a, h) = miss(d, h) = \{e, g\}$. The idea is to report the latest event in the execution sequence that is triggering the violation, in our case event d. Then, the violation for d and h is the root cause, $root(V^3_{\gg}) = (d, h)$.

Consecutive Violation. For a root cause (a_x, a_y) of occurrence violations, consecutive violations (a_p, a_q) are characterized by the fact that there exists an intermediate missing event at the time event a_n is recorded which (1) resulted in the root cause, and (2) led to a violation observed when an event a_m is added later, $m \geq n$.

$$consec(a_x, a_y) = \{(a_p, a_q) \in (A \times A) \mid \exists\, a \in miss(a_x, a_y) \,[\, a_p \gg a \ \wedge \ a \rightsquigarrow a_q \,]\}.$$

Consider the sequence $\sigma = \langle a, d, h \rangle$ for our example in Fig. 3, which resulted in $root(V^3_{\gg}) = (d, h)$. Now, k is observed, which yields $V^4_{\gg} = \{(a, k), (d, k)\}$. For both violations there are missing intermediate events, namely e and g, that resulted in the root cause (d, h). Hence, (a, k) and (d, k) are reported as consecutive violations.

5 Case Study: SIMP

As an evaluation, we applied our approach in a case study. This case study builds on a replay of real-world log data within a prototypical implementation of our approach. In this way, we are able to draw conclusions on the number and types of detected control-flow deviations. We also illustrate how feedback on these deviations is given. First, Section 5.1 gives background information on the investigated business process. Second, Section 5.2 presents the results on detected deviations.

5.1 Background

We applied our techniques to the Security Incident Management Process (SIMP), an issue management process. This process is run in one of IBM's global service delivery centres. This centre provides infrastructure management and technical support to customers. Figure 4 gives an overview of the SIMP as a BPMN diagram. The process is initiate by creating an issue. Then, issue details may be updated and a resolution plan is created. Subsequently, change management activities may be conducted, which may be followed by monitoring of target dates and risk management tasks. In the mean time, a customer extension may be conducted. Finally, a proposal to close is issued. If accepted, the issue is closed. If rejected, either a new proposal to close the issue is created or the whole processing starts over again.

The SIMP is standardised and documented within the respective IBM global service delivery centre, but it is not explicitly enforced by workflow technology. Therefore, employees may deviate from the predefined processing. The latter is tracked by a proprietary tool, in which an employee submits the execution of a certain activity.

For the SIMP, we obtained 852 execution sequences, aka cases, that have been recorded within nearly five years. SIMP instances are long-running, up to several months. We analysed the degree of conformance of these cases with the predefined behaviour shown in Fig. 4 in prior work, along with an evaluation of frequencies for particular violations [14]. Here, we apply the techniques introduced in this paper for identifying execution violations and providing aggregated feedback to all SIMP cases. This allows for evaluating the number and type of detected violations in a fine-granular manner. In addition, we evaluate our techniques for providing aggregated feedback in a real-world setting. This analysis offers the basis to judge on the applicability of our approach for online monitoring of the SIMP.

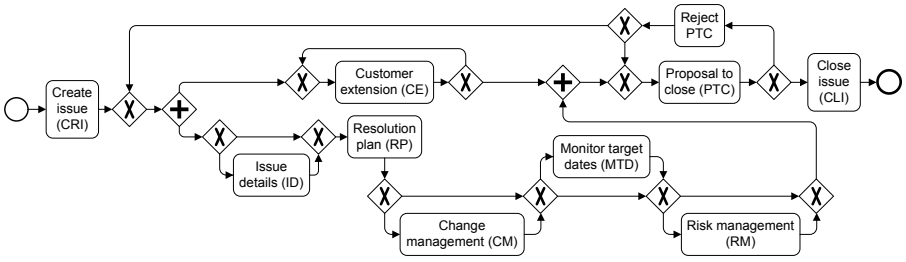

Fig. 4. BPMN model of the Security Incident Management Process (SIMP), see also [14]

Table 1. Identified violations in 852 cases of the SIMP

	Exclusiveness	Order	Occurrence	All Violations
Overall	194	173	247	614
Avg per Case (StDev)	0.23 (0.36)	0.20 (0.40)	0.29 (0.49)	0.72 (1.10)
Max per Case	6	74	4	77
Non-Consecutive (%)	179 (92.27%)	14 (8.09%)	134 (54.25%)	327 (53.26%)
# Fraudulent Cases (%)	179 (21.01%)	7 (0.82%)	134 (15.73%)	202 (23.71%)

5.2 Results

As a first step, we evaluated how often the queries derived from the process model match for the 852 cases of the SIMP. Table 1 gives a summary of the results. We detected 614 constraint violations in all cases. On average, a case violates 0.72 behavioural constraints. Further, we see that the violations are rather homogeneously distributed over the three types of constraints. However, when evaluating how the different types of violations are distributed over the cases, we observe differences. Exclusiveness and occurrence constraint violations typically appear isolated, at most 6 or 4 of such constraints are violated within a single case and the violations are spread over 179 or 134 cases, respectively. Order constraint violations, in turn, are observed in solely 7 cases. However, those cases violate a large number of constraints, at most 74.

The latter clearly indicates a need to provide the feedback in a structured way. To this end, we also evaluated our techniques for filtering consecutive violations. Table 1 lists how many non-consecutive violations have been detected for all types of violations. We see that exclusiveness constraint violations are virtually always independent, whereas occurrence and particularly order constraint violations are often consecutive. Further, Fig. 5 illustrates the percentage of violations that are filtered relative to the number of detected violations for a certain case. Clearly, for those cases that show a large number of violations, the vast majority of violations is reported as consecutive violations. Hence, a process analyst is faced only with a very small number of root causes for each case.

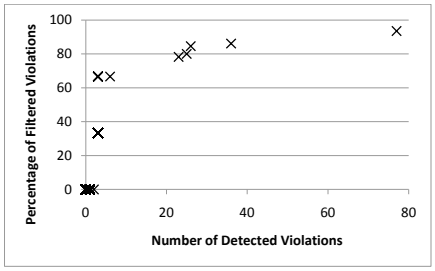

Fig. 5. Percentage of filtered violations relative to the number of detected violations for single cases of the SIMP

Finally, we turn the focus on the SIMP case with the most violations to illustrate our findings. This case showed 77 violations, 74 of them relate to order constraints. Analysis of the case reveals that after an issue has been processed, its close is proposed and then conducted, see also Fig. 4. However, the issue seems to have been reopened, which led to the execution of various activities, e.g., a customer extensions have been conducted. All these executions violate order constraints. Still, the 74 order violations are traced back

to three root causes, so that 95.95% of the violations are filtered. This illustrates that our approach is able to identify significant problems during processing and that feedback is given in a structured and aggregated manner.

6 Related Work

The research presented in this paper relates to work on business process monitoring (also called business activity monitoring), complex event processing, and transactional workflows.

The monitoring of workflows based on a common interface had been early recognized by the workflow management coalition as an important issue [15], though there did not emerge a standard which was accepted by vendors. Several works discuss requirements upon a data format for audit trails [16] and process log data [17], and how these process-related event logs can be analysed a posteriori. Different approaches have been developed for measuring conformance of logs with process models [18] and for identifying root causes for deviation from normative behaviour [14]. Only recently, this research area has progressed towards online auditing [19] and monitoring of choreographies [20]. This paper complements these works with a approach to immediately feed back cases of non-conformance in a compact and filtered way.

The approach presented in this paper also relates to the research domain of complex event processing. Several research projects addressed different aspects of event processing technologies [21–24]. Several works in this area provide solutions for query optimisation, e.g. considering characteristics of wireless sensor networks [24, 25], availability of devices [26], resource consumption [27], or size of intermediate result sets [28]. Up until now, research that combines complex event processing and process management is scarce and only focused on query optimisation [29]. In this regard, our contribution is an approach to integrate process monitoring with complex event processing technology.

In this paper, we consider a normative process model, which has to be monitored for behavioural conformance of event sequences. This problem is closely related to general properties of transactional workflows, which guarantee the adherence to specified behaviour by moving back to a consistent state when undesirable event sequences occur (see [30–33]). Transactional concepts defined for workflows have been adopted in web services technologies [34, 35] and incorporated in industry standard such as *WS-AtomicTransaction* [36]. In particular, transactional properties have to be considered for service composition [37]. From the perspective of these works, our approach provides immediate feedback on events that need to be rolled back or compensated.

7 Conclusion

In this paper, we have addressed the problem of monitoring conformance of execution sequences with normative process models in a complex event processing environment. Our contribution is a technique to generate queries from a process model based on its behavioural profile. Further, we present query results in a compact way – root causes of a violation are directly visible to the process analyst. Our approach has been implemented

and evaluated in a case study in the IT service industry. The results demonstrate that even large sets of violations can be traced back to a small number of root causes.

In future research, we aim to enhance our techniques in two directions. Behavioural profiles provide only an abstracted view on constraints within loops. We are currently studying concepts, first, for capturing the fact that certain activities have to be executed before an activity can be repeated. Second, behavioural profiles currently miss a notion of cardinality constraint, when for instance an activity a has to be repeated exactly as many times as b is executed. Beyond these conceptual challenges, we plan to conduct additional case studies with industry partners to further investigate the efficiency of our filtering techniques for different types of processes. Further, the scalability of our approach using a dedicated CEP environment has to be investigated.

References

1. Dumas, M., ter Hofstede, A., van der Aalst, W., eds.: Process Aware Information Systems: Bridging People and Software Through Process Technology. Wiley Publishing (2005)
2. Pesic, M., van der Aalst, W.: A declarative approach for flexible business processes management. In Eder, J., Dustdar, S., eds.: BPM Workshops. LNCS 4103, Springer (2006) 169–180
3. Etzion, O., Niblett, P.: Event Processing in Action. Manning, Stammford, CT, USA (2010)
4. Ferreira, D.R., Gillblad, D.: Discovering process models from unlabelled event logs. In Dayal, U., Eder, J., Koehler, J., Reijers, H.A., eds.: BPM. LNCS 5701, Springer (2009) 143–158
5. Musaraj, K., Yoshida, T., Daniel, F., Hacid, M.S., Casati, F., Benatallah, B.: Message correlation and web service protocol mining from inaccurate logs. In: ICWS, IEEE Computer Society (2010) 259–266
6. Jacobsen, H.A., Muthusamy, V., Li, G.: The padres event processing network: Uniform querying of past and future events. it - Information Technology **51**(5) (2009) 250–261
7. Gyllstrom, D., Wu, E., Chae, H.J., Diao, Y., Stahlberg, P., Anderson, G.: SASE Complex event processing over streams. In: Int. Conf. on Innovative Data Systems Research (2007)
8. Madden, S., Franklin, M.J.: Fjording the stream: An architecture for queries over streaming sensor data. In: International Conference on Data Engineering. (2002)
9. EsperTech: Esper - Complex Event Processing http://esper.codehaus.org, as of March 2011.
10. Brenna, L., Gehrke, J., Hong, M., Johansen, D.: Distributed event stream processing with non-deterministic finite automata. In: DEBS, ACM (2009) 1–12
11. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Efficient computation of causal behavioural profiles using structural decomposition. In: Petri Nets. LNCS 6128, Springer (2010) 63–83
12. Weidlich, M., Mendling, J., Weske, M.: Efficient consistency measurement based on behavioral profiles of process models. IEEE Trans. Software Eng. **37**(3) (2011) 410–429
13. Bornhövd, C., Lin, T., Haller, S., Schaper, J.: Integrating automatic data acquisition with business processes experiences with sap's auto-id infrastructure. In: Int. Conference on Very Large Data Bases, VLDB Endowment (2004) 1182–1188
14. Weidlich, M., Polyvyanyy, A., Desai, N., Mendling, J., Weske, M.: Process compliance analysis based on behavioural profiles. Inf. Syst. **36**(7) (2011) 1009–1025
15. Hollingsworth, D.: The Workflow Reference Model. TC00-1003 Issue 1.1, Workflow Management Coalition (24 November 1994)
16. Muehlen, M.: Workflow-based Process Controlling. Foundation, Design, and Implementation of Workflow-driven Process Information Systems. Logos, Berlin (2004)

17. van der Aalst, W., Reijers, H., Weijters, A., van Dongen, B., Alves de Medeiros, A., Song, M., Verbeek, H.: Business process mining: An industrial application. Information Systems **32**(5) (2007) 713–732
18. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1) (2008) 64–95
19. 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) (2011) 636–647
20. Wetzstein, B., Karastoyanova, D., Kopp, O., Leymann, F., Zwink, D.: Cross-organizational process monitoring based on service choreographies. In Shin, S.Y., Ossowski, S., Schumacher, M., Palakal, M.J., Hung, C.C., eds.: Proceedings of ACM SAC, ACM (2010) 2485–2490
21. Abadi, D.J., Ahmad, Y., Balazinska, M., Cetintemel, U., Cherniack, M., Hwang, J.H., Lindner, W., Maskey, A., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y., Zdonik, S.: The design of the Borealis stream processing engine. In: Int. Conf. on Innovative Data Systems Research. (2005) 277–289
22. 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. Technical report, Department of Computer Science, Stanford University (2004)
23. Chandrasekaran, S., Cooper, O., A.Deshpande, Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., Shah, M.A.: Telegraphcq: Continuous dataflow processing for an uncertain world. In: Int. Conf. on Innovative Data Systems Research. (2003)
24. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: TinyDB: An acquisitional query processing system for sensor networks. ACM Transactions on Database Systems (TODS) **30**(1) (2005) 122–173
25. Yao, Y., Gehrke, J.: The cougar approach to in-network query processing in sensor networks. In: Proc. of the Intl. ACM Conf. on Management of Data (SIGMOD). (2002)
26. Srivastava, U., Munagala, K., Widom, J.: Operator placement for in-network stream query processing. In: Proc. of PODS, ACM (2005)
27. Schultz-Møller, N.P., Migliavacca, M., Pietzuch, P.: Distributed complex event processing with query rewriting. In: DEBS, Proceedings, ACM (2009) 1–12
28. Wu, E., Diao, Y., Rizvi, S.: High-performance complex event processing over streams. In: SIGMOD, ACM (2006) 407–418
29. Weidlich, M., Ziekow, H., Mendling, J.: Optimising Complex Event Queries over Business Processes using Behavioural Profiles. In: BPM Workshops. LNBIP 66, Springer (2010) 743–754
30. Garcia-Molina, H., Salem, K.: Sagas. ACM SIGMOD Record **16**(3) (1987) 249–259
31. Georgakopoulos, D., Hornick, M.F., Sheth, A.P.: An overview of workflow management: From process modeling to workflow automation infrastructure. Distributed and Parallel Databases **3**(2) (1995) 119–153
32. Alonso, G., Agrawal, D., Abbadi, A.E., Kamath, M., Günthör, R., Mohan, C.: Advanced transaction models in workflow contexts. In Su, S.Y.W., ed.: ICDE, IEEE Computer Society (1996) 574–581
33. Dayal, U., Hsu, M., Ladin, R.: Business Process Coordination: State of the Art, Trends, and Open Issues. In: Int. Conference on Very Large Databases, Morgan Kaufmann (2001) 3–13
34. Papazoglou, M.P.: Web services and business transactions. WWW **6**(1) (2003) 49–91
35. Alonso, G., Casati, F., Kuno, H.A., Machiraju, V.: Web Services - Concepts, Architectures and Applications. Data-Centric Systems and Applications. Springer (2004)
36. F. Cabrera et al.: Web services atomic transaction (ws-atomictransaction). Technical report, IBM, Microsoft,BEA (2005)
37. Zhang, L.J.: Editorial: Context-aware application integration and transactional behaviors. IEEE T. Services Computing **3**(1) (2010) 1