Process Mining meets Statistical Model Checking: Towards a Novel Approach to Model Validation and Enhancement

Casaluce, Roberto ; Burattin, Andrea; Chiaromonte, Francesca; Vandin, Andrea

Published in:
Business Process Management Workshops

Link to article, DOI:
10.1007/978-3-031-25383-6_18

Publication date:
2023

Document Version
Peer reviewed version

Process Mining meets Statistical Model Checking: Towards a Novel Approach to Model Validation and Enhancement

Roberto Casaluce[1,2], Andrea Burattin[3],
Francesca Chiaromonte[2,4], and Andrea Vandin[2,3]

[1] Univeristà di Pisa, Pisa, Italy
[2] Institute of Economics and EMbeDS, Sant'Anna School of Adv. Studies, Pisa, Italy
[3] DTU Technical University of Denmark, Kgs. Lyngby, Denmark
[4] Dept. of Statistics and Huck Institutes of the Life Sciences, Penn State Univ., USA

Abstract. We propose a novel research line integrating Statistical Model Checking (SMC), a family of simulation-based analysis techniques from quantitative formal methods, with Process Mining (PM), a collection of data-driven process-oriented techniques. SMC and PM are complementary. SMC focuses on performing the *right number* of simulations to obtain statistically-reliable estimations (e.g., the probability of success of an attack). PM focuses on reconstructing a model of a system using logs of its traces. Nevertheless, both approaches aim at providing evidence of issues/guarantees of the system, and at proposing enhancements.
We aim at enriching SMC by *explaining why* it produced specific estimates. This might help, e.g., identifying issues in the model (validation) or suggesting improvements (enhancement). Given that SMC uses statistics to decide what is the *correct* number of simulations (or *traces*), we avoid by-construction the complex issue of under-representation of system behavior in the logs crucial to many PM exercises.
This work-in-progress paper demonstrates the proposed methodology and its usefulness using a simple example from the security threat modeling domain. We show how PM helps highlighting both mistakes in the model, and possibilities for improvement.

Keywords: Process Mining · Statistical Model Checking · Validation

1 Introduction

We present novel research integrating the simulation-based analysis technique from quantitative formal methods known as statistical model checking (SMC) [2], with the data- and process-driven techniques known as process mining (PM) [1].

Specifically, we aim at formulating a novel framework capable of making analyses results typical of SMC, and of simulation-based analysis in general, more explainable and understandable. Our framework will empower modelers to actually *see* the unfolded behavior of their models, as opposed to just numerical aggregated values. This will pave the way to new possibilities in terms of debugging and validating the models of interest. Considering the widespread use of simulation models, having tools to debug and validate them will represent a potentially very impactful contribution among several disciplines.

2 Preliminaries and Related work

2.1 Attack-Defense Trees

We briefly introduce the domain of risk modeling and analysis with so-called *attack-defense trees*. Attack-defense trees (ADT) and their variants [19,16,13] allow to represent security scenario by means of intuitive visual language constructs. These aim at providing means for specifying vulnerabilities and countermeasures, their interplay, together with quantitative aspects such as cost and effectiveness. The goal is to support policy- and decision-making in determining, e.g., the degree of vulnerability to specific attacks, based on the resources of attackers, or whether given defensive resources are cost-effective. Attack trees are widely used in several domains like, e.g., defense (e.g., their use is recommended by NATO [18]), aerospace [22], or safety-critical cyber-physical systems [12].

We present a simple running example for the risk assessment of a *"bank robbery"* scenario extrapolated from [5] as depicted in Fig. 1. We see that ADT are graphs whose nodes represent either attack goals or defensive measures, and sub-trees represent nodes' refinements. The root (`RobBank`), is the threat under analysis. In order to achieve it, we shall first succeed in opening the vault (`OpenVault`), *or* blowing it up (`BlowUp`), or both. Refinements might also come with other Boolean conjunctions like *and*, *xor*, etc. The figure also shows another kind of refinement, denoted by a grey dashed edge: `RobBank` attempts can be mitigated by the countermeasure `LockDown`. This is a reactive defense that might be turned on by `BlowUp` attempts (dashed blue arrow).

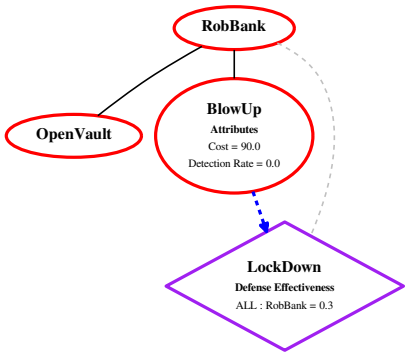


Fig. 1: Attack-defense Tree for a simple bank robbery scenario

Recently (e.g., [4,14,3]), ADT have been extended with *attacker profiles*, explicit attacker behaviours acting on the security scenario described by the ADT. This allows to assess systems against specific classes of attackers (e.g., large organizations with rich resources, or lone wolves). E.g., RisQFLan [4] allows one to specify *probabilistic attacker behaviours*. An example is shown in Fig. 2. We can see that the attacker can be in four *states*: `Start`, `TryOpenVault`, `TryBlowUp`, and `Complete`. When s/he is in state `Start`, s/he can decide to attempt `OpenVault` or `BlowUp` strategies by changing in states `TryOpenVault`, or `TryBlowUp`, respectively. This is done by performing *action* `chooseOV` with weight 2, or `chooseBU` with weight 4, respectively. Alternatively, if *allowed* by the ADT (i.e., if `OpenVault` or `BlowUp` attempts have already previously succeeded), the attacker can actually attempt to rob the bank by moving in state `Complete`. In particular, the attack will succeed (`succ(RobBank)`) with weight 3, or `fail(RobBank)` with weight 1. Likewise, from state `TryOpenVault` or `TryBlowUp`, the corresponding attacks will be attempted as dictated by the transitions with actions `succ` and `fail`. We note

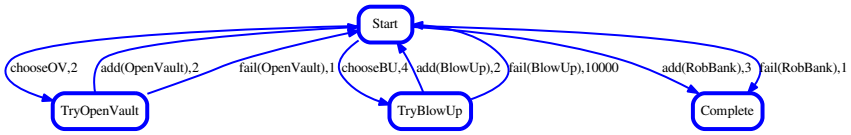


Fig. 2: Probabilistic attacker behaviour

that the weight for `fail(BlowUp)` is particularly high, this will be instrumental to our analyses in Section 4.

The weights are used to compute the probability with which each transition is executed, allowing to obtain probabilistic simulations of the attacker behaviour modulo the constraints by the ADT. More details on the RisQFLan language, on the further quantitative attributes of nodes shown in Fig. 1, and on the simulation of RisQFLan models will be provided in Section 4.

2.2 (Black-box) Statistical Model Checking

We introduce a family of simulation-based analysis techniques that comes under the name of (black-box) statistical model checking (SMC). SMC [2], consists in performing a *sufficient* number of probabilistic simulations of a model to obtain *statistically reliable estimations* of model properties. Black-box SMC (e.g., [21,25,24]) is a fragment of SMC where no assumption is made on the studied model, only that probabilistic simulations of it can be performed. E.g., MultiVeStA [24,20,10] is a black-box SMC tool that can be integrated with existing simulators to enrich them with automated statistical analysis techniques. The idea is simple: to each simulation we assign a real value, e.g., 0 or 1 if the attacker succeeds or fails, resp., in robbing the bank within the first 60 simulation steps. Technically, this is a random variable X in the interval $[0, 1]$ over simulations of the model. Notably, the expected value $p = E[X]$ of X corresponds to the probability of success of the event (the attacker succeeds within the first 60 steps of simulation). As discussed in [24,20], MultiVeStA estimates such expected values $E[X]$ as the mean $\overline{x}$ of n independent simulations, with n large enough to guarantee an (α, δ) *confidence interval* (CI) [15, Chapter 9]. In other words, MultiVeStA guarantees that $E[X]$, i.e., the studied probability, belongs to the interval $[\overline{x} - \delta/2, \overline{x} + \delta/2]$ with statistical confidence $(1-\alpha)\cdot 100\%$. δ is a parameter chosen by the user that gives a sort of *precision* on the performed estimation. Instead, roughly, α is related to the probability that the studied probability actually belongs to the computed interval. The interesting part is that the *correct* number of simulations to be performed for a given property and CI is chosen by MultiVeStA by using standard statistical machinery [15, Chapter 9].

Black-box SMC and MultiVeStA can estimate more complex properties. However, for the sake of presentation, here we focus on simple ones like the mentioned one. MultiVeStA has been successfully applied to a wide range of domains, including, e.g., security risk modeling [4], economical agent-based models [24], highly-configurable systems [5,23], public transportation systems, [11,8], business process modeling [9]. robotic scenarios with planning capabilities [6],

and crowd steering scenarios [17]. This has been made possible by the fact that black-box SMC can be applied to virtually any simulation model. E.g., MultiVeStA only requires to implement a simulator-specific adaptor to perform basic operations such as `reset`: reset simulator to do a new simulation, providing a new random seed; `oneStep`: perform one step of simulation; and `eval`: evaluate an observation on the current simulator state. However, this high generality might come at the cost of low interpretability of results. Indeed, it might be difficult to provide *behavioral explanations* about *why* a property is estimated to a given value. SMC approaches like, e.g., [7], partially address this by providing a counterexample – an example of *problematic/relevant* simulation. However, to the best of our knowledge, no SMC technique has ever been integrated with rich support for describing and reasoning upon the *whole behavior* that led to a specific result. It is relevant to highlight that the methodology presented in this paper can be applied to any simulation model and SMC tool, since it does not rely on the internal mechanics of the analyzer or of the model, but exploits logs of the computed simulations. In the next section we present a process-oriented data-driven family of techniques known as process mining that we combine with SMC to offer rich *white-box behavioral* interpretability of the analysis results.

2.3 Process Mining

Process Mining (PM) is the scientific discipline bridging the gap between data science and process science [1]. It aims at using actual executions of a process to infer relevant information about how the underlying behavior is observed in action (as opposed to the intended behavior). PM consists of three main activities: (control-flow) discovery, enhancement, and conformance checking. *Discovery* aims to identify an abstract representation of the executed process by combining all the observed instances into a single model. Such a model can be enhanced with additional information (e.g., how often activities/paths are executed). This is called *enhancement*. Finally, *conformance checking* tries to understand the extent to which a normative model is violated in actual executions.

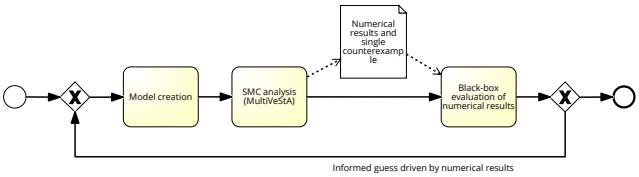
In this paper, we are interested in the discovery and enhancement activities since we aim at consuming the execution traces coming from SMC analyses to see whether the generated behavior is aligned with the original expectations.

3 Extending (Black-box) SMC with Process Mining

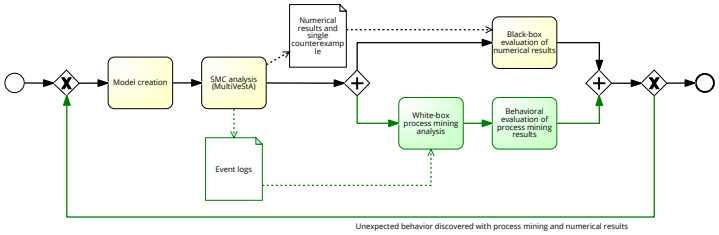
In this section we discuss a novel methodology for white-box behavioral explanation of SMC analysis results by process mining.

3.1 Methodology

Fig. 3a graphically depicts the typical methodology adopted for the construction and validation of simulation models based on SMC analysis. It starts with the creation of the model, which is then analyzed producing numerical results, plots,



(a) State-of-the-art life-cycle of SMC-analysed simulation models.



(b) Novel SMC- and PM-guided methodology for white-box behavioral model validation. Additions wrt Fig. 3a are shown in green.

Fig. 3: SMC- and PM-based ethodology for white-box behavioral validation.

and possibly counterexamples for the studied properties. At this point, if the modeler spots unexpected numerical results, e.g., the probability of success of an attack is lower than expected, they might make an informed guess to hypothesize changes to be applied to the model to improve its performance and *fix* it. This can be seen as an *SMC-guided black-box validation* of the model (we use the term black-box, because there is no overview of the overarching behavior, but just numerical values or single counterexamples).

Fig. 3b depicts a new methodology we propose which enriches (black-box) SMC with *SMC- and PM-guided white-box behavioral model validation*. The main idea is to augment SMC, e.g., MultiVeStA, with the ability to generate logs for the computed simulations, corresponding to one *trace* per simulation (here we use the term *trace* to indicate one sequence of observed events, all belonging to the same process instance). Once the analysis is concluded, we feed these logs to existing PM tools, enabling a behavioral interpretation of the analysis results. Indeed, the output of PM tools can be visually inspected to evaluate the behaviors originating from the collected instances of the attacker behavior (the simulations). Therefore, the result is not anymore given as raw numbers, which require an important cognitive step for *debugging* the model and discovering if and where something went wrong. Rather, PM provides interactive and navigable (via abstraction and filtering of traces and connections) graphical results which are closer and more comparable to the original model. This better supports and guides the identification of flaws or enhancement opportunities.

3.2 Operationalizing the Methodology

In order to enrich MultiVeStA with PM-oriented logging capabilities we have extended its interface with simulators, and in particular the RisQFLan's, with

two methods, namely `createLogFile` (invoked once per analysis by MultiVeStA to create an empty log file); and `addLogRow` (invoked by MultiVeStA every time we want to note down an *event of interest*). In principle, MultiVeStA could be instructed in invoking `addLogRow` once or several times per simulation step, or only when specific events arise. This would give freedom to the modeler in deciding the coarseness-degree of the logs. Currently, we invoke the method `addLogRow` once per simulation step, recording:

- The incremental counter of steps (the *time stamp*);
- The unique random seed used by the current simulation (the *case ID*);
- The executed action (the *activity*);
- The target state of the executed transition (also part of the *activity*);
- Relevant additional information, e.g., the achievement of (sub)-attack attempts, the activation status of countermeasures, and further information on the obtained simulation state.

The set of information recorded are then structured in a file that can be used for the process mining analyses. Indeed, in order to analyse a log with process mining tools, it is necessary to have: *(i)* a case identifier (i.e., case ID) which is used to group together all the observations belonging to the same process instance (i.e., the simulation number); *(ii)* an activity name, indicating which action has been executed in each event; *(iii)* a timestamp (used for sorting the events and producing a sequence). Additional attributes can also be used in order to refine the analyses, for example, by applying some filters to focus only on successful attacks or to remove executions involving some paths/actions. For the actual process mining analyses we used Fluxicon Disco (`https://fluxicon.com/disco/`) which allows importing CSV files and provides very useful tools for interactive navigation and exploration of the results.

4 Experiments

Here we apply our proposed methodology to the discussed case study. In our experiments, we use Fluxicon Disco to study the generated event logs of the attacks. We investigate how the integration of SMC (MultiVeStA) and PM (Disco) allows us to create two model refinements improving the *performances* of the attacker, and fixing *mistakes* in the model. We will use SMC to analyze the original model and the two devised refinements with respect to the following property:

> *What is the probability for the attacker to succeed in a robbing the bank within 60 steps of simulation?*

The property is purposely simple. We chose 60 steps because we found it sufficient to cover the dynamics of the model. The results obtained by for the three variants are given in Fig. 4. In all the three variants, we used $\alpha = 0.1$, $\delta = 0.1$, while MultiVeStA decided to run 240 simulations.

Model	*Prob.*
Original	0.17
1^{st} refinement	0.31
2^{nd} refinement	0.72

Fig. 4: Probability of bank robbery.

We start discussing how the bank robbery case study can be encoded in RisQFLan.

```
begin attributes
    Cost = {BlowUp = 90, OpenVault = 0}
end attributes

begin quantitative constraints
    { value(Cost) <= 100 }
end quantitative constraints
```

```
begin attack detection rates
    BlowUp = 0.0
end attack detection rates
```

Fig. 5: RisQFLan: (Left) Nodes' attributes and quantitative constraints. (Right) Detection rates of attack nodes.

4.1 RisQFLan Encoding of the Running example

Fig. 1 and 2 were generated by RisQFLan starting from a textual description which we detail here. The RisQFLan code for the tree-structure of the ADT itself from Fig. 1 is straightforward, therefore we do not discuss it here. The nodes in Fig. 1 contain quantitative information like `Cost`, `Detection rate`, and `Defense effectiveness`. These are given in *RisQFLan code blocks* shown in Fig. 5. The `attributes` block specifies quantitative attributes of nodes. In the example we defined only `Cost`, which specifies that the cost of `BlowUp` and `OpenVault` attempts is 90 and 0, resp. This means that every time `succ(BlowUp)` or `fail(BlowUp)` are executed, a counter `Cost` is increased by 90. This has a strong impact on the allowed simulations. Indeed, the block `quantitative constraints` imposes that, *no matter what*, the value of such counter will never be allowed to be greater than 100. Practically, this imposes that the attacker will only be allowed to attempt at most one `BlowUp` attempt per simulation. This is obtained as follows: at every simulation step, the weight (and therefore the probability of execution) of actions violating constraints is scaled down to 0. Block `attack detection rates` specifies that `BlowUp` attempts have probability 0 of being detected. This de facto imposes that the countermeasure `LockDown` will never be activated. We omit discussing `Defense effectiveness` because irrelevant to this paper.

We now move our attention to the probabilistic attacker behavior in Fig. 2, given by the block in Fig. 6. We note an almost one-to-one correspondence among Fig. 2 and Fig. 6. Looking at the first transition, we see how those are given by a source (`Start`) and target state (`Complete`), the executed action (`succ(RobBank)`), the weight (3), and an optional *transition constraint* (a guard) that blocks the execution of the transition if not satisfied. In this case, `allowed(RobBank)` imposes that we can attempt `RobBank` attacks only if allowed by the constraints from the ADT. This models a sort of *smart* attacker that does not waste resources in attempting the attack until s/he knows to have chances of success. Indeed, the semantics of RisQFLan imposes that, without such guard, only `fail(RobBank)` (and not `succ(RobBank)`) would be allowed to execute if the the constraints coming from the ADT are not satisfied [4]. Finally, in the figure we also see that the modeler can add model-specific actions on top of the `succ` and `fail` ones.

4.2 Analysis and Refinement of the Original Model

Behavioral analysis of results. As shown in Fig. 4, in the original model the attacker has only 0.17 probability in succeeding in its attack. By feeding the

```
begin attacker behavior
  begin attack
    attacker = Thief
    states = Start, TryOpenVault, TryBlowUp, Complete
    transitions =
      // If vault open or blown up,  attacker can rob
      Start - (succ(RobBank), 3, allowed(RobBank)) -> Complete,
      Start - (fail(RobBank), 1, allowed(RobBank)) -> Complete,
      // Strategy where the attacker can open the vault
      Start -(chooseOV, 2) -> TryOpenVault,
        TryOpenVault -(succ(OpenVault), 2) -> Start,
        TryOpenVault -(fail(OpenVault), 1) -> Start,
      // Strategy where the attacker tries to blow up the vault
      Start -(chooseBU, 4) -> TryBlowUp,
        TryBlowUp -(succ(BlowUp), 2) -> Start,
        TryBlowUp -(fail(BlowUp), 10000) -> Start
  end attack
end attacker behavior
```

```
begin actions
    chooseOV
    chooseBU
end actions
```

Fig. 6: RisQFLan: Probabilistic attacker behavior.

obtained logs into Disco (with filters set to 100% for both the activities and the path), we obtain the process in Fig. 7. We can see that the starting state is `Start-reset`, where `Start` is the initial state of Fig. 2, while `reset` is a special action implicitly execute by MultiVeStA when resetting the simulator before performing a new simulation. For example, we have a transition from the root state to `TryBlowUp-chooseBU`, labeled 167, to denote that 167 times out of 240 the attacker attempts a `BlowUp` attack. This is about twice the number of times (73) that `OpenVault` is chosen as first attack, coherently with the weights of the transitions in Fig. 6 with action `chooseOV` (2), and `cooseBU`, respectively.

Let us look at the process state `Complete-succ(RobBank)` at the bottom-left of Fig. 7. No other process state contains `succ(RobBank)`. Therefore, all and only the simulations reaching this configuration represent executions in which the attacker succeeded in robbing. The label 41 in the process state denotes that this happens 41 times. Indeed, $^{41}/_{240} = 0.17$, the probability given in Fig. 4. At the bottom-left of Fig. 7 we also find state `Start-succ(OpenVault)`. Once we get into such state, we have succeeded in attacking `OpenVault` (the attacker successfully completed the left-most strategy in Fig. 2, and went back in state `Start`). According to the constraints expressed in the ADT in Fig. 1, the attacker could now attempt `RobBank` attacks. This is because the ADT requires that at least one among `OpenVault` or `BlowUp` is necessary.

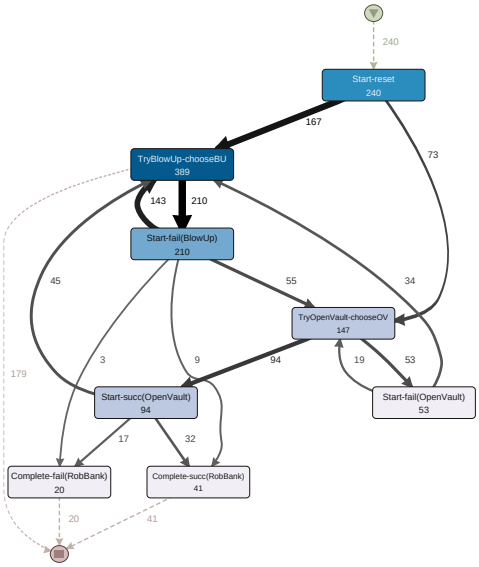


Fig. 7: Process model generated by Disco for the original model.

Indeed, the two downward process transitions outgoing from procees state

```
// We add the !allowed(RobBank)
Start -(chooseBU, 4, !allowed(RobBank)) -> TryBlowUp,
TryBlowUp -(succ(BlowUp),     2) -> Start,
TryBlowUp -(fail(BlowUp), 10000) -> Start
```

Fig. 8: First model refinement: we fix the *non-parsimonious attacker* problem.

`Start-succ(OpenVault)` attempt the root attack, failing 17 times, and succeeding 32 times. Instead, the upward transition entering in `TryBlowUp-chooseBU` depicts that in many cases, 45, the attacker decided to first attempt also a `BlowUp` attack. This is permitted by the model and the ADT. However, it might be somehow unexpected or irrational by the attacker. Clearly, this will lead to higher expenses (`BlowUp` attempts have high cost), and lower success rate (due to `fail` actions possibly execute).

Model refinement. Leveraging the analysis in Disco we discovered that the original model erroneously, or at least unexpectedly, describes a sort of *non parsimonious* attacker that attempts non-necessary `BlowUp` attacks even if they already succeeded in `OpenVault` ones. We can solve this issue by changing the transition with action `chooseBU` from Fig. 6 as shown in Fig. 8. The fix is simple: we only need to add a guard `!allowed(RobBank)`. As we shall see in the next section, this guarantees that the attacker will not attempt `BlowUp` attacks if they already met the ADT requirements to attempt the root goal.

4.3 Analysis of the First Refinement

Behavioral analysis of results. As shown in Fig. 4, in the first refinement, the probability of success of the attacker increased to 0.31. By feeding the obtained logs into Disco, we obtain the process in Fig. 9 (left). Here, we can see that we have solved the issue from Section 4.2: from state `Start-succ(OpenVault)` we only have the two downward transitions attempting `RobBank` attacks.

From the discovered process, we can see a further issue in the model. We can see that 3 process states have a grey dashed transition towards the bottom-most small grey circle with an inscribed square. These are three *endpoints*, meaning that simulations terminated in such states. The two bottom endpoints related to `Complete` are expected, because Fig. 6 dictates that we do not have outgoing transitions from state `Complete`. Instead, the third endpoint was not expected because `TryBlowUp` was not expected to be a terminal state, and therefore it can be considered as a *deadlock problem*. Notably, as shown in the label of the corresponding dashed transition, this issue has a particularly strong impact, as 138 simulations out of 240 terminate getting stuck in state `TryBlowUp`.

Using Disco, it is easy to *zoom in* specific behavior. Fig. 9 (right) shows a process generated by Disco by focusing only on the 138 simulations getting stuck in `TryBlowUp`. We can see that the endpoint state is visited 276 times, twice per simulation. This clearly highlights a conflict among the cost of attempting `BlowUp` attacks (90), and the constraint on maximum allowed cost (100), both shown in Fig. 5 (left). What happens is that the second time the attacker gets in state `TryBlowUp`, they have spent already 90. Looking at Fig. 6, we can see that the two transitions outgoing from this state have action `succ/fail(BlowUp)`, modeling

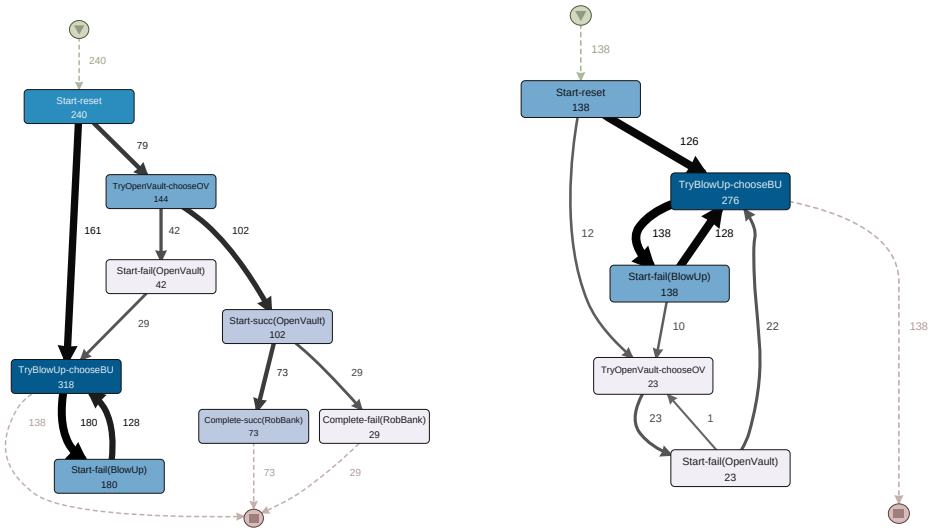


Fig. 9: Behavioral analysis of first refinement. (Left) PM process. (Right) Filtered process for end-point `TryBlowUp`.

```
// Strategy where the attacker tries to blow up the vault
Start -(chooseBU, 4, !allowed(RobBank)) -> TryBlowUp,
TryBlowUp -(succ(BlowUp),     2) -> Start,
TryBlowUp -(fail(BlowUp), 10000) -> Start,
TryBlowUp -(goBack, 0.00001) -> Start
```

```
begin actions
    chooseOV
    chooseBU
    goBack
end actions
```

Fig. 10: Second model refinement: we fix the *deadlock* `TryBlowUp`.

the success or failure of this attack, and target state `Start`. Unfortunately, both actions would lead to a cost of 180, violating the constraint of maximum cost. In other words, both transitions or disabled on second visit of `TryBlowUp`, making it a terminal state.

Model refinement. This behavior was unintended. We would like the attacker to be able to always get back to state `Start` to attempt further attack strategies without remaining stuck in state `TryBlowUp`.

This can be easily solved by adding a third outgoing transition from state `TryBlowUp` as shown in Fig. 10. Here, with a very low weight, we can take a transition with new action `goBack` to go back to state `Start` without attempting any attack. Being the weight so small, this transition will almost always be selected only in the cases that were creating a deadlock in the previous variant. As we shall see in the next section, this guarantees that we find only the two expected `complete`-related endpoints.

4.4 Analysis of the Second Refinement

Behavioral analysis of results. As shown in Fig. 4, in the second refinement,

the success probability of the attacker increases considerably from to 0.31 to 0.72. This is coherent with the fact that we expect to have improved on the 138 simulations that were getting stuck in state `TryBlowUp`. By feeding the obtained logs into Disco, we obtain the process in Fig. 11. The process generated by Disco confirms that we now have only the two expected endpoints. From the process we see we get in process state `Complete-succ(RobBank)` 173 times, with $^{173}/_{240} = 0.72$ as indicated in Fig. 4.

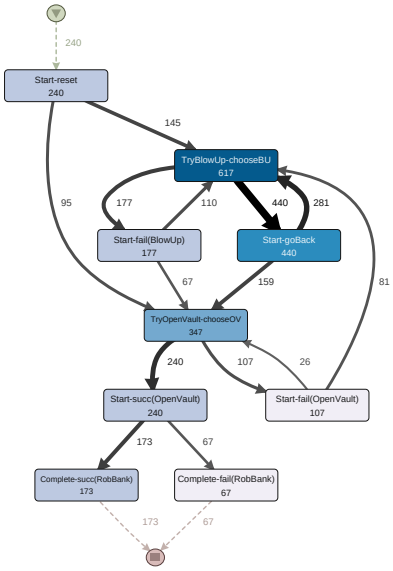


Fig. 11: PM behavior of 2^{nd} refinement

Finally, we note that we never enter in a process state involving `succBlowUp`. This is expected from the very high weight (10K) of `succBlowUp`. This was on purpose, to allow for an easier presentation of the results instrumental for presenting the methodology.

5 Conclusions and Future Work

We proposed a novel methodology for validating and enhancing simulation models to make them more reliable. The methodology is based on the integration of simulation-based analysis techniques known as statistical model checking (SMC), with process-oriented data-driven techniques known and process mining (PM). A simulation in SMC corresponds to a trace in PM. To the best of our knowledge, this is the first integration of SMC and PM. We obtained a novel methodology for *SMC- and PM-guided white-box behavioral model validation and enhancement.*

We demonstrated our methodology on a toy example from the threat modeling domain. We have seen how PM can *discover* issues in the model by inspecting the simulations generated by SMC. Notably, given that SMC is able to decide the *correct* number of simulations necessary to perform an analysis, in some sense we also obtain statistically-reliable event logs for being used with PM.

As future work, we will consider more realistic models, from several domains. E.g., agent-based models from the social sciences, given that the SMC tool MultiVeStA has been recently redesigned and extended to tailor them [4]. Additionally, conformance checking techniques might become relevant in order to ensure that the simulations produced by SMC fulfill normative models. We also foresee a richer integration of PM and SMC. Currently, we use PM *after* SMC completion. We might consider scenarios where streaming process mining techniques are performed *during* SMC in order to tailor the SMC analysis. In addition, PM might also be used *before* SMC. E.g., discovery algorithms might be applied to real data to synthesize attack-defense trees and/or attacker behaviors.

References

1. van der Aalst, W.M.: Process Mining. Springer, 2nd edn. (2016)
2. Agha, G., Palmskog, K.: A survey of statistical model checking. ACM Trans. Model. Comp. Simul. **28**(1), 6:1–6:39 (2018)
3. Aslanyan, Z., Nielson, F., Parker, D.: Quantitative Verification and Synthesis of Attack-Defence Scenarios. In: Proc. CSF'16. pp. 105–119. IEEE (2016)
4. ter Beek, M.H., Legay, A., Lafuente, A.L., Vandin, A.: Quantitative security risk modeling and analysis with RisQFLan. Computers & Security **109**, 102381 (2021)
5. ter Beek, M.H., Legay, A., Lluch-Lafuente, A., Vandin, A.: A Framework for Quantitative Modeling and Analysis of Highly (Re)configurable Systems. IEEE Trans. Software Eng. **46**(3), 321–345 (2020)
6. Belzner, L., De Nicola, R., Vandin, A., Wirsing, M.: Reasoning (on) service component ensembles in rewriting logic. In: Spec., Alg., and Soft. pp. 188–211 (2014)
7. Bulychev, P.E., David, A., Larsen, K.G., Mikucionis, M., Poulsen, D.B., Legay, A., Wang, Z.: UPPAAL-SMC: statistical model checking for priced timed automata. In: Proc. QAPL'12. vol. 85, pp. 1–16 (2012)
8. Ciancia, V., Latella, D., Massink, M., Paškauskas, R., Vandin, A.: A tool-chain for statistical spatio-temporal model checking of bike sharing systems. In: ISOLA'17
9. Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F., Vandin, A.: A formal approach for the analysis of BPMN collaboration models. JSS **180**, 111007 (2021)
10. Gilmore, S., Reijsbergen, D., Vandin, A.: Transient and steady-state statistical analysis for discrete event simulators. In: IFM. pp. 145–160. Springer (2017)
11. Gilmore, S., Tribastone, M., Vandin, A.: An analysis pathway for the quantitative evaluation of public transport systems. In: IFM (2014)
12. Hu, J., Niu, H., Carrasco, J., Lennox, B., Arvin, F.: Fault-tolerant cooperative navigation of networked uav swarms for forest fire monitoring. Aerospace Science and Technology **123**, 107494 (2022)
13. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Foundations of Attack-Defense Trees. In: Proc. FAST'10 (2011)
14. Kumar, R., Ruijters, E., Stoelinga, M.: Quantitative Attack Tree Analysis via Priced Timed Automata. In: Proc. of FORMATS. pp. 156–171. Springer (2015)
15. Law, A.M.: Simulation Modeling and Analysis. McGraw-Hill, 5th edn. (2015)
16. Mauw, S., Oostdijk, M.: Foundations of Attack Trees. In: ICISC (2005)
17. Pianini, D., Sebastio, S., Vandin, A.: Distributed statistical analysis of complex systems modeled through a chemical metaphor. In: HPCS. pp. 416–423 (2014)
18. Research and Technology Organisation of NATO: Improving Common Security Risk Analysis report. RTO Technical Report TR-IST-049 (2008)
19. Schneier, B.: Attack trees. Dr. Dobb's Journal (1999), `bit.ly/3tcfuoZ`
20. Sebastio, S., Vandin, A.: MultiVeStA: statistical model checking for discrete event simulators. In: 7th Int. Conf ValueTools'13. pp. 310–315. ICST/ACM (2013)
21. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: CAV 2004. pp. 202–215. Springer (2004)
22. U.S. Department of Defense: Defense Acquisition Guidebook, Section 8.5.3.3 (2009), `https://bit.ly/3NJjs07`
23. Vandin, A., ter Beek, M.H., Legay, A., Lluch-Lafuente, A.: QFLan: A Tool for the Quantitative Analysis of Highly Reconfigurable Systems. In: FM (2018)
24. Vandin, A., Giachini, D., Lamperti, F., Chiaromonte, F.: Automated and distributed statistical analysis of economic agent-based models. Journal of Economic Dynamics and Control p. 104458 (2022)
25. Younes, H.L.: Probabilistic verification for "black-box" systems. In: CAV 2015. pp. 253–265. Springer (2005)