©2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

The published version of this paper can be found at https://doi.org/10.1109/RAM.2017.7889759.

Uniform Analysis of Fault Trees Through Model Transformations

Enno Ruijters, University of Twente
Stefano Schivo, University of Twente
Mariëlle Stoelinga, University of Twente
Arend Rensink, University of Twente

Key Words: attack-fault trees, fault tree analysis, meta-modelling

SUMMARY & CONCLUSIONS

As the critical systems we rely on every day, such as nuclear power plants and airplanes, become ever more complex, the need to rigorously verify the safety and dependability of these systems is becoming very clear. Furthermore, deliberate attacks have become a prominent cause of concern for safety and reliability.

One of the most prominent techniques for analyzing such systems is fault tree analysis (FTA), and a whole forest of variants, extensions, and analysis tools have been developed. In the security field, FTA was the inspiration for attack trees, used to analyze systems for vulnerability to malicious attacks. These formalisms are rarely compatible, making it difficult to exploit their different strengths in analyzing the same system.

The key contribution of this paper is a meta-model describing many varieties of fault and attack trees, and well as combined attack-fault trees. We provide translations to and from different formalisms, as well as our own analysis engine for combined models. We demonstrate this framework on three case studies.

1 INTRODUCTION

Modern society increasingly depends on complex systems such as power plants and sophisticated medical equipment. As our dependence on such systems grows, it is essential that their dependability grows with it. Furthermore, aside from traditional dependability concerns about accidental failures, critical systems are increasingly threatened by malicious actors attempting to disrupt them.

Aside from qualitative techniques like FMEA analysis [1], one of the most prominent techniques for safety analysis in industry is fault tree analysis (FTA) [2]. Fault trees (FTs) describe the various failures that can occur in a system and how these failures combine to cause system failures. Given information about the probabilities of failures, quantitative analysis can be performed to obtain dependability metrics such as reliability.

While the basics of FTA are well-understood, a large variety of tools and extensions have been developed providing different analyses and new ways of modeling. Unfortunately, most of these tools are hard to combine and support only one formalism. The need for a framework combining them is therefore clear.

In the field of security analysis, the formalism of attack trees (AT) has been developed. This formalism is similar to FTA in that it describes the various attacks that could be performed and which combinations of attacks lead to a successful system attack. Also similar to FTA, a wide variety of tools and extensions exist with very little interoperability.

This paper describes a meta-model for attack trees and fault trees, uniting several tools and extensions, suitable for easy extension to include new formalisms. The meta-model supports combinations of different extensions in one tree, and even allows attacks and faults to be included in the same model. Through model transformations, our framework allows models to be converted from one formalism to another. We also provide a conversion to our own analysis engine using the UPPAAL [3] tool to perform analyses of trees with feature combinations not supported by existing tools. We demonstrate this framework on three case studies, namely one fault tree, one attack tree, and one combined attack-fault tree.

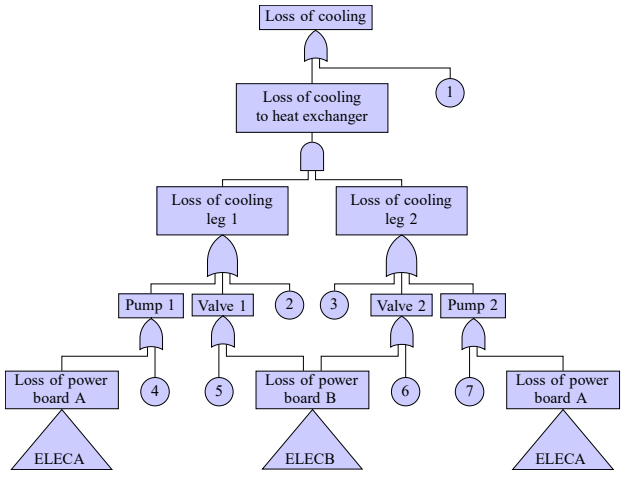

Figure 1: Part of the fault tree of case study 1. Omitted parts of the tree have been replaced by triangles.

2 PRELIMINARIES

2.1 Fault trees

Fault trees are an industry-standard tool for conducting safety and reliability analysis of critical systems. Fault trees (FTs) describe the propagation of failures from individual components to system-wide failures.

The components of a system are described by *leaves* in a fault tree, and are decorated with probability distributions describing their failure behavior over time. These failures combine in *gates*, which describe what combinations of subsystem failures cause an intermediate event.

An example of an FT is shown in Figure 1, taken from an example of the commercial tool Isograph FaultTree+ [4]. At the top of the tree is the *top level event (TLE)*, in this case loss of cooling. Immediately below that is an OR-gate, indicating that if either component 1 fails, or the intermediate event "loss of cooling to heat exchanger" occurs, the TLE occurs. Below the loss of cooling to heat exchanger is an AND-gate, indicating that both subsystems need to fail for this event to occur. As a final note, the subtrees ELECA and ELECB cause failures of multiple subtrees.

2.2 Attack trees

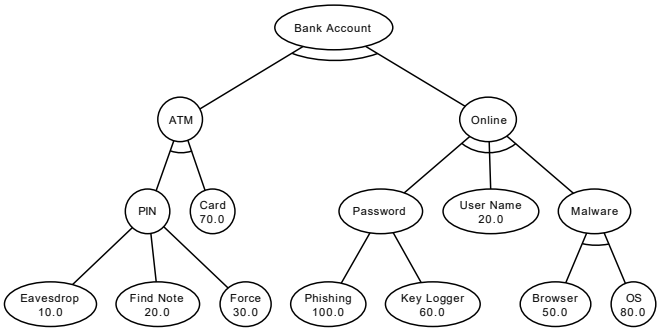

Figure 2: An example of attack tree. An attacker attempts to obtain access to somebody's bank account.

Fault Trees have spawned many different variants, and have been used as an inspiration for other research fields as well. One modeling formalism based on FTs is Attack Trees (ATs), which is applied in the field of security. While maintaining the same general structure as FTs, ATs describe how a system can be attacked. In particular, the leaves (called attack steps) represent an attacker's possible actions, intermediate nodes (gates) define different possible attack paths, and the root of the tree is the target of the attack. In Figure 2 we show a simple AT (taken from ADTool [5]) where an attacker tries to gain access to somebody's bank account. AND and OR gates are used in the example, where AND gates are marked with an arc running along the edges connecting the gate with its children. So we see in Figure 2 that in order to activate the "ATM" node, an attacker needs to activate both "PIN" and "Card". "PIN" in turn requires the activation of at least one out of the three attack steps "Eavesdrop", "Find note", and "Force".

The numbers in the leaves were arbitrarily chosen to represent minimum time durations (in hours) for the corresponding attack steps.

2.3 Timed automata

In this paper we present a tool for converting between different formalisms of fault and attack trees. In addition to translating to and from existing tools, our tool provides its own analysis engine based on priced timed automata (PTA) [6], which extend timed automata (TA) with costs on locations and transitions, and stochastic timed automata, which extend TA with probability distributions on transitions.

A TA is a model consisting of locations and transitions between these locations, with constraints describing when certain transitions may or must be taken. Clocks keep track of the passage of time, and are used in constraints specify when a transition may be taken or to limit how long a location may be occupied.

At any time, one of the locations is the current location. Depending on the constraints, a transition may be made to move to a new current location, or the TA can remain in the same location but increase the value of the clocks. When a transition is made, a specified subset of clocks is reset to start counting time over from 0.

In standard TAs, transitions are either guaranteed or prohibited to be taken at any given time, or there is a nondeterministic choice to take a transition. Stochastic timed automata (STA) [11] have the additional option of attaching a probability distribution to the time when a transition is taken. This enables one to compute properties such as the probability of reaching a certain state before a given time.

PTAs extend TAs by allowing costs to be specified for being in a location (per time unit), and for taking a transition. This allows analysis engines such as UPPAAL CORA [6] to compute cost-optimal policies to reach certain objectives.

These formalisms allow multiple TAs to be combined into one network, using *signals* to communicate. An automaton can emit an output signal (denoted by an exclamation mark, e.g. 'a!') when taking some transition. Automata that have the

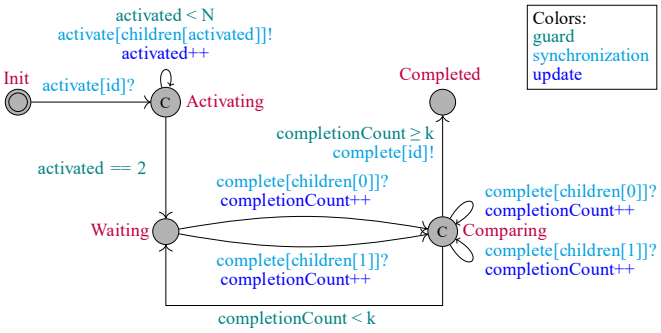

Figure 3: Timed automaton for a gate with N=2 children, failing when k children fail. The automaton begins in the Init state waiting for an activation signal. Upon receiving this signal, it activates all its children and waits for their completions. Whenever a child sends a completion signal, the TA examines whether enough children have completed (e.g. 1 for an OR-gate, or N for an AND-gate), and if so, sends its own completion signal. 'C's mark committed locations.

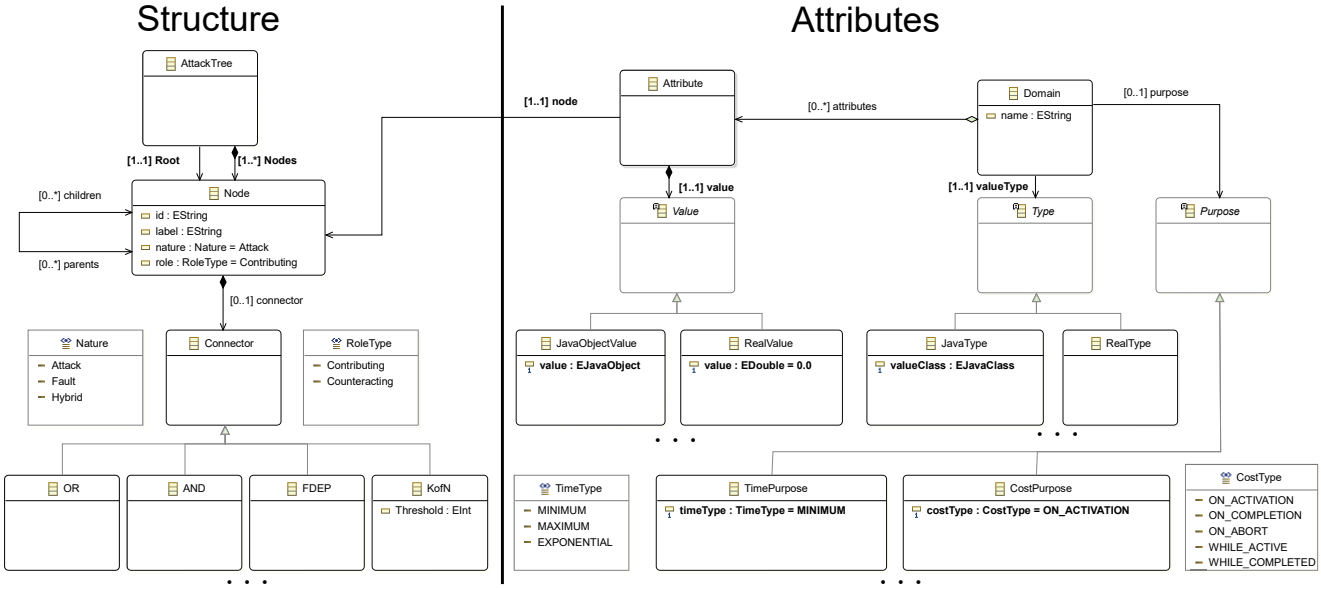

Figure 4: The unified AFT meta-model. The "..." indicate parts that can be further extended.

option to take a transition marked with an input signal (denoted by a question mark, e.g. 'a?') do so at exactly the same time that the corresponding output signal is emitted.

An example of a TA is shown in Figure 3, modelling the behavior of a fault tree gate. This TA communicates with its children and parents using the 'activate' and 'complete' signals.

2.4 Model-driven engineering

Many fields such as mathematics, physics, and biology use models to describe behavior and abstract away irrelevant details. In computer science, models have long been used as documentation, but model-driven engineering is an approach that uses models as a basic abstraction in software design as well.

In model-driven engineering, models are uniformly represented as instances of meta-models. These meta-models are in essence models of models, defining the structure and behavior that instances of the meta-model must adhere to [7]. Meta-models are similar to types in other programming languages (i.e. integers, strings, etc.) and allow systematic definitions of the creation, use, and transformation of models.

Due to the compact representation of model operations such as transformations, model-driven engineering allows complex operations to be concisely expressed. Furthermore, by describing the 'essence' of a model, instances of such models in different formats such as XML, JSON, and binary formats can be used together for easy interoperability.

3 METHODOLOGY

3.1 Attack-fault tree meta-model

The meta-model we built includes different versions of FT and AT, allowing easy translation of models between different formats to exploit the strong points of different analysis tools. Thanks to its encompassing nature, the meta-model can also represent so-called attack-fault trees (AFT), where both accidental and deliberate types of failures and attacks are represented. This allows one model to take into account the complex interactions occurring between maintenance needs and security requirements. In practice, an AFT could model the (possibly negative) impact on system reliability or safety of a new security measure, or help highlight the security risks inherent an unwary maintenance protocol.

As previously described, using a meta-model allows us to encompass different modeling formalisms and enables translation between them. For example, an AT model could be generated with ADTool and then analyzed with Attack Tree Evaluator [8]. We also provide a translation into a timed automata model we designed to fit different kinds of FTs and ATs. This timed automata model is then analyzed with UPPAAL [3] or its extensions for various properties such as reliability (for FT) and fastest attack vector (for AT). For examples of the analysis workflows enabled by our method, see the Section 4.

The meta-model we schematically present in Figure 4 is divided into two main parts, which represent the tree structure and the attributes associated with the leaves respectively. In this way, we exploit compositionality to analyze the same model with different parameter sets; this allows us to e.g. evaluate the effects of different maintenance strategies (for FT) or attacker profiles (for AT). Thanks to the extensibility of meta-models, additional features and model types can be introduced in a straightforward way.

The structure of the meta-model is made to be a generic representation of different types of FT and AT. We describe a tree as a collection of nodes connected by parent-child relations; a single (root) node has no parent. Nodes can be labelled either as "contributing" or "counteracting"; in FT terms, they would represent faults or repairs, respectively. In an AT, where the

point of view is often that of an attacker, a contributing node represents an attack and a counteracting node is a defensive countermeasure. A node can represent a gate (sometimes also called connector) and its type defines how the completion of the node's children determines the completion of the node itself. The meta-model already includes a number of well-known gates, both from the FT and AT worlds, such as AND, OR, sequential AND (SAND), functional dependency (FDEP), voting (k-of-N), etc.

The nature of a leaf determines whether that leaf represents a component failure or an attack step. The nature of a gate is automatically inferred from the nature of its children, with gates connecting nodes of different nature being assigned a "mixed" nature. A tree whose root has a mixed nature is an AFT.

Attributes can be assigned to the tree leaves to describe the characteristics of the corresponding basic events/attack steps. Each attribute is assigned to a domain, which describes both the data type used to represent its values and the purpose intended for that domain. For example, a domain with real-typed values may be used with the purpose of representing the rate of failure of a component. The same real-typed values can be used in another domain with the purpose of representing the minimum time an attack step takes to be performed. Different leaves can have attributes from different domains, although some restrictions are implied by the semantics (e.g. a leaf with two domains specifying different failure times may cause problems during analysis).

The domains featured in a tree determine which analysis types are available. Because different modeling formalisms use different attributes, the translation of an instance of the meta-model into a modeling formalism will need to include a subset of all the domains it refers, excluding the rest. For example, the Galileo format has no notion of minimum time to occurrence, so in order to analyze a model with DFTCalc such values must be converted to mean times.

3.2 Model transformations

Our tool provides bidirectional transformations between the unified meta-model and several formalisms, including an extended Galileo format also used by DFTCalc [9], the format of the ADTool program for attack trees, and several others. We also provide a transformation from the unified meta-model to the UPPAAL tool, including the SMC and CORA extensions. The produced UPPAAL models can be used for quantitative analysis to compute key performance indicators such as reliability, availability, minimal-time sequence of attacks to reach the TLE, and expected cost. The overall structure of the transformations is shown in Figure 5.

Since the various formalisms support different feature sets, the transformations leave out information that cannot be represented in a given output. For example, the ADTool program does not support spare gates that can be present in Galileo models. The transformation thus converts such spare gates to AND-gates and emits a warning. Whenever possible, the transformations ensure that the resulting model is at most as reliable as the original, to avoid creating an illusion of safety.

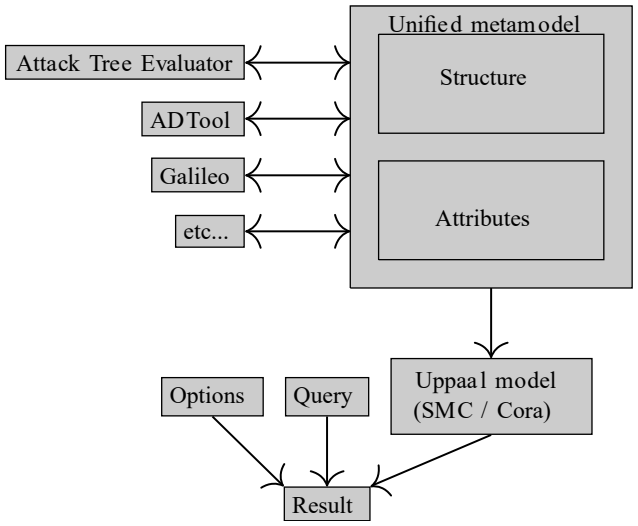

Figure 5: Overview of the transformations our framework supports.

3.3 Analysis tools

In addition to converting between formalisms of existing tools, we provide our own analysis using the UPPAAL tool. We use a translation to timed automata similar to that used in [10].

We translate each element of the AFT (i.e. leaf or gate) into a priced (stochastic) timed automaton, which communicates using signals to combine into a model of the entire tree.

For example, Figure 3 shows the timed automaton for one gate with two children, and Figure 6 shows the stochastic timed automaton for a basic event. During the translation, the IDs of the gate and children are assigned such that each element has a unique number.

After constructing the network of automata for the entire tree, UPPAAL has various extensions that allow different analyses to be performed. For example, the UPPAAL-CORA [6] tool can compute cost-optimal traces to reach a certain state, corresponding to the minimal-cost attack that successfully causes the TLE to occur. The UPPAAL-SMC [11] extension uses statistical model checking to allow statistical properties to be computed, such as the probability that the top event occurs before a certain time (a.k.a. the system unreliability).

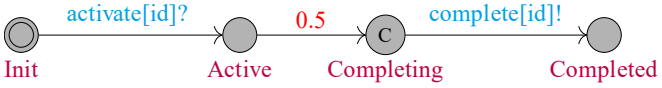

Figure 6: Stochastic timed automaton for a basic event with an exponentially distributed failure time with a failure rate of 0.5.

4 CASE STUDIES

To demonstrate our approach, we have conducted three case studies. The first considers a fault tree and shows that we can reproduce a tree from Isograph FaultTree+ and analyze it correctly. Next, we take an attack tree from ADTool and analyze it to obtain information that the original tool could not compute. Finally, we combine fault trees and attack trees and

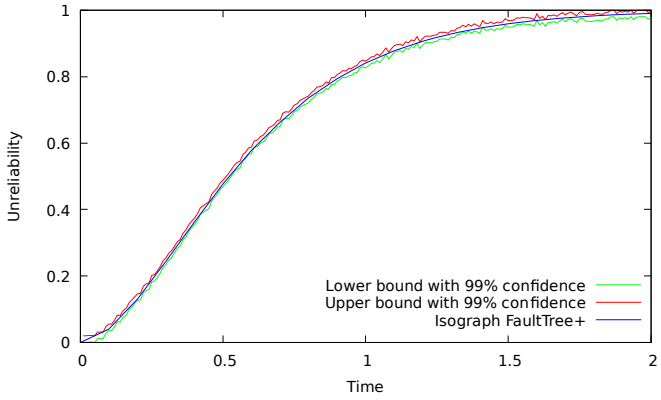

Figure 7: Analysis results for the fault tree case study.

show that we can obtain useful information from such a combined model.

4.1 Fault tree: Cooling

This case study considers one of the examples provided with the Isograph FaultTree+ tool demo: the cooling system. This example models a cooling system with redundant pumps and power supplies. A portion of the FT can be seen in Figure 1. The original FT contained repair rates, which are not currently supported by our tool and have therefore been removed. Other than that, the model was reproduced without modifications.

The model was transformed to two new formalisms for analysis: a Galileo model which was analyzed using DFTCalc [9], and an UPPAAL-SMC model as described in Section 3.3. The UPPAAL-SMC model was then analyzed to obtain the unreliabilities between times 0 and 2, performing enough simulations to achieve a 99% confidence interval with a width of 1%. Similarly, the Galileo and the original FaultTree+ models were analyzed to compute the unreliability over time.

The results of the UPPAAL analysis are shown in Figure 7. The graphs show that the correct unreliability as computed by FaultTree+ lies within the confidence interval computed by UPPAAL. The results computed by DFTCalc were equal to those computed by FaultTree+ to within four decimal places. We thus conclude that the fault tree was correctly translated.

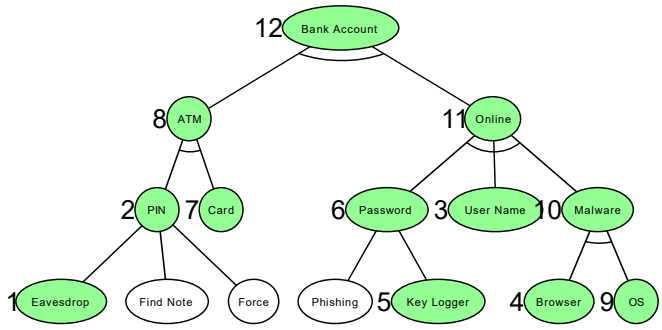

Figure 8: The fastest attack to obtain access to the bank account. Numbers indicate the sequence of steps that need to be completed in a successful attack.

4.2 Attack tree: accessing bank account

In this case study we present the analysis of the example AT described in Figure 2. The model was produced using ADTool, which already provides basic analysis features such as the inference of the minimal time for a successful attack. In order to perform further analyses, we translated the original model into a timed automata model, which was then analyzed with UPPAAL. The translation was made possible by the definitions of the meta-models of:

1. ADTool XML format,
2. Unified AFT as described in Section 3.1,
3. UPPAAL model for timed automata [12],
4. UPPAAL XML format.

The translation workflow proceeds then in order 1 → 2 → 3 → 4. The transformations are specified in the Epsilon framework [13], and our tool automatically chooses the correct transformations based on the provided input and requested outputs. Human-readable results are obtained at the end of the workflow. In this case study, we gave the model as input to our tool and requested the fastest way to perform an attack. The solution we obtained is represented in Figure 8 by highlighting the nodes that are activated in the solution, together with the sequence in which they are completed.

4.3 Attack-fault tree: redundant water pumps

We will now examine a hybrid model which contains both FT basic events and AT attack steps. The model represents a couple of redundant water pumps controlled by valves, both of which are normally running. Under normal circumstances, each pump has a failure rate of 0.01. An attacker has several options to sabotage the installation: she can damage one of the pumps, introducing a new failure cause with a rate of 0.1, and/or she can sabotage one or both valves causing them to fail after exactly 25 time units. We assign a cost of 10 to the sabotage of a pump and 100 to sabotage any valve.

Figure 9 shows the results of the analysis for attackers expending different costs. We observe that, as expected, an attacker with sufficient budget can cause a system failure after 25 time units by sabotaging both valves. We also see that there

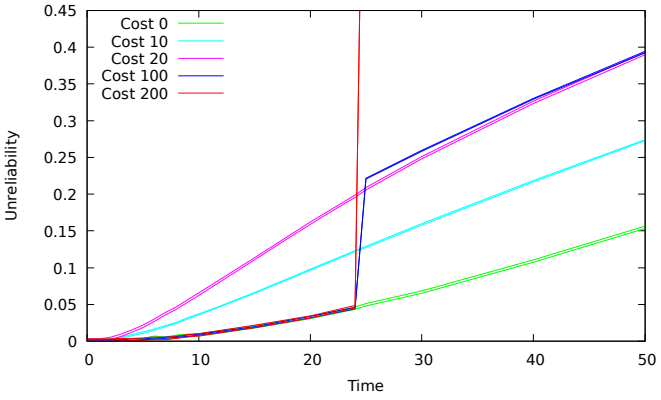

Figure 9: Results of the combined attack-fault tree. The cost=200 line goes to unreliability 1 at time 25. Each pair of curves of the same color describes a 99% confidence interval around the mean.

is very little difference in the unreliability after 25 time units when sabotaging both pumps (cost 20) compared to sabotaging one valve (cost 100), so the pumps will be a much more attractive target for a medium-budget attacker than the valves.

5 CONCLUSION AND FUTURE WORK

This paper presents a meta-model encompassing many different fault tree and attack tree formalisms as well as combinations thereof. Model transformations are used to convert between this meta-model and the existing formalisms, and to a novel analysis framework, based on timed automata, capable of analyzing models combining features of multiple formalisms. Application to case studies demonstrates the correctness and usefulness of the meta-model and transformations.

Future work includes extending the meta-model to cover more formalisms covering topics like repairs and uncertainty.

6 ACKNOWLEDGEMENTS

This work has been supported by the STW-ProRail partnership program ExploRail under the project ArRangeer (12238) and the EU FP7 project TREsPASS (318003).

REFERENCES

1. M. Rausand and A. Hoylan, "*System Reliability Theory. Models, Statistical Methods, and Applications*," Wiley, 2004.
2. W. E. Vesely, F. F. Goldberg, N. H. Roberts, D. F. Haasl, "Fault Tree Handbook," *Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commision*, 1981.
3. K. G. Larsen, P. Pettersson, W. Yi, "UPPAAL in a nutshell," *Int. J. Software Tools for Technology Transfer*, vol. 1(1), pp. 134-152, Dec. 1997.
4. Isograph, FaultTree+: http://www.isograph.com/software /reliability-workbench/fault-tree-analysis/
5. B. Kordy, P. Kordy, S. Mauw, P. Schweitzer, "ADTool: Security Analysis with Attack-Defense Trees," in: *Proc. 10th Int. Conf. Quantitative Evaluation of Systems*, LNCS vol. 8054, 2013, pp. 173-176.
6. G. Berhmann, K. G. Larsen, J. I. Rasmussen, "Optimal scheduling using priced timed automata," *ACM SIGMETRICS Performance Evaluation Review*, vol. 32(4), pp. 34-40, Mar 2005.
7. J. Sprinkle, B. Rumpe, H. Vangheluwe, G. Karsai, "Metamodelling: State of the Art and Research Challenges," in *Model-Based Engineering of Embedded Real-Time Systems*, LNCS vol. 6100, pp. 57-76, 2010.
8. Z. Aslanyan, "Attack Tree Evaluator", Developed for EU project TREsPASS, Technical University of Denmark.
9. F. Arnold, A. Belinfante, F. van der Berg, D. Guck, M. Stoelinga, "DFTCalc: A tool for efficient fault tree analysis," in: *Proc. 32nd Int. Conf. Comput. Safety, Reliability & Security*, LNCS vol. 8153, pp. 293-301, 2013.
10. E. Ruijters and M. I. A. Stoelinga, "Fault maintenance trees: reliability centered maintenance via statistical model checking," in: *Proc. IEEE 62nd Annu. Reliability and Maintainability Symp.*, 2016. doi: 10.1109/RAMS.2016.7447986
11. A. David, K. G. Larsen, A. Legay, M. Mikučionis, D. B. Poulsen, "UPPAAL SMC tutorial," *Int. J. Software Tools for Technology Transfer*, vol. 17(4), pp. 397-415, 2015.
12. C. Gerking, S. Dziwok, C. Heinzemann, W. Schäfer. "Domain-specific Model Checking for Cyber-physical Systems," in: *Proc. 12th Workshop on Model-Driven Eng., Verification and Validation*, Ottawa, Sep. 2015.
13. D. Kolovos, L. Rose, A. García-Domínguez, R. Paige, "The Epsilon Book," [Online]. Available: http://www.eclipse.org/epsilon/doc/book/

BIOGRAPHIES

Enno Ruijters
University of Twente, Formal Methods and Tools
Enschede, Overijssel, 7522 NB, The Netherlands

e-mail: e.j.j.ruijters@utwente.nl

Enno Ruijters is a PhD Student at the University of Twente, currently studying fault tree analysis and stochastic model checking in a railroad infrastructure context. He holds an MSc. in Operations Research from Maastricht University.

Stefano Schivo
University of Twente, Formal Methods and Tools
Enschede, Overijssel, 7522 NB, The Netherlands

e-mail: s.schivo@utwente.nl

Stefano Schivo received the Ph.D. degree in 2010 from the University of Trento, Italy. He is currently a Postdoctoral Researcher at the University of Twente, the Netherlands, where he works on distributed systems of various natures.

Mariëlle Stoelinga
University of Twente, Formal Methods and Tools
Enschede, Overijssel, 7522 NB, The Netherlands

e-mail: marielle@cs.utwente.nl

Dr. Mariëlle Stoelinga is an associate professor at the University of Twente, the Netherlands, leading a team on quantitative analysis and risk management of computer systems. She holds an MSc and PhD degree from Radboud University Nijmegen, the Netherlands.

Arend Rensink
University of Twente, Formal Methods and Tools
Enschede, Overijssel, 7522 NB, The Netherlands

e-mail: arend.rensink@utwente.nl

Prof. dr. Arend Rensink is a professor in software modelling, transformation and verification at the University of Twente, the Netherlands. His research interests are model-driven engineering and graph transformations.