SecSy: Synthesizing Process Event Logs

Thomas Stocker and Rafael Accorsi
University of Freiburg, Germany
{ stocker ¦ accorsi }@iig.uni-freiburg.de

Abstract: One difficulty at developing mechanisms for business process security monitoring and auditing is the lack of representative, controllably generated test runs to serve as an evaluation basis. This paper presents an approach and the corresponding tool support for event log synthesis. The novelty is that it considers the activity of an "attacker" able to purposefully infringe security and compliance requirements or simply manipulate the process' control and data flow, thereby creating deviations of the intended process model. The resulting logs can be readily replayed on a reference monitor, or serve as input for auditing tools based upon, e.g., process mining.

1 Introduction

Research into business process security and compliance is concerned with the requirements formalization of security (e.g. secrecy, binding of duties) and compliance requirements (e.g. obligations, interdependencies between activities) [BAS09] and the development of well-founded techniques and tools for analyzing, monitoring and auditing these requirements in business process specifications [Acc13]. Approaches for this may include, for instance, program analysis [AL12, AW10b, AW11], usage control monitoring [ASK08] and various types of a-posteriori process analysis [AS08, AW10a, vdA11].

Here, a particular challenge arises when it comes to testing the effectiveness of monitoring and auditing techniques and corresponding tools [AS12, AS13]. Specifically, to test these tools one needs controllably generated event logs that contain process executions comprising process flexibility and variability on the one hand, and process non-compliance on the other, thereby mimicking structural vulnerabilities, process dynamics, intentional attacks and user errors [LA11]. Such event logs can serve as input for monitoring, auditing and mining tools, thereby allowing developers to assess their *kill-rate*, i.e. the precision to identify the violation of the designated security and compliance properties or deviations from an original process model.

Business process simulation techniques can be used to generate these runs and, thus, to synthesize event logs. However, the current state of the art (e.g. [CKH98, Tum96, vdA10a]) and tool support (e.g. [Ala, BSB98, BS10, CPN], see [JVN06] for survey) do not take into account the controlled generation of event logs considering flexibility and non-compliance for a specific process.

In computer security, the "attacker model" describes activities that deviate from the prescribed behavior of programs, systems and protocols [Cer01]. This abstraction serves

not to solely model an “attacker”, but rather unexpected fault situations that interfere with the usual operation of a program or protocol. In security, if a program (provably) withstands a certain “attacker model”, then it is considered “secure” against such an attacker. This paper employs the “attacker model” abstraction as a building block of event log generation. Specifically, the paper presents a controllable simulation approach and corresponding tool support that allows users to synthesize event logs of business processes comprising non-compliance and business process flexibility wrt. their control flow.

The contributions of this paper are threefold: (1) the definition of model transformation operators to derive and encode variants of process models, e.g. skipping and swapping activities; (2) the definition of trace transformations which transform valid process traces by enforcing or violating security-related properties; (3) considering traditional security and organizational properties for business processes [KR01], the approach allows for a controllable generation of event logs in which these requirements are enforced or violated; and (4) the presentation and evaluation of SecSy , a tool that allows the configurable generation of event logs in several output formats. Due to space constraints, this paper will present only the main intuition behind the approach and key features of SecSy. (SecSy can be downloaded at http://bit.ly/SecSy). A follow-up paper will present the whole set of operators and a thorough description and evaluation of SecSy.

To the best of our knowledge, the insights behind the approach we present in this paper are novel. SecSy has been already used in the generation of several case studies and is capable of producing log files of industrial complexity that can be readily used for testing, e.g., process mining and monitoring approaches. Overall, by allowing users to configure the executions according to transformations on model and trace level, it is possible to capture the evolution (flexibility) of business process, as well as intentionally building non-compliance into the log. However, the approach does not guarantee that all possible ways of violating specific security properties (i.e. changing the shape of a trace accordingly) are reflected by the synthesized log. Ongoing work investigates how to apply testing methods – especially mutation testing [JH11] – as a foundation for synthesis.

Paper *structure.* Section 2 introduces the overall approach, whereas the main building blocks are presented in detail in Section 3. Section 4 presents a subset of operators for model transformation and property enforcement/violation, thereby defining the aforementioned “attacker model”. The tool SecSy is described and high-level evaluated in Section 5.

2 Simulation Approach

The overall approach is depicted in Fig. 1. It takes a series of business process specifications as input and generates a process log that contains traces of these specifications. Based upon the security and compliance requirements, deviations from

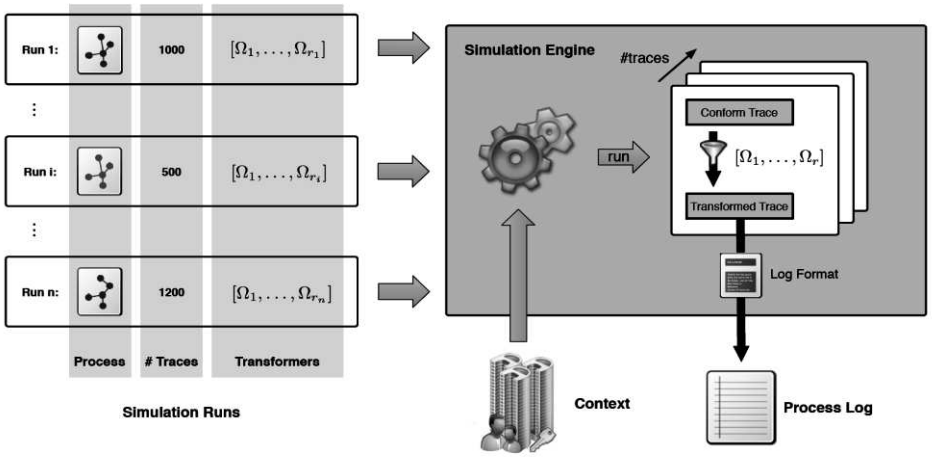

Figure 1: Overview of the approach

the defined control flow are generated with the help of *transformers* which encode specific trace properties that are either enforced or violated on a random basis. Together with the ability to subsequently generate traces of a series of different business process models into a single log, the approach provides for a definition of an attacker model, which can be configured in a detailed way for a particular test case. Generated logs are outputted in MXML (a format for process mining), as well as plain text. Ongoing work adds support to Extended Event Streams format (XES).

Processes subject to simulation are considered to be executed within a *context*. While the control flow of a process defines possible execution traces, subjects authorized to execute process activities and objects used by activities are defined by contexts. Our approach allows to define *simulation runs*, each relating to one process model which are processed one after another according to the number of desired traces for each run. This way, the engine is capable of simulating situations in which there is an initial model for planned process behavior, but a variant of this model was executed for some time, possibly due to the activity of an attacker or process variation/flexibility. To support the user in defining such variants, we define model transformation operators that replace specific control flow patterns of a process according to Weber et al. [RW12] (see Section 4 below). Additionally, each simulation run can relate to a set of *trace transformers* which operate on trace level.

During the processing of a simulation run, the engine generates valid log traces according to the control flow of the corresponding process and context and then passes them through the trace transformers which apply transformations in a post-processing manner. Trace transformers can remove or add process activities (simulating skipped activities or incomplete logging), as well as change information within traces in a way business related properties like *separation/binding of duties* or authorization constraints are enforced or violated.

3 Process Specifications and Execution Contexts

The main building blocks are, firstly, the process specification which defines activities and the order in which they can be executed and, secondly, the execution context which defines the organizational environment for process execution in terms of subjects authorized to execute process activities and objects, used by activities. For simulation purposes, a process specification and a compatible context are paired together to an execution system.

Process specification The process specification defines process activities and the order in which they are executed. To focus on the simulation procedure, we abstract from concrete specification meta-models at this point. The only prerequisite is that processes are well-structured [VVK09]: a *well-structured* process model is a model composed of blocks, which may be nested but must not overlap. A block can be a single activity, a sequence, a parallel/conditional branching and a loop block.
A process is defined as a tuple $P = (A_P, T_P)$, where A_P is a set of activities and $T_P \subseteq A_P^*$ is the set of possible traces over A_P. It is assumed that the control flow of a process which defines T_P by establishing a successor-relation among activities is modeled with the help of OR-, XOR- or AND-gateways (implicitly or explicitly). These gateways are logical connectors modeling choice, combination and parallelism of activities in the conventional way. In case a process contains more than one gateway of the same kind, they are enumerated in a natural way (e.g. OR_1, OR_2). Such process models (specifications) are instantiated when processes are executed. Each instance relates to a possible execution sequence of process activities (trace). The set of all processes is denoted as $\mathbb{P}$.

Activities are executed by subjects at a specific point in time and involve a number of objects. Subjects, objects and clearances (rights of subjects to execute activities) are defined within process contexts $C = (A_C, S_C, O_C, \mathcal{A}_C, \mathcal{O}_C)$, where A_C is a set of activities, S_C is a set of subjects and O_C is a set of objects. The function $\mathcal{A}_C: S \to \mathcal{P}(A_C)$ defines the set of activities a subject is authorized to execute and the function $\mathcal{O}_C: A \to \mathcal{P}(O_C)$ assigns the objects employed by activities.

The execution of process instances leads to the occurrence of events, each reporting the execution of a process activity. Formally, an event is a triple $E = (Z_E, A_E, S_E)$, where $Z_E \in \mathbb{N}$ is a unique timestamp, $A_E \in A_P$ is an activity and $S_E \in S_C$ is the executing subject. For convenience, we use $e.time$ to access the timestamp of an event, $e.activity$ and $e.subject$ respectively. In contrast to process traces that just consider activity orders, log traces denote the sum of all events relating to the execution of a process instance. A log trace is defined as sequence of events $t = \langle e_1, \dots, e_n \rangle$, where the sequence order is defined according to the event timestamps $e_i.time$. The set representation of a log trace is defined as t_E. Log trace t is a sub-trace of another log trace $t' = \langle e'_1, \dots, e'_n \rangle$, denoted $t \sqsubseteq t'$, iff $t_E \subseteq t'_E$ We use the notation $e^{\leftarrow t}$ for the prefix of t before the occurrence of e and $e^{\rightarrow t}$ for the suffix after the occurrence of e, whereas $e^{\ni t}$ denotes the direct predecessor of e within t, $e^{\in t}$ the direct successor. The set of all log traces that can be formed based on activities A_T and subjects S_T is denoted

as $\mathbb{T}_{A_T,S_T}$, for events $\mathbb{E}_{A_T,S_T}$ respectively. We use the notation $t[i]$ to access the i-th event of a log trace t. The activity reduction of a log trace is defined as $t|_A = \langle a_1, \dots, a_n \rangle, a_i = e_i.activity$. The function $\theta: \mathcal{P}(\mathbb{E}_{A_T,S_T}) \times A_T \to \mathcal{P}(\mathbb{E}_{A_T,S_T})$ determines the reduction of a set of events to those with a specific activity, i.e. $\theta(\{e_1, \dots, e_n\}, a) = \{e_i | e_i.activity = a\}$.

Execution System An execution system is a pair (P, C), where $P = (A_P, T_P)$ is a process and $C = (A_C \supseteq A_P, S_C, O_C, \mathcal{A}_C, \mathcal{O}_C)$ is a process context. A possible trace of an execution system is denoted $t_{(P,C)} \in \mathbb{T}_{A_P,S_C} \subseteq \mathbb{T}_{A_C,S_C}$ and contains only events relating to activities in P and subjects in C, i.e. $\forall (z, a, s) \in t_{(P,C)}: a \in A_P \wedge s \in S_C$. We use $\mathbb{T}_{(P,C)}$ for the set of all possible traces of an execution system. A possible trace t' of an execution system is *conform*, if it is a valid trace of the corresponding process and subjects are assigned according to the authorization function $\mathcal{A}_C$, i.e. $t'|_A \in T_P \wedge t'.activity \in \mathcal{A}_C(t'.subject)$. The set of all conform traces of an execution system is denoted as $\mathbb{T}^C_{(P,C)} \subseteq \mathbb{T}_{(P,C)}$. A process log resulting from executing activities of process instances with respect to an execution system is defined as a sequence of traces $L_{(P,C)} = \langle t_0, t_1, \dots, t_n \rangle, t_i \in \mathbb{T}_{(P,C)}$, where the ordering of sequence elements is defined by case starting times, i.e. $t_i[1].time$.

4 Model- and Trace-Transformations

Model transformations regard only the control flow (structure) of a process and change the order in which activities can be executed and are determined with the help of transformation functions $\Theta_T: \mathbb{P} \to \mathbb{P}$. Currently, the following transformation operators at gateway-level are considered:

- Θ_{And2Xor}: This transformation provides a variation of the input process in which a specific AND-gateway is rewritten to an XOR-gateway. This means: while the original process requires traversing both parts of the AND-gate, the variant assumes, that only one (randomly chosen) path is traversed.
- Θ_{Xor2And}: The opposite of Θ_{And2Xor}, i.e. a specific XOR-gateway is rewritten to an AND-gateway.
- Θ_{Swap}: This transformation provides for variants in which the order of a pair of activities is swapped.

In contrast to changing the behavior of a process persistently by modifying its structure, trace transformers are used to transform traces in a way specific trace properties are enforced or violated. Transformers operate on valid traces generated during the simulation process in a post-processing manner, before they are added to the output log file. Formally, a transformer wrt an execution system $(P, C) = ((A_P, T_P), (A_C, S_C, O_C, \mathcal{A}_C, \mathcal{O}_C))$ is a function $\Omega_F: \mathbb{T}^C_{(P,C)} \times I_1 \times \dots \times I_k \to T_F \subseteq \mathbb{T}_{A_P,S_C}$ which takes a valid process trace as input and generates a modified version of this trace according to a number of parameters I, which is not necessarily valid. The set of possible events to apply the transformer is denoted E^S. The set of events, for which the

transformer is actually applied is denoted $E_\pi^S \subseteq E^S$. The modified events resulting from applying the transformer is denoted E'^S_π. Below we define a number of ftransformer that can be applied while simulating processes. It should be noted that in situations where more than one transformer is applied on a trace, the fields modified by the transformers have to be "locked". This is to prevent further transformers from corrupting the already enforced properties coming from previous applications of transformers.
Currently, the approach comprises the following six trace transformers: (a) *delay* inserts a delay in the process execution; (b) *skip* generates executions in which some activities are skipped; (c) *silent* captures a situation in which a particular activity has not been logged onto the file; (d) *authentication* mimics an access control policy and its violation; (e) *binding of duty* (BOD) mimics the compliance with or the violation of a binding of duty requirement; and (f) *separation of duties* can be seen as the opposite of BOD: it states that a particular activity must be performed by a *different* subject or role. Due to space constraints, below we present only three transformers – namely, delay, authentication and separation of duties. The definition of the other transformers occur analogously. It is important to emphasize, though, that this list of transformers can be extended with or refined to other domain-specific transformers and, subsequently, added to SecSy .

Delay ***transformer.*** The delay transformer inserts a delay in the process execution. Formally, the transformer is defined as $\Omega_{\text{delay}}: \mathbb{T}^C_{(P,C)} \times A \subseteq A_P \times \mathbb{N} \times \mathbb{R} \times \mathbb{N} \to \mathbb{T}^C_{(P,C)}$. The transformer $\Omega_{\text{delay}}(t, a, \delta, \lambda, \omega)$ transforms an input trace $t = \langle e_1, \dots, e_n \rangle$ into an output trace $t' = \langle e'_1, \dots, e'_n \rangle$ by adding a random delay $\varepsilon \in [\delta - \lambda; \delta + \lambda]$ to events relating to activity a. The number of events which are delayed is bounded by the parameter ω, whereas $E^S = \{e \in t \mid e.activity = a\}$ and $0 \leq |E_\pi^S| \leq min(\omega, |E^S|)$. Note that E^S can be empty. All events e'_i of the generated output trace fulfill the following properties:

- $e'_i.activity = e_i.activity$
- $e'_i.subject = e_i.subject$
- $e'_i.time = \begin{cases} e_i.time + \sum_{e_j \in e_i^{\leftarrow t}} (e'_j.time - e_j.time) + \varepsilon in[\delta - \lambda; \delta + \lambda] \\ \text{iff } e_i \in E_\pi^S \\ e_i.time + \sum_{e_j \in e_i^{\leftarrow t}} (e'_j.time - e_j.time) \\ \text{otherwise} \end{cases}$

Authentication transformer. The authentication transformer mimics an access control policy and its violation. Formally, the transformer is defined as $\Omega_{\text{auth}}: \mathbb{T}^C_{(P,C)} \times \mathbb{N} \to \mathbb{T}_{(P,C)}$. The transformer $\Omega_{\text{auth}}(t, \omega)$ transforms an input trace $t = \langle e_1, \dots, e_n \rangle$ into an output trace $t' = \langle e'_1, \dots, e'_n \rangle$ by randomly assigning subjects to events, which are not authorized to execute the event activity. The number of events for which the transformer is applied is bounded by the parameter ω, whereas $E^S = t_E$ and $0 \leq |E_\pi^S| \leq min(\omega, |E^S|)$. Note that the transformer fails when E_π^S is empty and subjects cannot be assigned accordingly (e.g. when all subjects are allowed to execute all activities). All events e'_i of the generated output trace fulfill the following properties:

- $e'_i.activity = e_i.activity$
- $e'_i.time = e_i.time$
- $e'_i.subject \in S_C \backslash \mathcal{A}_C(e_i.activity)$ iff $e' \in E^S_{\pi'}$
- $e'_i.subject = e_i.subject$, otherwise

Separation-of-duties transformer. The separation-of-duties (SOD) can be seen as the opposite of the aforementioned BOD transformer: it states that each activity in an activity set must be performed by a *different* subject or role. The transformer is formally defined as $\Omega_{\text{SoD}}: \mathbb{T}^C_{(P,C)} \times \mathcal{P}(A_P) \times \mathbb{N} \times \{enforce, violate\} \to \mathbb{T}^C_{(P,C)}$. The transformer $\Omega_{\text{SoD}}(t, A, \lambda)$ transforms an input trace $t = \langle e_1, \dots, e_n \rangle$ into an output trace $t' = \langle e'_1, \dots, e'_n \rangle$ by enforcing or violating the *separation of duties* security property on trace t. Formally, the SOD property on a set of activities M, whereas $S: M \to S_C$ denotes the executing subject is defined as $|\bigcup_{m \in M} S(m)| = |M|$. In the *enforcement* case, applied on a given trace t and a set of *bindings* $A = \{a_1, \dots, a_n\}$, this translates to:

- $E^S = \{e \in t \mid e.activity \in A\}$
- $|E^S_\pi| \overset{!}{=} |E^S|$
- $A_\phi = \{(s_1, \dots, s_n) \mid s_i \in \bigcap_{e \in \theta(E^S, a_i)} \mathcal{A}_C(e)\}, s_i \neq s_j$
- $A_\phi \neq \emptyset$
- $\gamma = (\gamma_1, \dots, \gamma_n) \in A_\phi$ (conform subject configuration)
- $\forall a_i \in A \ \ \forall e' \in \theta(E^S_{\pi'}, a_i): e'.subject = \gamma_i$
- $\psi: A \leftrightarrow S_C \ , \ \psi(a_i) \mapsto \gamma_i$

The enforcement fails, if there is no conform subject assignment which enforces the property, i.e. $A_\phi = \emptyset$. All events e'_i of the generated output trace fulfill the following:

- $e'_i.activity = e_i.activity$
- $e'_i.time = e_i.time$
- $e'_i.subject = \begin{cases} \psi(e'.activity) & , e' \in E^S_{\pi'} \\ e_i.subject & , \text{otherwise} \end{cases}$

In the *violation* case, the transformer ensures that there are events with activities in A with the same subject: it first enforces the SOD property as stated above and then randomly chooses a set of events $V = \{v_1, \dots, v_u\} \subseteq E^S_{\pi'}$ for which it assigns the same authorized subject, i.e. $v_1.subject = \dots = v_n.subject$. The violation fails if there is no conform subject assignment which violates the property. This procedure allows for a more realistic simulation of SOD violations than a simple random choice of different authorized subjects.

5 SecSy: Prototypical Realization and Initial Evaluation

The event log synthesis approach based upon the framework presented in Section 3-4 has been implemented as a standalone, extensible Java application, called SecSy that can be downloaded at http://bit.ly/SecSy). Figure 2 depicts the configuration panel of the application. The simulation settings require the path for the generated log file and its name, as well as the simulation type (SIMPLE for traces containing only timestamps and activity names and EXTENDED for traces containing also information about executing subjects and data items) and the output log format (plain text or MXML). The latter is a standard input for process mining technologies, thereby serving as a basis for generating logs to test with process mining tools.

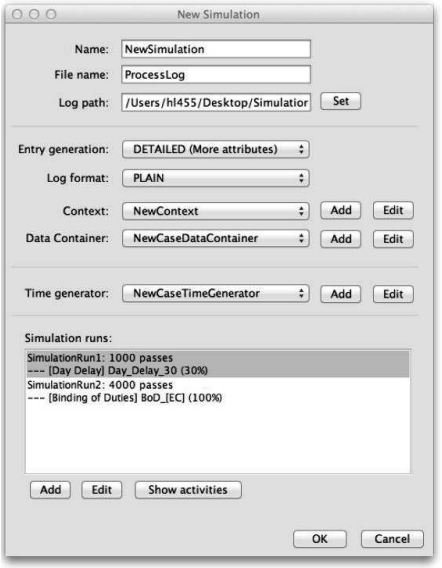

Figure 2: Screenshot of the configuration panel

The tool allows for flexible configuration of all required parameters and the creation and editing of corresponding components (Fig. 3). The time generator component contains all timing related simulation properties, including the start time for the simulation (date for first trace), the number of cases per day, office-days and -hours, as well as individual activity durations and delays between succeeding activities. All properties can be randomized by adding deviation bounds. The possibility to add several simulation runs is useful for simulating the execution of different process models or different property enforcements along time. To model the requirements that should (not) hold for single simulation runs (thereby specifying the policy applicable to the process), the user can assign different transformer configurations.

This way, the frequency and kind of security violations can be thoroughly configured. In the case of simulation type EXTENDED , the user has to specify a context and a data container. Data containers generate values for data items used during process execution (e.g. credit amount) and store the values until a trace is completed. This way, the consistent usage of attribute values along a complete trace is ensured. A context holds subjects and their permissions to execute activities and access data elements together with activity data usage (attributes used by activities) information. To specify subject permissions, the user can choose between an access control list or a role based concept, which is particularly helpful for large contexts. Additionally, a context allows to specify constraints on attribute values that can be added to process activities. This influences the generation of process traces in a sense, that only those activities are executed whose constraints are fulfilled. On Petri net level (which is the used meta-model for process models), the set of enabled transitions is reduced according to existing constraints, when the simulation tool decides about the next transition to fire.

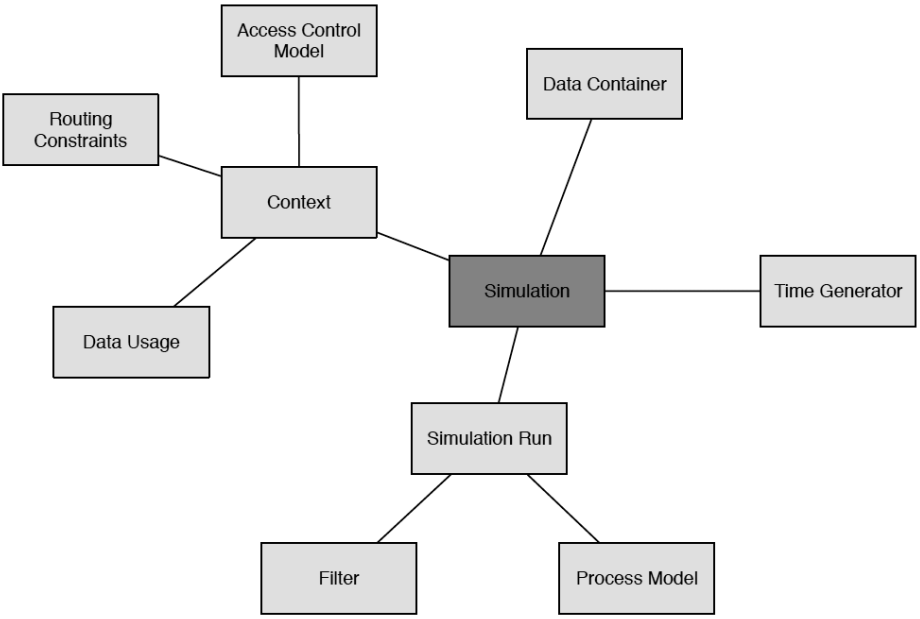

Figure 3: Configuration of simulation runs

Evaluation. We have carried out both a *quantitative* and *qualitative* evaluation of SecSy . The quantitative evaluation regards the overall performance of SecSy , which is acceptable for synthesizing large event logs. As an example, a process log containing 100K traces of a non-trivial loan application process with 19 activities incorporating 50 subjects organized in 4 different roles takes 24 seconds including the enforcement of a binding of duties constraint on a pair of process activities (3GHz Intel Core i7, 8GB DDR3 memory under OSX 10.8). For comparison, the generation of 10K traces in the same configuration takes 3 seconds. Future work will further qualitatively evaluate

SecSy in other dimensions. For instance, the interplay between the output format chosen (plain text or MXML) and the performance and limits to the generation of event log files has to be clarified. Such an evidence is necessary for the generation of "big log files". Further, detailed sensitivity analysis of different context parameters (e.g. number of subjects and roles) have to be carried out.

The qualitative evaluation aims at showing that the logs produced with SecSy in fact correspond to the original configuration setup for model- and trace-transformations. Put another way, checking whether the transformations are effective. For this, we generate log files employing, for each, the same context configuration but different transformers. Further, we use PROM [vdA11, Chap.~10] to check whether these parameters are fulfilled or not. For example, to validate the effectiveness of the SOD transformer, we first configure the transformer so that, say, 20 percent of the synthesized process runs violate this requirement: subject S performs both activities X and Y. Subsequently, we employ the LTL -checker plug-in to isolate the cases in which this requirement is violated. For this, it is sufficient to use the predefined formula "*exists person doing task X and Y*". Similar checks employing other techniques – e.g. conformance checking and process discovery – were carried out for all the transformations, so that we could validate their effectiveness. Clearly, this is an ad-hoc, empirical validation style based upon the assumption that process models respect the designated structural constraints. Further work aims at obtaining well-founded evidence that transformations are effective.

6 Related Work

Business process simulation is an important tool in business process management to assessing the runtime operation of a business process. Simulation can also be used as a means to synthesize traces to test analysis techniques and tools, e.g. conformance checking and process mining. In both cases, it is important to insert deviations from the prescribed behavior in order to achieve variability (c.f. [vdA10a, vdA10b] and [DRMR13, Chap. 7.4]).

However, the current state of the art consider either variations on the execution time and workflow of the process or control flow deviations mimicking process flexibility and dynamics. Nakatumba et al. [NWvdA12] present an approach to generating event logs with workload-dependent speeds from simulation models. Mans et al. [MRvdA+10] employ simulation to analyze the impact of a schedule-aware workflow management system. Schonenberg et al. [SJSvdA10] devise a simulation approach for the analysis of business trends. Rozinat et al. [RMSvdA09] proposes an approach to discovering simulation models. These works consider, taking our transformers, delays and workload changes, but they do not consider security and compliance policies. Other works propose business process simulation frameworks (e.g. [CKH98, Tum96]) without actually testing them. Weber et al. [WPZW11, Ala] propose an simulation approach and tool support that considers flexible business processes. However, also here one cannot configure policy violations.

Several tools exist to simulate business processes (see [JVN06] for survey up to 2006). Bahrami [BSB98] present a special purpose tool developed at IBM. The focus of the tool

was on the enterprise architecture and not on processes. Burattin and Sperduti [BS10] present a framework for the generation of business process models and their execution logs. However, the tool does not offer any control over the generated models and event logs. The AristaFlow BPM Suite provides support for simulation, however it is unclear whether compliance and security rules can be considered [RW12, Chap.~15]. The CPNTools toolset provides for a highly configurable simulation environment. However, the toolset is not designed exclusively for this purpose, requiring the programming or definition of the aforementioned transformers. Our tool provides them in a natively and extensible manner.

Overall, the generation of "defect" data has been applied to software process improvement in general [RTVA12], but has never been seen in the BPM area. This paper is a first step in this direction. We firmly believe that the controlled, push-button generation of (large) test data is a promising research direction and application domain in business process testing and improvement. Not only for testing mechanisms in the context of security and compliance (as argued in this paper), but also for resilience [KS11, WL07], visualization [KRM12] and dependability [WLR+09] engineering for business process and process-aware information systems.

7 Summary and Further Work

This paper outlined a method for simulating business process flexibility and noncompliance, thereby synthesizing event logs. Overall, this approach is novel, both in the security, compliance and business process management communities. It allows for highly-controllable simulation output incorporating conform process behavior with respect to targeted security properties, as well as random failures that may lead to the violation of selected properties and incomplete/distorted process logs. The approach has been realized in a prototype and the results obtained with it are promising and useful in practice.

We firmly believe that the automated generation of such event logs is a necessary step toward the development of tools for business process analysis. This not only applies to standard process-aware information systems. More importantly, with the advent of big data and business process management "in the large", there will be the need to test with such log data. Today, generating these data is very tedious. Our proposal should, in the future, be also used for this setting, thereby considering process architectures (instead of single models) and event-intensive simulation.

Besides the ongoing and future works mentioned above, there are several other issues that need to be added to obtain a fully-fledged log synthesizing tool. *Firstly*, the simultaneous execution of parallel process models, thereby allowing for the trace interleaving. *Secondly*, the completeness of generated logs has not been investigated, i.e. are all the possible cases and policy violations considered while generating the log. *Thirdly*, soundness should be guaranteed by construction, but will be investigated formally in the future. *Finally*, we plan to extend the set of policies. In particular, we will employ the policy patterns provided by Dwyer et al. [DAC99] to provide for policy templates and allow their specification before execution.

References

[Acc13] Rafael Accorsi. Sicherheit im Prozessmanagement. *digma Zeitschrift für Datenrecht und Informationssicherheit*, 2013.

[AL12] Rafael Accorsi and Andreas Lehmann. Automatic Information Flow Analysis of Business Process Models. In *Conference on Business Process Management*, volume 7481 of *Lecture Notes in Computer Science*, pages 172–187. Springer, 2012.

[Ala] Alaska Simulator. http://www.alaskasimulator.org, Last seen: June 2, 2013.

[AS08] Rafael Accorsi and Thomas Stocker. Automated Privacy Audits Based on Pruning of Log Data. In *EDOC International Workshop on Security and Privacy in Enterprise Computing*. IEEE, 2008.

[AS12] Rafael Accorsi and Thomas Stocker. On the Exploitation of Process Mining for Security Audits: The Conformance Checking Case. In *ACM Symposium on Applied Computing*, pages 1709–1716. ACM Press, 2012.

[AS13] Rafael Accorsi and Thomas Stocker. On the Exploitation of Process Mining for Security Audits: The Process Discovery Case. In *ACM Symposium on Applied Computing*. ACM Press, 2013.

[ASK08] Rafael Accorsi, Yoshinori Sato, and Satoshi Kai. Compliance Monitor for Early Warning Risk Determination. *Wirtschaftsinformatik*, 50(5):375–382, 2008.

[AW10a] Rafael Accorsi and Claus Wonnemann. Auditing Workflow Executions against Dataflow Policies. In *Business Information Systems*, volume 47 of *Lecture Notes in Business Information Processing*, pages 207–217. Springer, 2010.

[AW10b] Rafael Accorsi and Claus Wonnemann. Static Information Flow Analysis of Workflow Models. In *Conference on Business Process and Service Computing*, volume 147 of *Lecture Notes in Informatics*, pages 194–205. GI, 2010.

[AW11] Rafael Accorsi and Claus Wonnemann. Strong Non-Leak Guarantees for Workflow Models. In *ACM Symposium on Applied Computing*, pages 308–314. ACM, 2011.

[BAS09] Travis Breaux, Annie Antón, and Eugene Spafford. A distributed requirements management framework for legal compliance and accountability. *Computers & Security*, 28(1-2):8–17, 2009.

[BS10] Andrea Burattin and Alessandro Sperduti. PLG: A Framework for the Generation of Business Process Models and Their Execution Logs. In *Business Process Management Workshops*, volume 66 of *Lecture Notes in Business Information Processing*, pages 214–219. Springer, 2010.

[BSB98] Ali Bahrami, Deborah A. Sadowski, and Soheila Bahrami. Enterprise Architecture for Business Process Simulation. In *Winter Simulation Conference*, pages 1409–1414, 1998.

[Cer01] Iliano Cervesato. The Dolev-Yao Intruder is the Most Powerful Attacker. In *Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, 2001.

[CKH98] Yoon Ho Cho, Je Keyong Kim, and Kim Soung Hie. Role-based approach to business process simulation modeling and analysis. *Computers & Industrial Engineering*, 35(1/2):343–346, 1998.

[CPN] CPNTools. http://cpntools.org/.

[DAC99] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in Property Specifications for Finite-State Verification. In *Conference on Software Engineering*, pages 411–420. ACM, 1999.

[DRMR13] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013.

[JH11] Yue Jia and Mark Harman. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.

[JVN06] Monique Jansen-Vullers and Mariska Netjes. Business process simulation – A tool survey. In *Workshop and Tutorial on Practical Use of Coloured Petri Nets and the*

CPN Tools, 2006.

[KR01] Konstantin Knorr and Susanne Röhrig. Security Requirements of E-Business Processes. In *IFIP Conference on E-Commerce, E-Business, E-Government*, volume 202 of *IFIP Conference Proceedings*, pages 73–86. Kluwer, 2001.

[KRM12] Simone Kriglstein and Stefanie Rinderle-Ma. Change Visualizations in Business Processes - Requirements Analysis. In *Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications*, pages 584–593. SciTePress, 2012.

[KS11] Thomas Koslowski and Jens Strüker. ERP On Demand Platform - Complementary Effects Using the Example of a Sustainability Benchmarking Service. *Business & Information Systems Engineering*, 3(6):359–367, 2011.

[LA11] Lutz Lowis and Rafael Accorsi. Vulnerability Analysis in SOA-Based Business Processes. *IEEE Transactions on Service Computing*, 4(3):230–242, August 2011.

[MRvdA+10] Ronny Mans, Nick C. Russell, Wil M. P. van der Aalst, Piet J. M. Bakker, and Arnold J. Moleman. Simulation to Analyze the Impact of a Schedule-aware Workflow Management System. *Simulation*, 86(8-9):519–541, 2010.

[NWvdA12] Joyce Nakatumba, Michael Westergaard, and Wil M. P. van der Aalst. Generating Event Logs with Workload-Dependent Speeds from Simulation Models. In *CAiSE Workshops*, volume 112 of *Lecture Notes in Business Information Processing*, pages 383–397. Springer, 2012.

[RMSvdA09] Anne Rozinat, R. S. Mans, Minseok Song, and Wil M. P. van der Aalst. Discovering simulation models. *Inf. Syst.*, 34(3):305–327, 2009.

[RTVA12] Anu Raninen, Tanja Toroi, Hannu Vainio, and Jarmo J. Ahonen. Defect Data Analysis as Input for Software Process Improvement. In *Conference on Product-Focused Software Process Improvement*, volume 7343 of *Lecture Notes in Computer Science*, pages 3–16. Springer, 2012.

[RW12] Manfred Reichert and Barbara Weber. *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*. Springer, 2012.

[SJSvdA10] Helen Schonenberg, Jingxian Jian, Natalia Sidorova, and Wil M. P. van der Aalst. Business Trend Analysis by Simulation. In *Conference on Advanced Information Systems Engineering*, volume 6051 of *Lecture Notes in Computer Science*, pages 515–529. Springer, 2010.

[Tum96] Kerim Tumay. Business Process Simulation. In *Winter Simulation Conference*, pages 93–98, 1996.

[vdA10a] il M. P. van der Aalst. Business Process Simulation. In *Handbook on Business Process Management*, volume 1, pages 313–338. Springer, 2010.

[vdA10b] Wil M. P. van der Aalst. Business Process Simulation Revisited. In Joseph Barjis, editor, *Enterprise and Organizational Modeling and Simulation*, volume 63 of *Lecture Notes in Business Information Processing*, pages 1–14. Springer, 2010.

[vdA11] Wil M. P. van der Aalst. *Process Mining – Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[VVK09] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. The refined process structure tree. *Data Knowl. Eng.*, 68(9):793–818, 2009.

[WL07] Qihua Wang and Ninghui Li. Satisfiability and Resiliency in Workflow Systems. In *Proceedings of the European Symposium On Research In Computer Security*, volume 4734 of *Lecture Notes in Computer Science*, pages 90–105. Springer, 2007.

[WLR+09] Branimir Wetzstein, Philipp Leitner, Florian Rosenberg, Ivona Brandic, Schahram Dustdar, and Frank Leymann. Monitoring and Analyzing Influential Factors of Business Process Performance. In *IEEE International Enterprise Distributed Object Computing Conference*, pages 141–150. IEEE Computer Society, 2009.

[WPZW11] Barbara Weber, Jakob Pinggera, Stefan Zugal, and Werner Wild. Alaska Simulator Toolset for Conducting Controlled Experiments on Process Flexibility. In *Information*

Systems Evolution, volume 72 of *Lecture Notes in Business Information Processing*, pages 205–221. Springer, 2011.