Monitoring Security Compliance of Critical Processes

Roland Rieke[*][†], Jürgen Repp[†], Maria Zhdanova[†], and Jörn Eichler[‡]
[*]Philipps-Universität Marburg, Germany
[†]Fraunhofer SIT, Darmstadt, Germany
Email:{roland.rieke,juergen.repp,maria.zhdanova}@sit.fraunhofer.de
[‡]Fraunhofer AISEC, Munich, Germany
Email: joern.eichler@aisec.fraunhofer.de

***Abstract*—Enforcing security in process-aware information systems at runtime requires the monitoring of systems' operation using process information. Analysis of this information with respect to security and compliance aspects is growing in complexity with the increase in functionality, connectivity, and dynamics of process evolution. To tackle this complexity, the application of models is becoming standard practice. Considering today's frequent changes to processes, model-based support for security and compliance analysis is not only needed in pre-operational phases but also at runtime.**

This paper presents an approach to support evaluation of the security status of processes at runtime. The approach is based on operational formal models derived from process specifications and security policies comprising technical, organizational, regulatory and cross-layer aspects. A process behavior model is synchronized by events from the running process and utilizes prediction of expected close-future states to find possible security violations and allow early decisions on countermeasures. The applicability of the approach is exemplified by a misuse case scenario from a hydroelectric power plant.

***Keywords*-predictive security analysis; process behavior analysis; security modeling and simulation; security monitoring; critical infrastructures; security information and event management.**

I. Introduction

Electronic business processes contribute significantly to the performance of today's enterprises and their correct execution is vital for many companies. Automated enactment of business processes applying Information Technology (IT) does not only bring competitive advantages but induces higher security risks. A new Internet security threat report [1] states a more than 81% surge in malicious attacks including sophisticated targeted attacks. Yet, existing Business Process Management (BPM) methodologies often neglect security and dependability objectives [2]. At the same time, business processes become more complex encompassing a wide range of heterogeneous systems and applications and undergo continuous changes to sustain business competitiveness [3]. Another dimension is added by the inter-connection of business processes with modern automated management systems that support remote control of multiple infrastructures. Thus, cross-layer connections between high-level business processes, organizational processes, and low-level technical processes controlling sensors and actuators in cyber-physical systems emerge. Increasing complexity and changeability complicates analysis of distinctive process properties demanding frequent adjustments of process models to address changing business needs [4]. This involves not only functional correctness of a process model, but also related compliance and security features. Hard-coded controls can restrain flexibility required to ensure adequate formal representation of evolving processes [5], [6].

We present an approach for predictive security analysis at runtime, which allows to add security requirements regarding process behavior during execution without the need to modify the corresponding process model. In doing so, we do not intend to diminish the significance of security-by-design. Our work is aimed as a critical add-on in order to address the dynamics of electronic business processes. Based on close-future behavior models computed on-the-fly from process specifications, we demonstrate early detection of deviations of process execution from expected behavior which can be caused by attacker intervention. We propose a new method for security analysis at runtime exploiting process behavior models, which enables on-the-fly security compliance checks and prediction of close-future violations of security requirements. The proposed integration of simulation and runtime monitoring allows for early security warnings and predictive alarms on possible security critical states in close future. In order to demonstrate how our model-based runtime analysis is applied, we have chosen processes from a hydroelectric power plant in a dam that was analyzed in a European research project [7]. We describe an implementation of our approach and provide results of evaluation of specific aspects, such as effects of the number of security requirements, different abstraction levels and the variation of prediction depths.

Section II of this paper gives an overview of our approach for predictive security analysis at runtime. Section III introduces the operational process model, the close-future behavior model, and the synchronization with the running process. Section IV presents the security model applied at runtime to identify security relevant states. Section V provides an example for the runtime analysis of security requirements from a hydroelectric power plant. Section VI describes the prototype implementation and evaluation results. Section VII reviews related work and Section VIII presents conclusions and further research.

II. Predictive Security Analysis at Runtime

In this section we introduce a new model-based approach for Predictive Security Analysis at Runtime (PSA@R). Our approach integrates formal process modeling with simulation of (close-future) process behavior triggered by real-time data. Process behavior models are used to identify and predict violations of security requirements during process execution.

In PSA@R the operation of a system or a system of systems is observed analyzing events received from this system. PSA@R is not executed by this *observed system* but rather by an *observing system* such as a Security Information and Event Management (SIEM) system. It is presupposed here that the observing system itself is trustworthy. A SIEM system can be easier protected against attacks than the system under observation. Regarding the observed system it is assumed that its purpose is given by technical, organizational, and business processes and that the intended behavior can be specified by process models. The behavior of the observed system is then a composition of the behaviors of the running processes.

PSA@R operates with formalized views on the control flow and security properties of a business process that can exist in any common or application-specific technical workflow notation [8], [9], [10]. A *process model*, which provides a formal representation of the controlled process, and an *event model*, an abstraction defining the internal mapping for input event streams, need to be created at the preliminary stage of PSA@R. Security requirements to be satisfied during process execution are formalized by a *security model*, which must be derived systematically [11], [12] at the initialization time.

At the analysis stage of PSA@R, the formal models are applied to monitor and predict process behavior and identify security relevant states on-the-fly. Figure 1 illustrates steps of predictive security analysis at runtime. Given the process model and the current state of the running process, a *process behavior model* representing the adjacent expected future states of the process can be computed. The process behavior model is synchronized with the running process through events received from the execution environment. Incoming events are interpreted using the event model and mapped to the process behavior model. Compliance of the predicted states with the established security policy is evaluated against the security model. To identify security relevant states on-the-fly PSA@R uses a new method described in Section IV, which enables detection of (close-future) requirements violation.

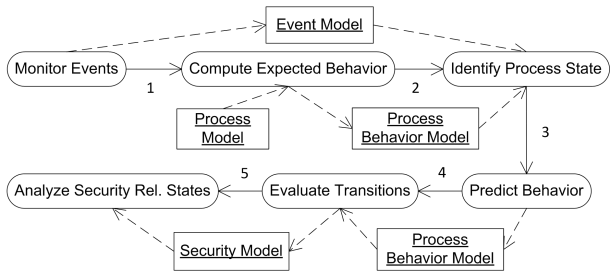


Fig. 1: Analysis stage of PSA@R

III. Process Model

This section introduces the formal process model, which is utilized to reflect the current state of the system and provides the basis for the prediction of close-future actions. PSA@R uses a process model given by an Asynchronous Product Automata (APA) representation that provides a flexible operational specification concept for cooperating systems [13]. An APA consists of a family of *elementary automata* communicating by common components of their state (shared memory).

Definition 1: An *Asynchronous Product Automaton* consists of

- a family of *state sets* $Z_s, s \in \mathbb{S}$,
- a family of *elementary automata* $(\Phi_e, \Delta_e), e \in \mathbb{E}$ and
- a *neighbourhood relation* $N : \mathbb{E} \to \mathcal{P}(\mathbb{S})$.

$\mathbb{S}$ and $\mathbb{E}$ are index sets with the names of state components and of elementary automata and $\mathcal{P}(\mathbb{S})$ is the power set of $\mathbb{S}$. For each elementary automaton (Φ_e, Δ_e) with *Alphabet* Φ_e, its *state transition relation* is

$$\Delta_e \subseteq \bigtimes_{s\in N(e)}(Z_s) \times \Phi_e \times \bigtimes_{s\in N(e)}(Z_s).$$

For each element of Φ_e the state transition relation Δ_e defines state transitions that change only the state components in $N(e)$. An APA's (global) *states* are elements of $\bigtimes_{s\in\mathbb{S}}(Z_s)$.
To avoid pathological cases it is generally assumed that $\mathbb{S} = \bigcup_{e\in\mathbb{E}}(N(e))$ and $N(e) \neq \emptyset$ for all $e \in \mathbb{E}$.
Each APA has one *initial state* $q_0 = (q_{0s})_{s\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s)$. In total, an APA $\mathbb{A}$ is defined by

$$\mathbb{A} = ((Z_s)_{s\in\mathbb{S}}, (\Phi_e, \Delta_e)_{e\in\mathbb{E}}, N, q_0).$$

Definition 2: An elementary automaton (Φ_e, Δ_e) is *activated* in a state $q = (q_s)_{s\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s)$ as to an *interpretation* $i \in \Phi_e$, if there are $(p_s)_{s\in N(e)} \in \bigtimes_{s\in N(e)}(Z_s)$ with $((q_s)_{s\in N(e)}, i, (p_s)_{s\in N(e)}) \in \Delta_e$. An activated elementary automaton (Φ_e, Δ_e) can execute a state transition and produce a *successor state* $p = (p_s)_{s\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s)$, if $q_r = p_r$ for $r \in \mathbb{S}\backslash N(e)$ and $((q_s)_{s\in N(e)}, i, (p_s)_{s\in N(e)}) \in \Delta_e$. The corresponding state transition is $(q, (e, i), p)$.

However, PSA@R does not depend on the formal method chosen for model representation. The only requirement is, that it must be possible to compute the process behavior from the process model (cf. Section III-C). For example, Petri nets [9] also meet this requirement and models produced in Petri Net Markup Language (PNML) [14] by process mining and discovery tools [15] can be used instead of APA specifications.

A. Process Behavior Model

Formally, the behavior of an operational APA model of a business process is described by a Reachability Graph (RG), also referred to as Labeled Transition System (LTS) [16].

Definition 3: The behavior of an APA is represented by all possible coherent sequences of state transitions starting with initial state q_0. The sequence

$$(q_0, (e_1, i_1), q_1)(q_1, (e_2, i_2), q_2) \dots (q_{n-1}, (e_n, i_n), q_n)$$

with $i_k \in \Phi_{e_k}$, where Φ_{e_k} is the alphabet of the elementary automaton e_k, represents one possible sequence of actions of an APA.

State transitions $(p,(e,i),q)$ may be interpreted as labelled edges of a directed graph whose nodes are the states of an APA: $(p,(e,i),q)$ is the edge leading from p to q and labelled by (e,i). The subgraph reachable from the node q_0 is called *Reachability Graph* of an APA.

Example 1: A process specification provides the control flow structure of a process as a sequence of *events* and *functions*. In an APA model that is derived from a process specification, the set of possible output events of a process function can be used as the alphabet of the elementary automaton representing the function [17]. So the interpretation i is the output event. An example for a state transition is: $(p,(transfer, event = 'critical'),q)$. The parameters of this state transition are the state p, the tuple composed of the elementary automaton *transfer* and its interpretation $event = 'critical'$, and the follow-up state q.

B. Event Model

A stream of events characterizes one specific execution trace of the observed system. This trace is a shuffle of the traces of the executed process instances. The event model determines the internal mapping for the runtime events defined by an event schema. To reduce the complexity only data required for the analysis or in generated alarms should be used in the model.

Formally, it is assumed that an event represents a letter of the alphabet that denotes the possible actions in the system. Different formal models of the same system are partially ordered with respect to different levels of abstraction.

Definition 4: Abstractions are described by so-called alphabetic language homomorphisms. These are mappings $h^* : \Sigma^* \longrightarrow \Sigma'^*$ with $h^*(xy) = h^*(x)h^*(y)$, $h^*(\varepsilon) = \varepsilon$ and $h^*(\Sigma) \subset \Sigma' \cup \{\varepsilon\}$. So they are uniquely defined by corresponding mappings $h : \Sigma \longrightarrow \Sigma' \cup \{\varepsilon\}$. In the following both the mapping h and the homomorphism h^* is denoted by h. In general, let $\check{L} \subset \check{\Sigma}^*$ and $L \subset \Sigma^*$ be prefix closed languages. $\check{L}$ is called *finer than* L and L is called *coarser than* $\check{L}$ iff an alphabetic homomorphism $\nu : \check{\Sigma}^* \to \Sigma^*$ exists with $\nu(\check{L}) = L$.

Let now P denote a finite set of process instances i of some process with $i \in P$ and let Σ_i denote pairwise disjoint copies of Σ. The elements of Σ_i are denoted by e_i and $\Sigma_P := \dot{\bigcup}_{i \in P} \Sigma_i$. The index i describes the bijection $e \leftrightarrow e_i$ for $e \in \Sigma$ and $e_i \in \Sigma_i$. Now the projection π identifies events from a specific process instance i.

Definition 5: For $i \in P$, let $\pi_i^P : \Sigma_P^* \to \Sigma^*$ with

$$\pi_i^P(e_r) = \begin{cases} e \mid & e_r \in \Sigma_i \\ \varepsilon \mid & e_r \in \Sigma_P \setminus \Sigma_i \end{cases} .$$

This is similar to the notion of a *correlation condition* [18] that defines which sets of events in the service log belong to the same instance of a process.

Remark 1: For effective use of PSA@R it is assumed that a process instance projection is possible for each event. In many applications, a process instance identification is directly available as an attribute of the event. Sometimes a set of attributes identifies the process instance. In some cases the assumption about pairwise disjoint alphabets is not true.

If the event data contain redundant or irrelevant attributes, a proper subset of attributes for use in model construction has to be selected. In order to avoid state space explosion problems, the *coarsest* abstraction that still contains all security relevant information should be used.

Example 2: Let us assume that Σ is the alphabet of events from the measured system and for a given event e the term $\#(e)$ denotes the value of an attribute involved in a transaction.

Let $h_2, h_3 : \Sigma^* \to \{'high', 'medium', 'low'\}^*$ the homomorphisms given by

$$h_2(e) = \begin{cases} 'high' \mid & 10^5 < \#(e) \\ 'low' \mid & \#(e) \leq 10^5 \end{cases}$$

$$h_3(e) = \begin{cases} 'high' \mid & 10^5 < \#(e) \\ 'medium' \mid & 10^3 < \#(e) \leq 10^5 \\ 'low' \mid & \#(e) \leq 10^3 \end{cases} .$$

Then h_3 and h_2 can be used to differentiate process control flow with respect to events with different attribute values. h_3 is finer than h_2 because $\nu : \{'high', 'medium', 'low'\}^* \to \{'high', 'low'\}^*$ exists.

C. Prediction of Close-future Process Actions

At runtime, the current state of the process behavior model of the process instance i is synchronized with the running process using the projection of the measured events to the respective state transitions $(p,(e,i),q)$ of the RG. PSA@R uses the RG to predict the close-future behavior of the process instance. As the process description is formalized in the process model, a subgraph of the RG can always be computed on-the-fly starting with the current state of the process instance. The *prediction depth* is the depth of this subgraph starting from the current state.

Example 3: The approach taken for the prediction of close-future behavior within a process is illustrated by Fig. 2. The ellipses in the event stream pane denote the observed events, whereby the filled ellipses e_0, e_1, e_2, and e_3 denote the events that belong to the specific process instance i, i.e.,

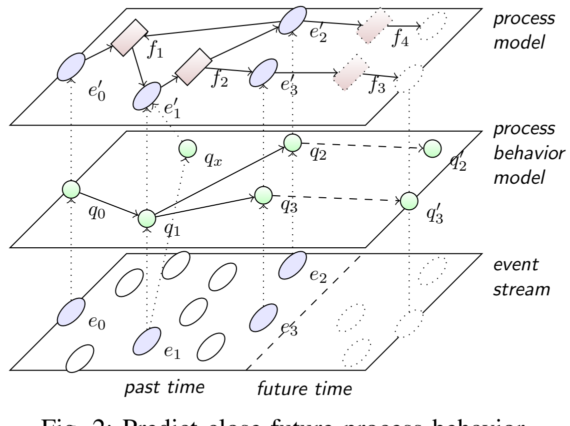


Fig. 2: Predict close-future process behavior

$e_0, e_1, e_2, e_3 \in \Sigma_i$. The ellipses in the process model pane denote abstract events with respect to the event model, e.g., $e'_1 = h(\pi_i^P(e_1))$. The dotted arrows denote this mapping. The rectangles in the process specification pane denote the process functions and the solid lines denote the transitions. If in Fig. 2 the function f_2 is modeled by the elementary automaton *transfer* and $e'_3 = h(\pi_i^P(e_3)) = 'high'$ and the depicted process instance i is in the state q_1 and the event e_3 is received, then the transition $(q_1, (transfer, event = 'high'), q_3)$ will match the current situation. The process specification contains possible close-future functions f_3 and f_4 and associated events to be predicted. The dashed arrows in Fig. 2 denote the predicted close-future process behavior.

IV. On-the-fly Identification of Critical States

In addition to the predicted process behavior, the security model is needed to identify security relevant states of the current state of the business process. As a notation for the security model we use monitor automata.

Definition 6: A monitor automaton $\mathbb{M}$ consists of a set $\mathscr{M}$ of labeled states, an alphabet Λ of predicates on RG state transitions, a transition relation $\mathscr{T}_{\mathscr{M}} \subseteq \mathscr{M} \times \Lambda \times \mathscr{M}$, a set of initial states $\emptyset \neq \mathscr{M}_0 \subset \mathscr{M}$, and a set of accepting states $\mathscr{M}_f \subset \mathscr{M}$.

Predicates of $\mathbb{M}$ are applied to state transitions $(p_i, (e_j, i_k), q_l)$ of the RG.

Example 4: The predicate $(, (, event = 'high'),)$ is *true* if $'high'$ is bound to the interpretation variable *event* of the interpretation i_k. No condition for the predecessor state p_i, successor states q_l and the elementary automaton e_j is given.

With a monitor automaton it is possible to express security requirements with respect to current and close-future behavior of a process represented by a RG. In this case accepting states refer to *security critical states*. Each state of the RG has an associated state set of $\mathbb{M}$, which is computed during simulation. Security critical states are reached whenever an accepting state of $\mathbb{M}$ becomes a member of such state set.

State sets of $\mathbb{M}$ are successively assigned to RG states during simulation as follows: $\mathscr{M}_0$ is assigned to the initial state q_0 of the given RG. Each predicate $\lambda \in \Lambda$ of $\mathbb{M}$ is of the form $\lambda(x)$, where x is a state transition $(p, (e, i), q)$ of the RG. Each $\lambda \in \Lambda$ is associated with one of the transitions $\mathscr{T}_{\mathscr{M}}$ of $\mathbb{M}$. During the run of the simulation, for each transition $(q_i, (e_j, i_k), q_x)$ of the RG, the monitor automaton state set for the RG state q_x is computed as follows:
Let $\mathscr{A}_i \subseteq \mathscr{M}$ be the state set of $\mathbb{M}$ assigned to the current RG state q_i. The set $\mathscr{T}_{A_i}$ of transitions to be checked is now given by:

$$\mathscr{T}_{A_i} = \{(m_m, \lambda_o, m_n) \in \mathscr{T}_M \mid m_m \in \mathscr{A}_i\}.$$

All predicates λ_o have to be checked for the current transitions $(q_i, (e_j, i_k), q_x)$ of the RG. Based on the monitored transitions $\mathscr{M}\mathscr{T}_x$ new states $\mathscr{B}_x$ of $\mathbb{M}$ and the changed states $\mathscr{C}_x$ of $\mathbb{M}$ are computed as follows:

$$\mathscr{M}\mathscr{T}_x := \{(m_m, m_n) \in \mathscr{M} \times \mathscr{M} \mid (m_m, \lambda_o, m_n) \in \mathscr{T}_{A_i} \wedge \lambda_o((p_i, (e_j, i_k), q_x))\}$$
$$\mathscr{B}_x := \{m_n \in \mathscr{M}\mathscr{T}_x | (m_m, m_n) \in \mathscr{M}\mathscr{T}_x\}$$
$$\mathscr{C}_x := \{m_m \in \mathscr{M}\mathscr{T}_x | (m_m, m_n) \in \mathscr{M}\mathscr{T}_x\}$$

The computation of $\mathscr{B}_x$ and $\mathscr{C}_x$ will be implicitly assumed when used in Algorithm 1 and 2. The set $\mathscr{R}\mathscr{S}_n$ includes possible states in the RG which represents the current state of the real system. After the occurrence of a certain event and extension of the RG if necessary, the new set $\mathscr{R}\mathscr{S}_{n+1}$ and the corresponding monitor automaton state set $\mathscr{A}_i^r$ has to be computed for every state $q_i \in \mathscr{R}\mathscr{S}_{n+1}$ by Algorithm 1.

Algorithm 1 (Security compliance check):

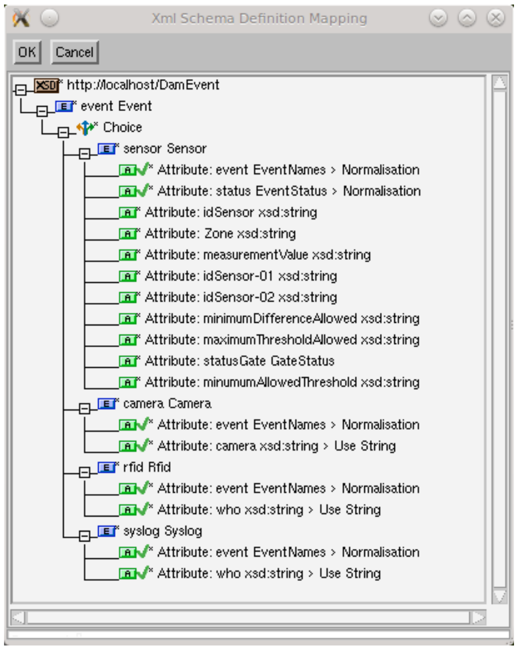
```
$SP := \emptyset$
**for** $(p_i, (e_j, i_k), q_l) \in \{p_i | p_i \in \mathscr{R}\mathscr{S}_n\} \wedge \lambda_e((p_i, (e_j, i_k), q_l))$ **do**
    $SP := SP \cup \{p_i\}$
    **if** $\mathscr{B}_l = \emptyset$ **then**
        **if** $q_l \in SP$ **then**
            $\mathscr{A}_l^r := \mathscr{A}_l^r \cup \mathscr{A}_i^r$
        **else**
            $\mathscr{A}_l := \mathscr{A}_i$
    **else**
        **if** $q_l \in SP$ **then**
            $\mathscr{A}_l^r := \mathscr{A}_i^r \cup \mathscr{B}_l$
        **else**
            $\mathscr{A}_l^r := \mathscr{B}_l \cup (\mathscr{A}_i^r \setminus \mathscr{C}_l)$
$\mathscr{R}\mathscr{S}_{n+1} := SP$
```

For the RG state set $\mathscr{R}\mathscr{S}_{n+1}$ new state sets $\mathscr{A}_i^p$ of $\mathbb{M}$ have to be computed for every predicted RG state q_i. The function *visit* sets a mark to a certain state which can be checked by the predicate *visited*. The predicate *closer* indicates that the current path to elements of the state set $\mathscr{R}\mathscr{S}_{n+1}$ to the node given as a parameter is shorter than the paths to this node processed before. The monitor automaton state sets of the predicted states which can be reached from states of the set $\mathscr{R}\mathscr{S}$ are computed by Algorithm 2.

Algorithm 2 (Predict security violations):

```
$S := \emptyset$
**for** $q_l \in \mathscr{R}\mathscr{S}_{n+1}$ **do**
    $S := S \cup \{q_l)\}$
    **while** $S \neq \emptyset$ **do**
        $S := S \setminus \{q_{i_2}\}$
        **for** $(q_{i_2}, (e_{j_2}, i_{k_2}), q_{l_2}) \in$ RG **do**
            $A := \begin{cases} A_{i2}^r \mid & \neg visited(q_{i2}) \wedge \mathscr{A}_{i_2}^p = \emptyset \\ \mathscr{A}_{i_2}^p \mid & \textbf{\textit{else}} \end{cases}$
            $visit(q_{i_2})$
            **if** $\mathscr{B}_{l_2} = \emptyset$ **then**
                **if** visited$(q_{l_2})$ **then**
                    $\mathscr{A}_{l_2}^p := \mathscr{A}_{l_2}^p \cup \mathscr{A}_{i_2}^p$
                    **if** $closer(q_{l_2}, \mathscr{R}\mathscr{S}_{n+1})$ **then**
                        $S := S \cup \{q_{l_2})\}$
                **else**
                    $\mathscr{A}_{l_2}^p := \mathscr{A}_{i_2}^p$
                    $S := S \cup \{q_{l_2}\}$
            **else**
                **if** visited$(q_{l_2})$ **then**
                    $\mathscr{A}_{l_2}^p := \mathscr{A}_{i_2}^p \cup \mathscr{B}_{l_2}$
                **else**
                    $\mathscr{A}_{l_2}^p := \mathscr{B}_{l_2} \cup (\mathscr{A}_i^p \setminus \mathscr{C}_{l_2})$
                $S := S \cup \{q_{l_2})\}$
```

In this algorithm we do not analyze the consequences of reaching security critical states. Trigger actions such as generating alerts, which are executed when accepting monitor



```
Xml Schema Definition Mapping
OK  Cancel
XSD* http://localhost/DamEvent
  E* event Event
    * Choice
      E* sensor Sensor
        A ✓* Attribute: event EventNames > Normalisation
        A ✓* Attribute: status EventStatus > Normalisation
        A* Attribute: idSensor xsd:string
        A* Attribute: Zone xsd:string
        A* Attribute: measurementValue xsd:string
        A* Attribute: idSensor-01 xsd:string
        A* Attribute: idSensor-02 xsd:string
        A* Attribute: minimumDifferenceAllowed xsd:string
        A* Attribute: maximumThresholdAllowed xsd:string
        A* Attribute: statusGate GateStatus
        A* Attribute: minumumAllowedThreshold xsd:string
      E* camera Camera
        A ✓* Attribute: event EventNames > Normalisation
        A ✓* Attribute: camera xsd:string > Use String
      E* rfid Rfid
        A ✓* Attribute: event EventNames > Normalisation
        A ✓* Attribute: who xsd:string > Use String
      E* syslog Syslog
        A ✓* Attribute: event EventNames > Normalisation
        A ✓* Attribute: who xsd:string > Use String
```

Fig. 3: Attribute selection and mapping

automaton states become members of a state set, can be defined. Different security properties might be monitored simultaneously by allowing more than one transition of the monitor automaton to be triggered at the same time.

V. Hydroelectric Power Plant Security

In order to demonstrate what kind of security requirements we consider and how our model-based runtime analysis is applied, we use a combined technical and organizational process from a hydroelectric power plant in a dam [7] and explain the evaluation of security requirements for this process at runtime. Since dams are complex infrastructures, a huge number of parameters must be monitored to guarantee safety and security. Which parameters are actually monitored, depends on the dam's structure, design, purpose and function [19].

Figure 3 shows a mapping (an event model) with regard to the events from dam sensors, cameras, RFID scanners, and syslog.

Here, we examine a misuse case related to the insider threat that is still prevalent and posing a serious risk to critical infrastructures [20]. We assume that the respective security goal is given as: *All safety critical actions in the control room are carried out by a dam operator with administrative rights.* Other types of security requirements, which could be supervised by PSA@R, are typical authenticity and integrity requirements like the following example: *Whenever a certain control decision is made, the input information that presumably led to it must be authentic [19].* Specifically, *authenticity* can be seen as the assurance that a particular action has occurred in the past.

A. Misuse Case Scenario

The storage dam of the hydroelectric power plant is remotely controlled by a Supervisory Control And Data Acquisition

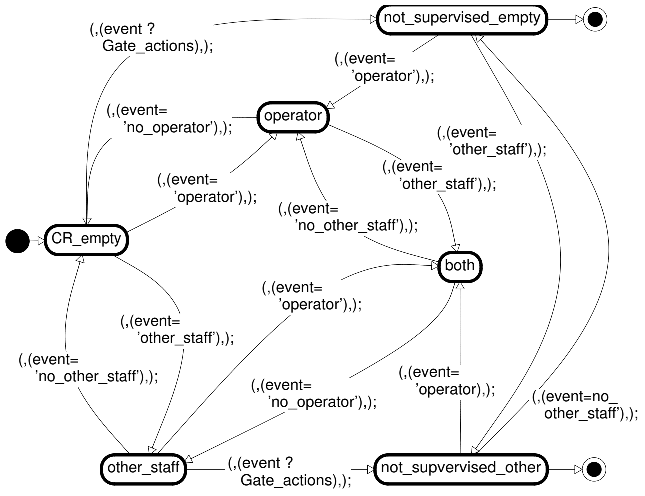

not_supervised_empty
(,(event ? Gate_actions),);
(,(event= 'operator'),);
(,(event= 'no_operator'),);
operator
(,(event= 'other_staff'),);
(,(event= 'other_staff'),);
(,(event= 'operator'),);
(,(event= 'no_other_staff'),);
CR_empty
both
(,(event= 'operator'),);
(,(event= 'other_staff'),);
(,(event= 'no_other_staff'),);
(,(event= 'no_operator'),);
(,(event= 'operator),);
(,(event=no_ other_staff'),);
other_staff
(,(event ? Gate_actions),);
not_supvervised_other

Fig. 4: Monitor automaton for hydroelectric power plant

(SCADA) system from the control station located in the control room. Physical (Radio Frequency Identification (RFID) based) and logical access controls are deployed. A disgruntled employee of the dam with a non-administrative role (i.e., cleaning staff) but who is enabled to access the control room uses stolen administrator credentials to open dam gates.

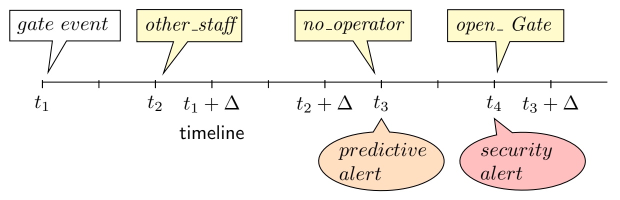
There are several attack steps. First, the disgruntled employee uses his RFID badge to enter the control room while an administrator is inside. The disgruntled employee waits until the administrator leaves the control room and uses the stolen administrator credentials to log in into the control system. Then he issues the open gate command from the control station. The water gates open discharging the dam's reservoir. The decrease of the water level endangers the people using the dam's reservoir for recreational activities.

Note, that this attack can be discovered if the system is able to detect that the administrator command was issued while no employee with the administrator role had accessed the control room with her badge.

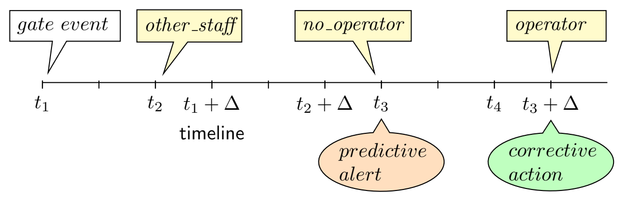
B. Specification of a Monitor Automaton

The security requirements that certain actions of the dam workflows have to be supervised will be controlled by a monitor automaton $\mathbb{M}$ as introduced in the previous section. Figure 4 shows a specification of $\mathbb{M}$.

The initial state of $\mathbb{M}$ *CR_empty* (control room is empty) is marked by the filled circle. The critical states *not_supervised_empty* and *not_supervised_other* are marked by the circle with the small filled circle inside. These states reflect the situation that an action from the set *Gate_actions* has been executed while the control room is either empty or only manned with non-administrative staff. If one of these states is reached during prediction an alarm will be generated. The predicates attached to the arcs of $\mathbb{M}$ define predicates for transitions of the RG. This automaton is scheduled according to the algorithm presented in the previous section during the computation of the RG in the prediction process. The



(a) Alert situation



(b) Predicted situation not occurred

Fig. 5: Security reasoning

λ predicates in Section IV correspond to the predicates of the arcs in this automaton.

Predicates of the monitor automaton are applied to state transitions of the RG $(p_i,(e_j,i_k),q_l)$, for example, the predicate $(,(,event='no\_other\_staff'),)$ is true if $'no\_other\_staff'$ is bound to the interpretation variable *event* of the interpretation i_k. No condition for the predecessor and successor state p_i, q_l and the elementary automaton e_j is given in this example. The event $'no\_other\_staff'$ (all non-administrative staff left the control room) referenced in the above predicate is a higher level event generated by preprocessing the low-level events from the RFID scanners and events from cameras which capture the motion of staff at the entrance of the control room.

C. Evaluation of State Transitions

In order to exemplify the security analysis at runtime, let us assume that the system is in a state where an operator is present in the control room, there is only one monitor automaton as shown in Fig. 4 defined, and the current state of the monitor automaton is *operator*. We now describe the reasoning process at runtime. Figure 5a shows a possible timeline of events. A *security warning* indicates a situation where a security requirement is broken but has no negative impact at creation time. A *predictive alert* is raised when a broken security requirement might lead to a security critical situation in the close future. A *security alert* is raised if a security critical situation has been detected. These warnings and alerts are mapped to corresponding events and fed into the runtime environment.

If at time t_1 an event from a gate function is received, then the state component of the process model representing the status of the gate will be changed but the state of the monitor automaton will not change. The reachability analysis does not "see" an upcoming security violation within the scope Δ, so no alarm has to be triggered.

If at time $t_2 > t_1$ the event $'other\_staff'$ produced by the RFID scanners of the control room is received, then the state component of the process model representing the manning of

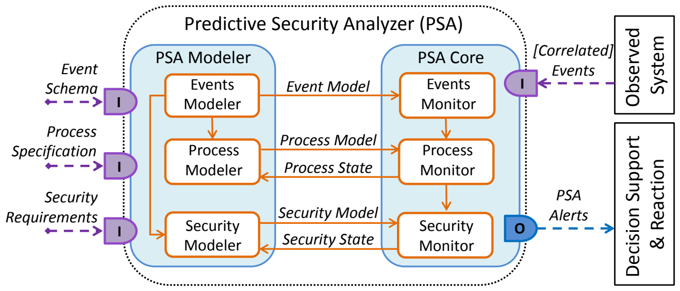


Fig. 6: Architecture of the PSA

the control room will be changed and the monitor automaton changes the state to *both*. No security violations is "seen" within the scope Δ.

If at time $t_3 > t_2$ the event $'no\_operator'$ is received which indicates that the last operator has left the control room, then the state component of the process model representing the manning of the control room will be changed and the monitor automaton also changes the state to *other_staff*. Now in one possible process execution sequence, an event from a gate function such as *open_Gate* is reachable within Δ. In this situation the reachability analysis shows that this function would violate an associated security requirement. Therefore, a predictive alert is raised because a broken security requirement might lead to a security critical situation in the close future.

If at time $t_4 > t_3$ an event from a gate function such as *open_Gate* is received, then the monitor automaton changes to the critical state *not_supervised_other*. As a security critical situation has now been detected, a security alert is raised.

Now let us assume that at time $t_3+\Delta$ an event is received which indicates that an operator is back in the control room and the critical state was not reached as predicted. In this case, we know that the issued predictive alert did not lead to a security alert (cf. Fig. 5b). Therefore, a corrective action such as the reduction of the security warning level or lifting of restrictions on the business process may be necessary.

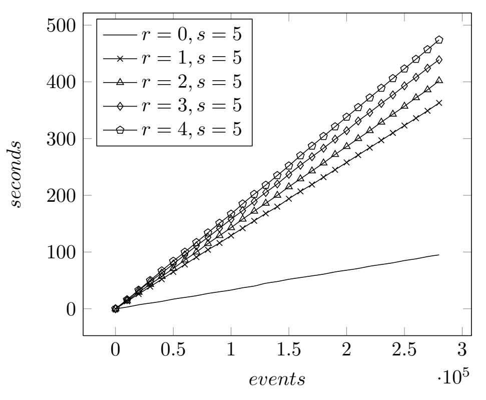
VI. Evaluation of Security Analysis at Runtime

To evaluate the performance of different modeling strategies in the scope of PSA@R, we have implemented a prototype, the PSA, that supports the complete life-cycle of security analysis at runtime from formal process specification to exhaustive validation, including visualization and inspection of computed RGs and monitor automata. Our implementation is based on Common LISP [21] and technical specifications from [13].

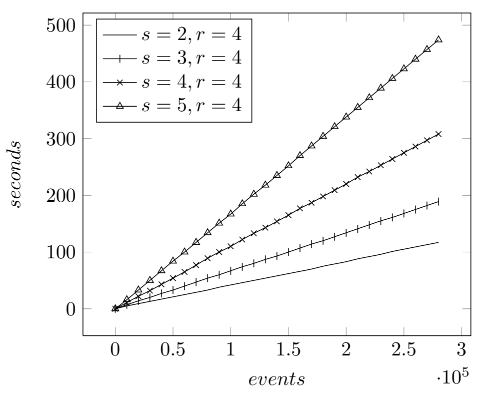
A. Prototype Architecture

Figure 6 shows the architecture of the PSA consisting of two main parts: the *PSA Modeler* that provides functionality for process formalization and the *PSA Core* that performs process security analysis. "I" and "O" are input and output interfaces.

During initialization an operator uses the PSA Modeler components to formalize input required for process simulation. The *Event Modeler* supports the derivation of an event model from given event schemata (cf. Fig. 3) and stores the respective



(a) Execution time depending on number r of security requirements



(b) Execution time depending on number s of successor states

Fig. 7: Execution time measurements

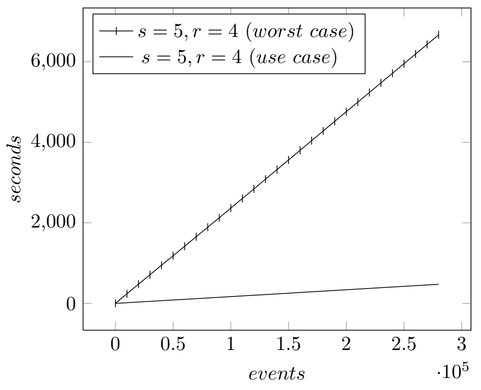
mapping for interpretation of runtime events. The *Process Modeler* allows to formalize process specifications using methods introduced in Section III. The *Security Modeler* provides means for graphical specification of monitor automata representing security models (cf. Section IV).

To launch the security analysis the PSA models with initial configurations such as the initial state of a process model and an active security model need to be loaded into the PSA Core in the form of compiled code. During the monitoring and analysis stage the PSA Core components receive runtime events from the observed system, for example, events from dam's SCADA system and RFID scanners. The *Event Monitor* interprets these events in accordance with the defined event model. The normalized events are fed to the *Process Monitor* that performs behavior prediction based on the process model. If a legitimate event does not comply with the model, the PSA supports an adjustment of the model on-the-fly within the process modeler utilizing backward references from the compiled process model. To detect security violations the information about predicted state transitions is forwarded to the *Security Monitor*. By executing the security model the Security Monitor identifies process states critical from the security perspective and issues alerts that are forwarded to decision support and reaction for further processing. Backward references within the compiled security model allow to show the current state within the security modeler.

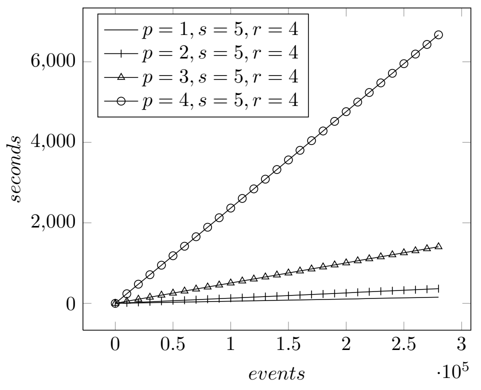
B. Evaluation

In the project MASSIF [22] PSA@R is currently applied to check security requirements in four industrial domains: (i) the management of the Olympic Games IT infrastructure [23]; (ii) a mobile phone based Mobile Money Transfer System (MMTS) [24], facing high-level threats such as money laundering; (iii) managed IT outsource services for large distributed enterprises and (iv) an IT system supporting a critical infrastructure (dam) [7]. We used the hydroelectric power plant scenario (iv) to demonstrate the capability of the PSA prototype to process and correlate events from heterogeneous sources (cf. Section V). To evaluate the PSA prototype with respect to performance issues, however, we used event logs from scenario (ii) as a resource intensive application which requires high throughput. In this case, events referred to transactions conducted in an MMTS and processes represented user behaviors observed from transaction logs [25]. To achieve high load a recorded event stream was sent directly to the PSA socket interface. Measurements were produced on a personal computer with Intel Core 6700 CPU and 4GB memory.

The measurements presented evaluate the execution time and the number of events received by the PSA. We have examined four aspects important from the application perspective: (i) effects of the number of security requirements to the execution time; (ii) effects of the abstraction level to analysis; (iii) effects of cycle reduction in a RG; (iv) effects of changing prediction depths. The prediction depth $p = 4$ was used in (i)–(iii), but did not effect the performance of the simulation because the complete RG could be computed in advance. Figure 7a shows that the execution time depends linearly on the number of received events and the gradient of this linear slope is determined by the number of security requirements. In order to investigate effects of the abstraction level we have evaluated finer and coarser process models. A coarser model results in less successor states and thus reduces the effort for the monitoring algorithm. The abstraction level can be adjusted, for instance, as shown in Example 2. In [8] Mendling presented an extensive metrics analysis on four collections of 2003 Event-driven Process Chain (EPC) process specifications. In this study, the number of nodes a connector is in average connected to resulted in 3.56 for the mean value μ and 2.40 for the standard deviation σ. Therefore, for our performance measurements we used a number $s \in \{2, 3, 4, 5\}$ of successor states. Figure 7b shows that the execution time depends linearly on the number of received events. The moderate increase of gradients of the corresponding linear slopes was caused by the optimization of the monitoring algorithm related to cycles in the RG. These experiments show that one month of data logged in an MMTS (285.619 events from 50.265 processes) is analyzed by the PSA within two to eight minutes depending on the model abstraction. In order to simulate the possible worst case for five successor



(a) Comparison of MMTS use case with worst case



(b) Execution time depending on prediction depth p

Fig. 8: Worst case behavior and influence of prediction depth

states (four requirements) we produced a synthetic model. Figure 8a displays the comparison between the MMTS model and the worst case model. The worst case performance can be improved by reasonably limiting the number of predicted steps. Figure 8b illustrates the effects of reduced prediction depth in the worst case model. During the experiment the security requirements were successfully checked in all combinations.

VII. Related Work

The work presented here combines specific aspects of security analysis with generic aspects of process monitoring, simulation, and analysis. The background of these aspects is given by the utilization of models at runtime [26]. The proposed approach is similar to the approaches described in [17], [27] in terms of event-driven process analysis. However, in our work we focus on an integrated algorithm for computation of reachability graphs with evaluation of security properties given by monitor automata. Recently, runtime monitoring of concurrent distributed systems based on Linear Temporal Logic (LTL), state-charts, and related formalisms has also received attention [28]. However, these works are mainly focused on error detection, e.g., concurrency related bugs.

Approaches focusing on security models at runtime are given in [29], [30]. The first work proposes a novel methodology to synchronize an architectural model reflecting access control policies with the running system. Therefore, the methodology emphasizes policy enforcement rather than security analysis. The integration of runtime and development-time information on the basis of an ontology to engineer industrial automation systems is discussed in [30]. Schneider [31] analyzed a class of safety properties and related enforcement mechanisms that work by monitoring execution steps of some target system, and terminating the target's execution, whenever it is about to execute an operation, which would violate the security policy. Extensions of this approach are discussed in [32]. However, security automata as defined in [31] are related to a specific trace of execution, whereas in the monitor automata concept proposed here, the whole RG is used as a reference to the possible system's behavior. Patterns and methods to allow for monitoring security properties are developed in [33], [34], [35].

Diverse categories of tools applicable for modeling and simulation of business processes are based on different semi-formal or formal methods such as EPCs [8] or Petri nets [9]. Likewise, some general-purpose simulation tools such as CPNTools [36] were proven to be suitable for simulating business processes. The process mining framework ProM [15] supports plug-ins for different types of models and process mining techniques. However, independently from the tools and methods used, such simulation tools concentrate on statistical aspects, redesign, and commercial optimization of the business process. On the contrary, we propose an approach for *on-the-fly* dynamic simulation and analysis on the basis of operational formal models. This includes consideration of the current process state and the event information combined with the corresponding steps in the process model. We consider the framework presented in [37] on runtime compliance verification for business processes as complementary to our work.

VIII. Conclusions and Further Work

In this paper, we presented an integrated approach called PSA@R to analyze the security status of a process and to identify possible violations of the security policy in close future. The approach also provides early awareness about deviations of a running process from expected behavior as specified by the model. When such anomalies refer to process misbehavior or disruption, alarms will be raised for decision support and reaction. Moreover, we described how to extend process behavior computation with algorithms for on-the-fly security compliance checks and prediction of close-future security violations. Thus, our integrated security analysis approach identifies current and close-future violations of the security policy. As security relies on the compliance of actual behavior with the given specifications this early detection of changes and reaction elevates security of the process in question. In combination with other novel applications PSA@R enables anticipatory impact analysis, decision support and impact mitigation by adaptive configuration of countermeasures. Moreover, we assume that our results can also be applied to on-the-fly analysis of compliance and dependability requirements. In further work, we consider to integrate methods, such as the one described in [38] using metrics to quantify deviations from process specifications.

Acknowledgment

The presented work was developed in context of the project MASSIF (ID 257475) being co-funded by the European Commission within the Seventh Framework Programme and the project ACCEPT (ID 01BY1206D) being funded by the German Federal Ministry of Education and Research.

References

[1] P. Wood, "Internet Security Threat Report, 2011 Trends, Vol. 17," Symantec Corporation, Technical Report, April 2012.

[2] M. Klemen, S. Biffl, and T. Neubauer, "Secure business process management: A roadmap," in *Proceedings of the First International Conference on Availability, Reliability and Security (ARES)*. IEEE, 1 2006, pp. 457–464.

[3] P. H. C. Wolf, "The state of business process management," http://www.bptrends.com/, BPTrends Report, 2012.

[4] P. Tallon, "Inside the adaptive enterprise: an information technology capabilities perspective on business process agility," *Information Technology and Management*, vol. 9, no. 1, pp. 21–36, 2008.

[5] S. Rinderle-Ma, M. Reichert, and B. Weber, "Relaxed compliance notions in adaptive process management systems," in *Proceedings 27th Int'l Conference on Conceptual Modeling (ER'08)*, ser. LNCS, no. 5231. Springer, October 2008, pp. 232–247.

[6] D. Schumm, F. Leymann, Z. Ma, T. Scheibler, and S. Strauch, "Integrating compliance into business processes: Process fragments as reusable compliance controls," in *Proceedings of the Multikonferenz Wirtschaftsinformatik, (MKWI 2010)*. Universitätsverlag Göttingen, 2010.

[7] L. Romano, S. D. Antonio, V. Formicola, and L. Coppolino, "Enhancing SIEM technology to protect critical infrastructures," in *CRITIS 2012, the seventh CRITIS Conference on Critical Information Infrastructures Security*, September 2012.

[8] J. Mendling, *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*, ser. LNBIP. Springer, 2008, vol. 6.

[9] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in BPMN," *Information and Software Technology*, vol. 50, no. 12, pp. 1281–1294, 2008.

[10] W. Arsac, L. Compagna, G. Pellegrino, and S. Ponta, "Security Validation of Business Processes via Model-Checking," in *Engineering Secure Software and Systems (ESSoS 2011)*, ser. LNCS. Springer, 2011, vol. 6542, pp. 29–42.

[11] B. Fabian, S. Gürses, M. Heisel, T. Santen, and H. Schmidt, "A comparison of security requirements engineering methods," *Requirements engineering*, vol. 15, no. 1, pp. 7–40, 2010.

[12] D. Mellado, C. Blanco, L. E. Sánchez, and E. Fernández-Medina, "A systematic review of security requirements engineering," *Comput. Stand. Interfaces*, vol. 32, no. 4, 2010.

[13] P. Ochsenschläger, J. Repp, R. Rieke, and U. Nitsche, "The SH-Verification Tool Abstraction-Based Verification of Co-operating Systems," *Formal Aspects of Computing*, vol. 10, no. 4, pp. 381–404, 1998.

[14] M. Weber and E. Kindler, "The petri net markup language," in *Petri Net Technology for Communication-Based Systems*, ser. LNCS. Springer, 2003, vol. 2472, pp. 124–144.

[15] W. M. P. van der Aalst, B. F. van Dongen, C. Günther, A. Rozinat, H. M. W. Verbeek, and A. J. M. M. Weijters, "ProM: The Process Mining Toolkit," in *BPM 2009 Demonstration Track*, vol. 489. CEUR, 2009, pp. 1–4.

[16] D. A. Peled, *Software Reliability Methods*, 1st ed. Springer, 2001.

[17] J. Eichler and R. Rieke, "Model-based Situational Security Analysis," in *Workshop on Models@run.time*. CEUR, 2011, vol. 794, pp. 25–36.

[18] H. R. Motahari-Nezhad, R. Saint-Paul, F. Casati, and B. Benatallah, "Event correlation for process discovery from web service interaction logs," *The VLDB Journal*, vol. 20, no. 3, pp. 417–444, Jun. 2011.

[19] L. Coppolino, M. Jäger, N. Kuntze, and R. Rieke, "A Trusted Information Agent for Security Information and Event Management," in *ICONS 2012, The Seventh International Conference on Systems*. IARIA, 2012, pp. 6–12.

[20] M. E. Luallen, "Managing Insiders in Utility Control Environments," SANS, A SANS Whitepaper in Association with SANS SCADA Summits, Q1, 2011, March 2011.

[21] G. L. Steele, Jr., *Common LISP: the language (2nd ed.)*. Newton, MA, USA: Digital Press, 1990.

[22] R. Rieke, E. Prieto, R. Diaz, H. Debar, and A. Hutchison, "Challenges for advanced security monitoring – the MASSIF project," in *Trust, Privacy and Security in Digital Business*, ser. LNCS, S. Fischer-Hübner, S. Katsikas, and G. Quirchmayr, Eds. Springer, 2012, vol. 7449, pp. 222–223.

[23] E. Prieto, R. Diaz, L. Romano, R. Rieke, and M. Achemlal, "MASSIF: A promising solution to enhance olympic games it security," in *Global Security, Safety and Sustainability & e-Democracy*, ser. LNICST, C. K. Georgiadis *et al.*, Eds. Springer, 2012, vol. 99.

[24] C. Gaber, B. Hemery, M. Achemlal, M. Pasquet, and P. Urien, "Synthetic logs generator for fraud detection in mobile transfer services," in *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems (CTS2013)*, 2013.

[25] R. Rieke, M. Zhdanova, J. Repp, R. Giot, and C. Gaber, "Fraud detection in mobile payment utilizing process behavior analysis," in *Proceedings of 2013 International Conference on Availability, Reliability and Security, ARES 2013*. IEEE Computer Society, 2013, pp. 662–669.

[26] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Future of Software Engineering*. IEEE, 2007, pp. 37–54.

[27] R. Rieke and Z. Stoynova, "Predictive security analysis for event-driven processes," in *Computer Network Security*, ser. LNCS. Springer, 2010, vol. 6258, pp. 321–328.

[28] T. Massart and C. Meuter, "Efficient online monitoring of LTL properties for asynchronous distributed systems," Université Libre de Bruxelles, Tech. Rep., 2006.

[29] B. Morin, T. Mouelhi, F. Fleurey, Y. Le Traon, O. Barais, and J.-M. Jézéquel, "Security-driven model-based dynamic adaptation," in *Automated Software Engineering (ASE 2010)*. ACM, 2010, pp. 205–214.

[30] M. Melik-Merkumians, T. Moser, A. Schatten, A. Zoitl, and S. Biffl, "Knowledge-based runtime failure detection for industrial automation systems," in *Workshop Models@run.time*. CEUR, 2010, pp. 108–119.

[31] F. B. Schneider, "Enforceable security policies," *ACM Transactions on Information and System Security*, vol. 3, no. 1, pp. 30–50, 2000.

[32] F. Martinelli, I. Matteucci, and C. Morisset, "From qualitative to quantitative enforcement of security policy," in *MMM-ACNS*, ser. LNCS, I. V. Kotenko and V. A. Skormin, Eds., vol. 7531. Springer, 2012, pp. 22–35.

[33] C. Serban and B. McMillin, "Run-time security evaluation (RTSE) for distributed applications," in *Symposion on Security and Privacy*. IEEE, 1996, pp. 222–232.

[34] T. Tsigritis and G. Spanoudakis, "Diagnosing runtime violations of security & dependability properties," in *Software Engineering and Knowledge Engineering (SEKE 2008)*. KSI, 2008, pp. 661–666.

[35] A. Evesti, E. Ovaska, and R. Savola, "From security modelling to run-time security monitoring," in *European Workshop on Security in Model Driven Architecture (SECMDA 2009)*. CTIT, 2009, pp. 33–41.

[36] A. Rozinat, M. T. Wynn, W. M. P. van der Aalst, A. H. M. ter Hofstede, and C. J. Fidge, "Workflow simulation for operational decision support," *Data & Knowledge Engineering*, vol. 68, no. 9, pp. 834–850, 2009.

[37] F. M. Maggi, M. Montali, M. Westergaard, and W. M. P. van der Aalst, "Monitoring business constraints with linear temporal logic: An approach based on colored automata," in *Business Process Management (BPM 2011)*, ser. LNCS, vol. 6896. Springer, 2011, pp. 132–147.

[38] S. Banescu and N. Zannone, "Measuring privacy compliance with process specifications," in *Workshop on Security Measurements and Metrics (MetriSec 2011)*. IEEE, 2011.