predict
prioritise
prevent

TRESPASS

Technology-supported Risk Estimation by Predictive Assessment of Socio-technical Security

Deliverable D3.5.1

Dynamics of stochastic models

Project:	TRESPASS
Project Number:	ICT-318003
Deliverable:	D3.5.1
Title:	Dynamics of stochastic models
Version:	1.0
Confidentiality:	Public
Editor:	Rajesh Kumar
Cont. Authors:	R. Kumar, J. C. van de Pol, A. Rensink, S. M. Hallberg, C. W. Probst, A. Lenin
Date:	2016-10-31

Part of the Seventh Framework Programme
Funded by the EC-DG CONNECT

Members of the TRESPASS Consortium

1.	University of Twente	UT	The Netherlands
2.	Technical University of Denmark	DTU	Denmark
3.	Cybernetica	CYB	Estonia
4.	GMV Portugal	GMVP	Portugal
5.	GMV Spain	GMVS	Spain
6.	Royal Holloway University of London	RHUL	United Kingdom
7.	itrust consulting	ITR	Luxembourg
8.	Goethe University Frankfurt	GUF	Germany
9.	IBM Research	IBM	Switzerland
10.	Delft University of Technology	TUD	The Netherlands
11.	Hamburg University of Technology	TUHH	Germany
12.	University of Luxembourg	UL	Luxembourg
13.	Aalborg University	AAU	Denmark
14.	Consult Hyperion	CHYP	United Kingdom
15.	BizzDesign	BD	The Netherlands
16.	Deloitte	DELO	The Netherlands
17.	Lust	LUST	The Netherlands

Document History

Authors		
Partner	Name	Chapters
UT	Rajesh Kumar	All
UT	Jaco van de Pol	All
UT	Arend Rensink	2
DTU	Christian W. Probst	3
CYB	Alexandr Lenin	4
TUHH	Sven M. Hallberg	5

Quality assurance		
Role	Name	Date
Editor	Rajesh Kumar	2016-10-31
Reviewer	Lizzie Coles-Kemp	2016-10-31
Reviewer	Sören Bleikertz	2016-10-31
Task leader	Jaco van de Pol	2016-10-31
WP leader	Jaco van de Pol	2016-10-31
Coordinator	Pieter Hartel	2016-10-31

Circulation	
Recipient	Date of submission
Project Partners	2016-10-31
European Commission	2016-10-31

Acknowledgement: *The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318003 (TRE$_S$PASS). This publication reflects only the authors' views and the Union is not liable for any use that may be made of the information contained herein.*

Contents

List of Figures

List of Tables

Management Summary

This deliverable D3.5.1 is the final deliverable of task T3.5, *Dynamics of Stochastic Models* (due M48).

Key takeaways:

- We introduce metamodels for attack trees, to describe the variety of input/output formats of the various WP3 analysis tools, which are due to the rich variation in analysis metrics and analysis algorithms.
- We introduce and demonstrate model transformation, as a means to provide interoperability, not only between the various analysis tools, but also with the models of the surrounding tools in the tool diagram.
- We show how model transformation and unique model identifiers help in propagating changes forward through the analysis models, and tracing analysis results backwards to the original sociotechnical model.
- We extend the dynamics of stochastic models by suggesting an easy-to-use infrastructure of updating the domain-dependent data (like attack patterns and statistical data), and to analyse the impact of uncertain data on the analysis results through sensitivity analysis.
- We show how a compositional modeling framework, combined with automatic model transformation, can gain efficiency by offering incremental generation and analysis of stochastic models, improving the response time of the tools when used during an interactive design exploration session.

1 Introduction

Socio-technical models tend to be complex – both due to the order of their size and due to the causal interactions that characterise their behaviour. A prime objective of the TRE$_S$-PASS modelling language is to provide its users with the flexibility to construct an attack scenario that adequately captures the architectural description of his organization. Hence in TRE$_S$PASS, the *architecture modelling language* (socio-technical model) is intentionally kept flexible with a rich user interface and a separate model pattern library (The TRE$_S$PASS Project, D6.4.3, 2016; The TRE$_S$PASS Project, D4.3.3, 2016; The TRE$_S$PASS Project, D5.3.2, 2015), facilitating reuse of domain knowledge. These attack scenarios are further transformed and represented as attack trees and attack defense trees (defined by an *attack DSL (domain specific language)*.

These attack defense trees are further analysed with different analysis tools developed in the project – e.g. ADTool, ApproxTree+, ATAnalyzer, ATevaluator, ATCalc, and ATtop (The TRE$_S$PASS Project, D3.4.2, 2016). All these tools operate by extracting (explicitly or implicitly) and analysing some stochastic state-transition-system from the attack defense tree. This rich set of analysis tools complement each other, and are designed with the aim to quantify various risk scenarios and risk estimation parameters, while the complexity of the analysis algorithms themselves is kept under the hood. All these tools, however, have different input/output formats. As the analysis algorithms are tightly coupled to the attack scenario, any change or update in the architecture model, or in the attack pattern library, means that a fresh attack tree must be regenerated, on which the analysis tools have to be used. The Deliverable (The TRE$_S$PASS Project, D5.3.3, 2016) elaborates further on the implication of model maintenance, model sharing and model update in the presence of a change either in the threat landscape, or in the evolution of the organization, or in the availability of richer or more up-to-date data sources.

This deliverable reports on Task 3.5.1, focusing on the re-generation of the stochastic models, their re-analysis, and the re-intrepretation of the analysis results after the some of the changes mentioned before happened. The general question is what would be the consequences of a change in some of the inputs on the risk assessment, how such changes in the input are propagated between the various models and tools, how an incremental analysis method can be performed more efficiently, and how the analysis results are propagated back to the models and the user.

Thus, a user is assisted in design exploration, by changing the current scenario, either by editing the architectural model or by incorporating new data, and by rerunning the analysis methods completely automatically, in order to learn the impact of the changes to the original model, be it positive or negative.

1.1 Dynamics in socio-technical models: definitions and terminology

The WP3 analysis tools take as input attack defense trees, for instance those generated automatically from a socio-technical model (architectural model) created in the attack navigator map user interface (The TRESPASS Project, D3.4.1, 2014). The generation consists of two stages: Treemaker generates attack trees at the granularity of the socio-technical model, and the Attack Pattern Library refines them based on standard attack patterns, and adds quantitative data obtained from a data base. Essentially, by viewing the TRESPASS modeling approach as a chain of models, processes and tools, *dynamics* can be embedded at multiple levels. In this deliverable, we report on the current interpretation of the *dynamics* of socio-technical models, addressing those different perspectives, four years after writing the Description of Work:

- First, *dynamics* refers to the glue between different models and supporting tools (data format transformations): In this view, we put Model Transformation as *the* integrating technique to manage change in the presence of model and tool variety. The maturity level of the project, the integration of tools and processes (as visualised in the integration diagram (The TRESPASS Project, D6.4.3, 2016)) and the scientific development of the analysis tools within WP3 over time (each having different Input/ Output formats) required a common knit to tie all the components in one single framework. By defining a generic metamodel (Section 2, based on (Huistra, 2015)) for attack defense trees, and by performing a horizontal transformation between several source and target data models, we have been able to establish coherence between the different analysis tools. This allows to combine and reuse various analysis tools in unforeseen ways, like computing Pareto frontiers by tool X on attack trees edited by the GUI of tool Y.
- Second, *dynamics* is interpreted as a means to propagate changes and to maintain consistency: The attack tree model generated from the socio-technical model through the Treemaker bears unique identifiers. Together with automated model transformation, using these identifiers ensures that all model artefacts stay consistent and reflect the new situation after changes to the architectural model or data bases happened. This is a bidirectional process: unique identifiers and model transformation are also used to trace back elements in the attack DSLs to the socio-technical models (cf. Section 3).
- Third, proper *dynamics* should ensure back propagation of analysis results to the model: The output of the analysis tools is either an attack trace or a nominal value that indicates the impact, execution time or skill level to reach the asset. Vertical model transformation techniques are used to automatically generate the lower level formalism (such as stochastic timed automata (The TRESPASS Project, D3.2.1, 2015)) from the attack DSL, through metamodels of Attack Trees and UPPAAL (Huistra, 2015; Brandt, 2016). Future work involves development of a tool supported reversible transformations, to map the analysis results back to the attack tree model itself. Tracing back a nominal value to either an attack model or the components in

the socio-technical model is a nontrivial task, however. Ultimately, for the user this can best be addressed by visually highlighting the affected components involved in the constructed attack scenarios on the original attack navigator map(The TRESPASS Project, D4.3.3, 2016). In this way, the final analysis results can be interpreted all the way back to the original sociotechnical model.

- Another aspect of proper *dynamics* is to pull in new data smoothly (Section 4). We envision a data pull facility to seamlessly rerun the analysis after pulling in newly discovered attacks and updating recent real-world data from databases representing some application domain.
- Closely related to the dynamics of data is *data sensitivity*: In order to assess the impact of the input parameters on the analysis results, Section 5 reviews several metrics and methods to assess the data sensitivity. For instance, one can perform a parameter sweep (The TRESPASS Project, D3.3.2, 2015). This can be useful to assess which element of the attack tree has the highest contribution to the success probability of a particular attack. Since each element of the attack tree is uniquely linked with the component in the socio-technical model itself, we can thus reason, which socio-technical model component is the most crucial one, and design a counter measure.
- Finally, the modularity of the analysis models enables incremental updates for efficiency: Section 6. The analysis models in TRESPASS are built compositionally (The TRESPASS Project, D3.2.1, 2015). A modular analysis methodology keeps the models comprehensible, so they can be used as a sub-systems in a larger system. Since the analysis models are generated from the socio-technical model description (consisting of attacker profiles, assets and policies and input parameters), we have to regenerate and reanalyse the new attack scenarios derived from the socio-technical model(The TRESPASS Project, D3.4.1, 2014). However, since those models are constructed in a compositional way, after a change only a part of the stochastic model needs to be regenerated and evaluated, potentially improving the response time of the analysis tools during design exploration.

1.2 Foreground and Background

The background of this deliverable is formed by the TRESPASS model building phase (The TRESPASS Project, D6.4.3, 2016), the extraction of attack DSLs using Treemaker (The TRESPASS Project, D3.4.1, 2014), the annotations using APL and the various WP3 analysis tools with all their I/O requirements (The TRESPASS Project, D3.4.2, 2016; The TRESPASS Project, D6.4.4, 2016). The foreground of this deliverable is constituted by all aspects of the integration of these analysis tools, along with their ability to support incremental updates in the socio-technical model, and to link changes forwards and trace results backwards within the TRESPASS tool chain. An important ingredient of this foreground is the work on metamodels of D. Huistra (UT) (Huistra, 2015).

2 Model transformation: interoperability and consistency

Model-driven engineering (Schmidt, 2006; Stahl, Voelter, & Czarnecki, 2006) proposes to use models as prime artefacts in the effective design and implementation of complex enterprise systems, not only to communicate between multiple stakeholders — system designers, developers, security analysts and managers — but also to drive the development process itself. The general idea is to first *model* all aspects of a system, from its static software architecture to its dynamic behaviour and deployment, before proceeding to the level of working *code*.

Models should be structured and documented. To ensure that this is the case, models are usually considered to be elements of a *language* created especially for the domain of discourse; a so-called *domain-specific language*. One of the most popular means to define such a language is through a so-called *metamodel* that captures the concepts of the domain as well as the possible relations between them. In the literature, the notions of metamodel and domain specific language are used more or less interchangeably, even though it is more proper to say that the former is one (and only one) way to define the latter.

2.1 Benefits of Model-Driven Engineering

Model driven engineering has a range of potential benefits, some of which we briefly discuss below.

- *Conceptualisation and standardisation.* By focussing on conceptual aspects rather than implementation-level data structures, one is able to separate the intrinsic concepts of a given domain from their incidental representation. This improves clarity by avoiding the obfuscation of particular choices that may be imposed by the requirements of an executable implementation.

  A well-defined domain-specific language (through a metamodel or otherwise) can also serve as a *standard representation* for the concepts in the domain; models then serve as a means of interchange between humans, as well as tools adhering to that standard.

- *Consistency and maintenance.* Together with the benefits of conceptual modelling comes the opportunity to capture and reason about inter-model consistency. Essentially, every domain concept should be modelled only once and imported from there

into every implementation context; where this is not possible, for instance due to a multitude of implementation platforms, at least by having the different representations of a domain concept available as models, one can properly define their relation and transformations back and forth (see also *model transformation* below).

- *Model transformation.* It is often necessary to transform models from one domain specific language to another or even within the same language, either because implementation details have to be added (sometimes called *vertical* transformation) or because of differences in representation, threatening the consistency mentioned in the previous item (*horizontal* transformation, or *refactoring* if the source and target languages are the same). In TRE$_S$PASS we do both: horizontal model transformations between various tools, and vertical model transformation between the several layers.

  When this is set up properly, the model transformation definitions are themselves artefacts of a domain-specific language and as such subject to analysis, evolution and maintenance. The same benefits discussed for models in general therefore also automatically apply to model transformation definitions. The research areas of model tranformation is very wide, with entire conferences dedicated to the topic; some early categorisations are given in (Czarnecki & Helsen, 2006; Mens & Gorp, 2006).

- *Interoperability.* In practice, in any large software system one typically has to deal with pre-existing tools, whose input and output formats were not designed to fit together. If, as is usually the case, the artefacts being input and output represent domain concepts, then (in the model-driven engineering method) there is bound to be a model representation for them. This, then, can serve as an intermediate format, into which tool output can be transformed (through a process of parsing and model-to-model transformation) and from which tool input can be generated (through a process of model-to-text transformation).

- *Change propagation.* In horizontal transformation, where two representations of the same concept are linked by model transformation definitions, there are many scenarios in which the source or target models are updated after a transformation has already been executed. Such changes should then be propagated back (if it was the target model that was updated) or forth (if it was the source model), while keeping the models consistent. To guarantee such propagation as well as consistency automatically, an area of research is dedicated to *bidirectional* model transformation; see, e.g., (Stevens, 2007).

2.2 Choice of Model Driven Engineering (MDE) technology

The principle of MDE has been picked up re-implemented in several contexts. We especially mention two of them: the OMG context and the Eclipse context.

Object Managament Group (OMG) The Object Management Group has been a strong early proponent of the idea of Model-Driven Engineering, primarily under the title of Model-Driven Architecture (MDA). This has resulted in several influential standards:

- The *Meta-Object Facility* (MOF). This is a format for defining metamodels. In other words, MOF is itself a domain-specific language, the elements (models) of which are metamodels. The specification can be found at (Object Management Group (OMG), 2006).
- The *Unified Modelling Language* (UML). Though called a language, this is actually a suite of domain-specific languages, defined through metamodels. Well-known examples of models captured by one of the UML languages are class diagrams and state diagrams. UML actually started out independently of the OMG, but they have taken over the task of maintainance; the latest version is 2.5, to be found at (Object Management Group (OMG), 2002).
- The *Query-View-Transformation* standard (QVT). This defines a domain-specific language for model transformation definitions, where it is assumed that the source and target languages are both defined through MOF-based metamodels. The latest version can be found at (Object Management Group (OMG), 2015).

A fairly large number of tools have grown up around these standards. In practice, however, it turns out that interoperability is not as good as it should have be given the existence of a common standard. This can be explained by the facts that (1) the standards are themselves written up in natural language and consequently contain a number of ambiguities; (2) there have been a number of versions of the standards, which were not always downward compatible; (3) many tools implement only part of the standard.

Eclipse Modelling Framework (EMF) The Eclipse Modelling Framework[1] (Steinberg, Budinsky, Paternostro, & Merks, 2009) has been an ongoing subproject of the Eclipse Foundation for many years, resulting in (among other things) the definition of an alternative to MOF, called ECore. The situation is analogous to the one described above: ECore serves as a domain-specific language in which one can define metamodels. Around this infrastructure a large number of plugins have sprung up that support modelling and model transformation for ECore-based languages. One important, mature such tool is Epsilon (Kolovos, Rose, García-Domínguez, & Paige, 2016), which is actually a suite of model transformation-related languages and tools.

In comparison to the OMG-based setting, we see the following advantages to ECore + Epsilon: (1) ECore is simpler than MOF and hence leaves less room for ambiguities; (2) ECore is based on a single, widespread tool (Eclipse) rather than a standard implemented, potentially differently, by multiple tools; (3) Ecore and Epsilon have been developed in the academic world, are open source and have a large, dedicated user base. For these reasons, in TRESPASS we have chosen to base our work on ECore (for the metamodels) and Eclipse (for the model transformations).

[1] See https://eclipse.org/modeling/emf/

2.3 Attack Tree metamodel

In TRESPASS, one of the central concepts for the analysis of security risks and countermeasures is that of an *attack tree*, proposed first by Schneier (Schneier, 1999) and later refined and formalised by, e.g., (Mauw & Oostdijk, 2005). In particular, many Attack-Tree-based tools have been developed or extended within the course of the project, such as ADTool, ApproxTree+, ATAnalyzer, ATevaluator, ATCalc, and ATtop, giving rise to the situation painted in Section 2.1, where different tools centered around the same domain concept have not been designed upfront to work together, and so require a post-hoc fix to ensure interoperability.

2.4 Metamodel requirements

For this purpose, in (Huistra, 2015) we have defined an Attack Tree metamodel (in ECore), with the following requirements in mind:

Support for existing formalisms. Kordy et al. in (Kordy, Pièvre-Cambacédès, & Schweitzer, 2014) survey and categorise a large number of attack tree-related formalisms. In (Huistra, 2015) we have investigated different features and settled on a set of such formalisms that we want to cover in the metamodel. In other words, the metamodel should be general enough to encompass all features set down in (Huistra, 2015).

Support for future extensions. The metamodel should be prepared for future extensions. It should be able to evolve and support small extensions without alterations to the metamodel and, to a certain degree, support larger extensions with minimal alterations to the metamodel and (more importantly) without breaking existing model instances of the metamodel.

Support for partial models. The metamodel should aim be expressive and support a large range of (partial) attack trees (e.g. so that it can be used to develop an attack tree in a step- by- step manner). Therefore It should not place too many restrictions inside the metamodel that would undermine this feature. Constraints to determine the correctness of a model should be specified externally.

Understandability. The metamodel should be intuitive to understand for domain-experts that have more experience with attack-trees than technical aspects. It should define domain-concepts and use names familiar to the domain-experts.

Minimality. The metamodel should aim to remain clean, minimal and easy to understand and use. This means that full support of all extensions is not developed if this would significantly impact the complexity of the metamodel. Extensions that are only proposed by a single formalism are specifically analysed on this criteria.

Convenience of use. The metamodel should also be convenient to use/interact with. It should model concepts and references in such a way that often performed operations on an attack tree, such as quantitative analysis, should be straightforward.

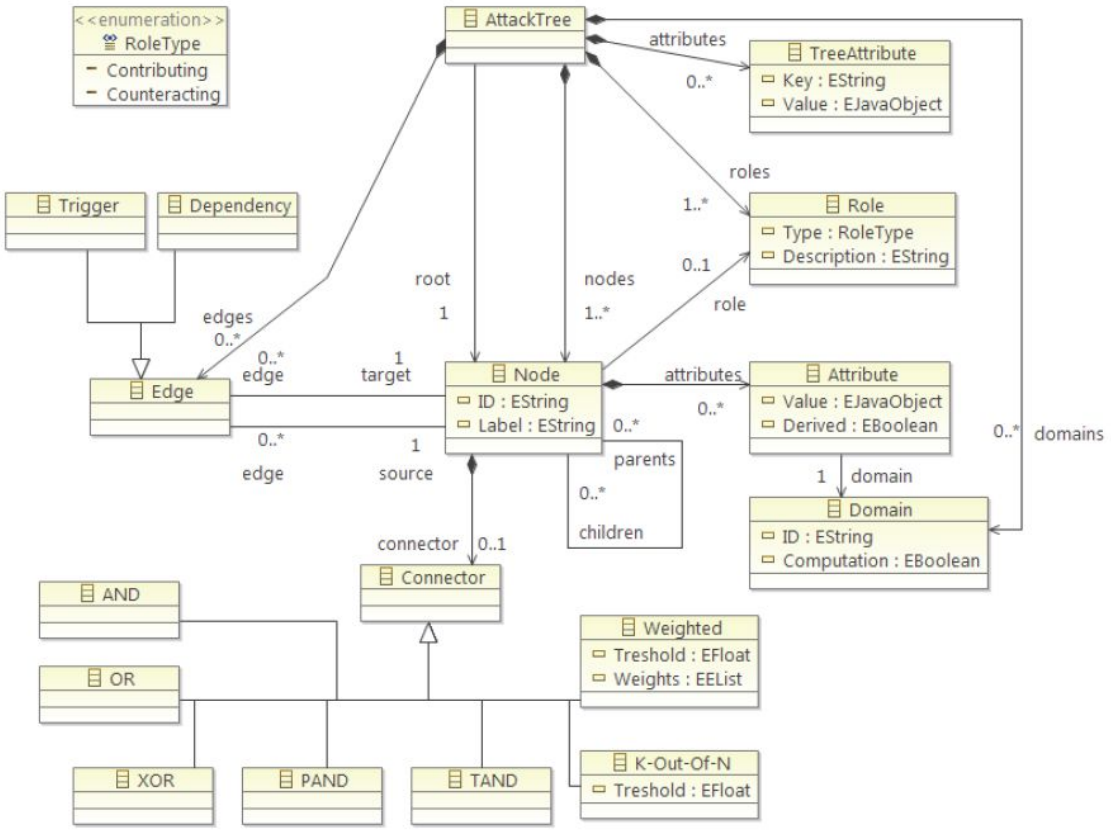

Figure 2.1: Attack Tree metamodel in graphical representation

Support results of attack tree analysis. The metamodel should also support (i.e. be able to model) the results that different formalisms/tools add to attack trees, so that multiple tools can be used in succession (where each tools uses the other tools output) and the resulting attack tree can contain the results of different analyses.

An extensive discussion of the solution is outside the scope of this deliverable; we merely show the full metamodel (Figure 2.1).

2.5 Metamodel design choices: meeting the requirements

We list a few of the design choices made in order to meet the above requirements.

- The type of a node (AND, OR, PAND, XOR, TAND, Weighted, k-out-of-n) are modelled as an optional attribute of the node. This not only allows the same classifier "Node" to be used for both leaves and gates, but also allows for partial models, where the tree structure is given but the nature of the gates is not yet fixed.
- The type of a node is modelled by subclasses of "Connector", rather than an enumerated type. This is an extensible solution (new kinds of gates can be accommodated

11

Formalism	Support	*Comment*
Attack Trees	Full	
Augmented attack trees	Full	
OWA trees	Full	
Parallel model for multi-parameter attack trees	Full	
Extened fault trees	Full	
Fault trees for security	Partial	*Two connectors not supported*
Enhanced attack trees	Full	
Dynamic fault trees for security	Partial	*Two connectors not supported*
Serial model for multi-paramter attack trees	Full	
Improved attack trees	Full	
Time-depdendent attack trees	Full	
Defense trees	Full	
Attack countermeasure trees	Full	
Attack-defense trees	Full	
Attack-response trees	Full	
Unified parameterizable attack trees	Full	
Augmented vulnerability trees	Full	
Protection trees	Full	
Security goal models	Partial	*Not all edges are supported*
Vulnerability cause graphs	Full	
Security activity graphs	Full	
Insecurity flows	Full (Secondary-citizen)	*Has different semantics*
Intrusion DAGs	Partial	*Multiple roots not supported*
Security goal indicator trees	Partial	*Not all edges are supported*
Boolean logic driven Markov process	Full (Secondary-citizen)	*Has different semantics*

Figure 2.2: Overview of metamodel support for AT-related formalisms

simply by adding a subtype of “Connector” but also enables some gates to have additional attributes, such as, for instance, the “Threshold” attribute of the k-out-of-n gate.

- There is a “Role” classifier associated with all nodes, which enables the modeller to distinguish attack and defense nodes.
- To add attributes to nodes, the chosen solution is to introduce the concept of a “Domain” (standing for the value domain of an attribute; essentially its type, reflected into the model), and have every “Attribute” point to one of those Domains. Every “Node” can have one or more Attributes.

To show that, for example, the metamodel of attack tree meets the first of the requirements above (“Support for existing formalisms”), Figure 2.2 cites from (Huistra, 2015) the list of AT-related formalisms that can be represented in the metamodel.

2.6 Attack Tree tool interoperability

To demonstrate the usability of the AT metamodel in practice, we have implemented a number of model transformations in Epsilon. Moreover, we also used Epsilonś constraint

language to define the conditions under which some of these transformations are well-defined. In summary, the available electronic artefacts are:

1. A model transformation definition from the XML format of ADTool to the metamodel, using the Epsilon Transformation Language ETL (see (Huistra, 2015))
2. A model transformation definition from the metamodel to the XML format of ADTool, using ETL (see (Huistra, 2015))
3. Validation constraints for the metamodel under which the above transformation is well-defined, using the Epsilon Verification Language EVL (see (Huistra, 2015))
4. A model transformation definition from the metamodel to itself, expanding general AND- and OR-nodes into binary ones, using ETL. (This is actually a case of refactoring rather than interoperability; however, it was used to achieve interoperability, see below.)
5. A model-to-text transformation definition from the metamodel to the input format of ATCalc, using the Epsilon Generation Language EGL.

With the help of these artefacts, the following interoperability scenarios were achieved:

- A transformation from ADTool models, for instance are produced by Treemaker, to ATCalc. This is achieved by chaining transformations 1 and 5. The transformation is used in the TRE$_S$PASS tool demonstration video on https://vimeo.com/143667223.
- A transformation from ADTool models, for instance produced by Treemaker, to ATAnalyzer. (ATAnalyzer actually takes the XML format of ADTool as input, but cannot cope with non-binary AND and OR nodes.) This is achieved by chaining transformations 1, 4 and 2. The transformation is used in the TRE$_S$PASS tool demonstration video on https://vimeo.com/145070436.

2.7 Model transformation for Attack Tree analysis

Apart from the model transformations described above, which were developed for the purpose of interoperability between the various analysis tools developed within TRE$_S$PASS, we also used the Attack Tree metamodel in one of the analysis methods themselves.

From Attack Trees to UPPAAL The work on ATTop, described in (Wolters, 2016), involves defining a translation from the Attack Tree formalism into UPPAAL (Behrmann, David, & Larsen, 2004), analyzing the resulting UPPAAL models, and using the outcome of that analysis to rank attacks.

An ECore metamodel for UPPAAL has been developed prior to our work by Gehrking et al. in (Gerking, Schäfer, Dziwok, & Heinzemann, 2015), together with a transformation to UPPAAL's native XML format. It turned out to be straightforward to define a model

transformation (in ETL) from the AT metamodel to the UPPAAL metamodel, implementing the translation defined in (Wolters, 2016).

From UPPAAL traces to Attack Trees Once an UPPAAL model has been checked, the results still need to be translated in terms of the original model, which is an attack tree. This is not immediate, due to the preceding transformation from that attack tree into UPPAAL. In (Brandt, 2016) we have studied this problem and devised a solution involving another transformation, namely from the output traces of UPPAAL to Attack Trees. Future work involves an implementation of this backward linkage.

3 Propagating Analysis Results back to the Socio-Technical Model

The attacks generated by tools such as Treemaker consider all three layers of an organisation: the physical, the virtual, and the social layer. From an attack tree it is however unclear, how much individual elements of an organisation contribute to the attack, and what the necessary countermeasures should be. In attack trees, this information is hidden in the steps performed as part of the attack; it cannot be mapped back to the model directly, since the actions usually involve several elements (attacker and targeted actor or asset). Especially for large attack trees, quickly understanding the relations between several model components results in a large quantity of interrelations, which are hard to grasp.

For analysis results, the same problem exists: the analyses compute the impact or execution time or needed skills and budget for an attack, but none of these numbers is easily expressed in the model or mapped back to it.

In this section we present several approaches for relating attributes of attacks such as likelihood of success, impact, and required time or skill level to the elements contributing to those attacks (Gu, Aslanyan, & Probst, 2016). Based on analysis results, and potentially combining results from different analyses, we annotate model elements with the quantitative properties. The annotation in the knowledge base (The TRE$_S$PASS Project, D2.4.1, 2016) can influence values fed back in the analysis in the next iteration of the TRE$_S$PASS process (The TRE$_S$PASS Project, D5.4.2, 2016).

In a first step, this information can be used for highlighting parts of the organisation that have a significant influence, for example through the heat map in the Attack Navigator Map (The TRE$_S$PASS Project, D4.2.2, 2016). The resulting visualisations provide a link between attacks on an organisation and the contribution of parts of an organisation to the attack and its impact, and can be used to guide countermeasure selection.

3.1 Values of Attack Components

The attack models generated from system models form the basis of analytic risk assessment. Properties of interest (Lenin, Willemson, & Sari, 2014) of these attacks include required resources, such as time or money, likelihood of success, or impact of the attack based on annotations of the leaf nodes in attack trees. Analyses (Aslanyan & Nielson,

2015a; The TRESPASS Project, D3.3.2, 2015) also identify the Pareto frontier of incomparable properties, for example, the likelihood of success of an attack, and the required budget.

To compute a value such as impact or risk for an attack, every leaf node must be mapped to some metrics. The mapping of actions to these metrics can be achieved by mapping the action and its arguments to a specific value. These metrics can represent any quantitative knowledge about components, for example, likelihood, time, price, impact, or probability distributions. The latter could describe behaviour of actors or timing distributions. For the mapping described in this section we only require an attack tree and a mapping from its nodes to an analysis result; how leaf nodes are associated with metrics and which analyses are performed on the tree does not influence the result.

We assume that this mapping is in place to perform the next steps, namely computing the contribution of components of an organisation to attacks.

3.2 Contribution of Components of an Organisation to Attacks

Now we put the different elements described above together to visualise the relation between attack trees and system models. Remember that we require all elements in the model to have unique identifiers; we use these identifiers to associate model components and attack tree actions.

As for attack trees we need a measure for how much a model element contributes to a given attack. We apply techniques similar to work on insiderness (Probst & Hansen, 2013).

3.2.1 Measuring Impact

Computing the actual impact of a model component on an attack is as difficult as computing the impact of an attack; the results can be used for ordering attacks or influence, but they should not be taken as absolute answers. With this in mind we have applied several techniques for measuring the impact of components on attacks.

The techniques presented here were chosen for their simplicity both in computing the value and in explaining the resulting mapping. The selection is base don the assumption that the difficult part is the computation of the impact of the analysis, which provides results for the individual components in the attack tree. The measuring approaches presented here only aggregate the computed impact under the assumption that more occurrences either in total or by weighing the impact resembles the overall impact of a component.

As mentioned before we require the attack model to support extraction of actor and assets from the actions in an attack tree, and actions are contained in the attack-tree leafs. Leaf labels provide information about type of action, performing actor, which asset is obtained,

and where the asset is obtained from. All this information is provided through the identifiers that connect the attack tree with the system model.

3.2.2 Counting Occurrences

The simplest concept of measuring impact is that of *counting occurrences* of identifiers. It computes for a given entity in how many places it contributes to the whole attack tree or a path. The occurrence-based impact ignores impact, likelihood, or other analysis results. It is either measured as absolute number or as percentage of occurrences of identifiers in the path or tree being analysed. It is computed per identifier id for a set of nodes in a subtree S of the attack tree that represents an attack, assuming that $id \in S$ returns 1 if true, and 0 otherwise, and that node n has successors $c \in succ(n)$:

$$\mathcal{I}_C(id, n) := \begin{cases} [x, x] & x = (id \in actor(n)) + (id \in assets(n)), \text{ if } n \text{ is a leaf node} \\ [l, u] & l = min(\mathcal{I}_C(id, c)), u = max(\mathcal{I}_C(id, c)), \text{ if } n \text{ is a disjunctive node} \\ [l, u] & l = \Sigma_c\{l' | [l', \_] = \mathcal{I}_C(id, c)\}, u = \Sigma_c\{u' | [\_, u'] = \mathcal{I}_C(id, c)\}, \text{ if } n \text{ is a conjunctive node} \end{cases} \tag{3.1}$$

As a first crude measure, this impact provides a defender with a quick overview of which components of the organisation actually occur in the attack tree.

The occurrence-based impact provides for every identifier a lower and an upper bound of occurrences; for conjunctive nodes these will be sum of all the occurrences at child nodes, for disjunctive nodes the lower bound is the minimum of the lower bounds, and the upper bound is the maximum of the upper bounds of the child nodes. The combination of lower and upper bounds provides a measure for how reliable the numbers are. It also allows to identify, whether certain elements occur in all attacks: if $\mathcal{I}(id, r) = [x, \_]$ for some identifier id, the root r of the attack tree, and $x > 0$, then the element with id is contained in every attack in the tree.

3.2.3 Weighted Sum

The impact factor based on occurrences in the generated attacks is a rather crude approximation, since every occurrence of an identifier is assigned the same impact independent of the *actual* contribution to the attack. Given that the analyses of attack trees described in Section 3.1 provide us with quantitative information about attacks, we can improve over the occurrence-based ranking by weighting occurrences of identifiers with the impact of the attack they occur in. The factors we can choose from are limited by available analyses only, but include, for example, the likelihood of success, required time, difficulty, and cost.

In contrast to the occurrence-based impact we now *include* one of the analysis results, by weighting the count for an identifier with the weight of the path, and potentially normalising

it. As before, it is either measured as absolute number or as percentage of occurrences of identifiers in a subtree of the tree being analysed. It is computed per identifier id for a node on a path in the attack tree, assuming that $id \in S$ returns 1 if true, and 0 otherwise, that node n has successors $c \in succ(n)$, and that val returns the result of the attack tree analysis for a node n in the (sub-)tree p:

$$\mathcal{I}_W(id, n, p) := \begin{cases} v_l & v_l = val(n,p) \star (id \in actor(n) + id \in assets(n)), \text{ if } n \text{ is a leaf node} \\ v_d & v_d = val(n,p) \star min(\mathcal{I}_W(id, c, p)), \text{ if } n \text{ is a disjunctive node} \\ v_{ca} & v_{ca} = val(n,p) \star \Sigma_c \mathcal{I}_W(id, c, p), \text{ if } n \text{ is a conjunctive node and we measure difficulty, time, or cost} \\ v_{cm} & v_{cm} = val(n,p) \star min(\mathcal{I}_W(id, c, p)), \text{ if } n \text{ is a conjunctive node and we measure likelihood of success} \end{cases} \quad (3.2)$$

3.3 Refining Parameters

Parameters of model components provide the input for the quantitative analysis developed in this work package and for the sensitivity analysis described in Chapter 5. The initial values are identified through properties of the elements selected for building the model.

Later on in the TRESPASS process (The TRESPASS Project, D5.4.2, 2016), the analysis results lead to the introduction of countermeasures. Countermeasures will most often result in changes of the parameters of a model component. For example, surveillance cameras or a guard lower the risk of an attacker successfully passing a location, an awareness training lowers the risk of a successful social-engineering attack, a fortified door increase time and skill needed to break open a door, etc.

These changes in parameter values are triggered by analysis results or by manual actions:

- The defender who is using the TRESPASS tools can choose to manually edit the parameter values through the ANM (The TRESPASS Project, D2.4.1, 2016);
- The defender can add countermeasures to the map and connect them to the locations in question, resulting in changed parameter values; or
- Based on the analysis results and the accumulated values for components in the model, the ANM can suggest countermeasures for locations that significantly contribute to an attack.

The first solution has the drawback that it overwrites the existing values, and that the reason is not documented in the model. However, the data is under version control, so changes will be captured with history, and when reasons are added to the text file, they would be visible and kept (The TRESPASS Project, D2.4.1, 2016) This documentation is also available in the second solution, which results in visible elements in the map. Finally, the third solution for refining parameters can trigger both other approaches, causing either direct changes to parameter values, or the addition of elements to the map.

4 Propagating Data Change - Data Pull

A crucial concept of TRE$_S$PASS language is the availability of an attack navigator map (ANM). The attack navigator model is represented by a set of files:

- socio-technical model description;
- model patterns;
- attack patterns;
- attack tree augmentation and annotation logic;
- attacker profiles;
- additional parameter sets for running the analysis tools.

Additionally, several files are available during the run time of the toolchains.

- attack tree generated by the Treemaker tool;
- attack tree generated by augmentation and annotation of the APL;
- analysis results from the ATA and ATE tools.

We refer the reader to Section 3.2 of D2.4.1 for details on the format and meaning of the configuration files containing entries regarding the data items presented above.

A user may wish to edit some data entries – e.g. add a model or attack pattern, change the annotation logic for attack trees, edit the attacker profiles, or tweak parameters of the analysis algorithms, tweak the socio-technical model. This may be done in several ways. The user can download the entire attack navigator model as an archive, edit them, and uploading back to the server. Additionally, ANM allows to edit the configuration files in its user interface. In order to see the impact of the changes to the original model, the user can rerun the analysis using the TRE$_S$PASS tools.

5 Consequence of Data Change — Sensitivity Analysis

Viewing the TRESPASS analysis as a function from input parameters (attack tree annotations) to success probabilities, we can frame input sensitivity in terms of the continuity of this function. In previous deliverables, we proposed parameter sweeps for Markov automata (The TRESPASS Project, D3.3.2, 2015, Section 4.2.2) as a straight-forward baseline strategy for detecting sensitivity. Here we look for more efficient algorithms and more precise statements about when and how strongly sensitivity manifests itself. In particular, we exploit the monotonicity of many metrics.

The TRESPASS process annotates attack tree nodes with data such as success probability, costs in time or money, and skill requirements. Each of these values must be estimated and can only approximate an ideal value. Moreover, they may simply change with time. The question thus arises how (small) changes in inputs can affect the outcome.

An important question is whether the analysis represents, in some sense, a continuous function. Even before defining this in a precise mathematical way, it is already apparent that using skill and cost thresholds as filters introduces discontinuity, with a dramatic impact on the robustness of the results: For instance, overestimating the cost of some basic step even slightly may lift an attack vector over budget and prune it from the tree, regardless of its impact or success probability. Therefore it becomes our goal to develop methods to investigate input sensitivity in individual cases.

5.1 Analytical Model

In the model used for Pareto analysis as discussed in (The TRESPASS Project, D3.3.2, 2015), Section 4.3, attack trees correspond to logical formulas using standard conjunction, disjunction, and negation ($\wedge$, $\vee$, $\neg$). Attack-defense trees are represented by adding a "player-switching" variant of negation ($\sim$). Each basic attack step corresponds to a variable in the formula and is assigned a success probability and a cost. If an efficient *algorithmic evaluation* is used, the tree must be linear and we may assume without loss of generality that only $\wedge$ and $\vee$ appear.[1]

Pareto analysis proceeds by considering boolean assignments that satisfy the attack tree formula; a value of *true* assigned to a variable means that the corresponding basic action is

[1] Otherwise, push negation into the leaves via De Morgan's laws and identify them as part of the corresponding basic action (which appears only once by the assumption of linearity).

taken, *false* that it is not. Costs and probabilities follow from this assignment. Actions that are not taken have zero cost but may nevertheless have a non-zero *incidence* probability to model cases where the effect of an action can occur independently.

An algorithm to compute the set of Pareto-efficient points with respect to probabilities and costs is provided in (Aslanyan & Nielson, 2015b). Given such a point, and the corresponding boolean assignment, an attack's total cost and probability are given by closed formulas. Let A and B denote subtrees of the attack tree:

$$\begin{aligned} \mathrm{P}(A \wedge B) &= \mathrm{P}(A) \cdot \mathrm{P}(B) \\ \mathrm{P}(A \vee B) &= 1 - (1 - \mathrm{P}(A)) \cdot (1 - \mathrm{P}(B)) \\ \mathrm{cost}_p(A \wedge B) &= \mathrm{cost}_p(A) + \mathrm{cost}_p(B) \\ \mathrm{cost}_p(A \vee B) &= \mathrm{cost}_p(A) + \mathrm{cost}_p(B) \\ & \qquad \text{where } p \in \{\mathrm{attacker}, \mathrm{defender}\} \end{aligned}$$

The result is a *multiaffine function* of the basic actions' costs and probabilities, respectively. Its partial derivatives with respect to each input parameter form a measure for sensitivity to that parameter, cf. (Rushdi, 1985).[2]

It can be seen from the formulas above that total cost is simply the sum of the basic costs; all derivatives are 1. I.e. the output uncertainty here depends solely and equally on the uncertainty present in each input. Attack probability, on the other hand, reduces to a linear expression when considered as a function of one basic input. Its derivative is a constant that depends on the rest of the parameters.

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{dP}(A)} \mathrm{P}(A \wedge B) &= \mathrm{P}(B) \\ \frac{\mathrm{d}}{\mathrm{dP}(A)} \mathrm{P}(A \vee B) &= 1 - \mathrm{P}(B) \end{aligned}$$

(Note how $\mathrm{P}(A)$ does not appear on the right-hand sides.)

These formulas describe the sensitivity of overall success probability to the basic steps' probabilities. Even where they are not implemented by an individual tool, they highlight valuable and intuitive guidelines for users:

1. Sensitivity to an uncertain parameter is mitigated (only) by lowering the success chance of steps that appear in conjuction ($\wedge$) with it.
2. Uncertainty is of low impact for attacks to which high-probability alternatives ($\vee$) exist. Conversely, exact measurements are important where an attack path appears much more likely than others.

Naturally, such considerations must be re-evaluated as parameters are refined and adjusted in the continuous TRESPASS process.

The discussion so far has concerned sensitivity between the same kind of values — attack probability to basic probabilities, total cost to individual costs. If we want to consider

[2] Alternative uncertainty measures are also discussed in (Rushdi, 1985).

sensitivity of probability versus costs or vice versa, it is clear, as already stated, that the respective function will not be continuous. In fact, the Pareto frontier yields the graph of maximum success probability as a (monotonous, step-wise constant) function of budget. Therefore, it answers the sensitivity question partially: The closer Pareto-efficient points fall in one dimension, the more sensitive are the other dimensions to that value: A small change to it could mean — moving along the Pareto frontier — that one of the other values changes suddenly.

5.2 Automata Models

When working with a translation to stochastic models such as interactive Markov chains, Markov automata, and priced timed automata, analytical formulas as above are not available. A straight-forward approach to sensitivity analysis in this case is to repeat the analysis over a range of values around an estimated parameter and to examine the resulting variation in the output.

This technique can be used with any tool in principle. It has been demonstrated with ATCalc on an example from the literature in (The TRESPASS Project, D3.3.2, 2015), as well as on the TRESPASS cloud case study (The TRESPASS Project, D7.2.2, 2016) in the ATCalc video presentation at https://vimeo.com/channels/trespass/143667223.

We can do a bit better than blind trials, however. If our models are to reflect reality, the results must represent monotonous functions:

- A higher probability on some basic action cannot lead to a lower overall success probability.
- Vice versa, a lowered basic probability cannot raise overall success probability.
- A lower cost on some basic action cannot decrease overall probability.
- A higher cost cannot increase success probability.

Analogous arguments apply to costs as outputs. This allows discontinuities in the output to be approached by a process of bisection. Also, interval bounds of an input estimate translate directly to bounds on the output. The cost-optimal reachability analysis of UPPAAL Cora already takes advantage of this structure by smart branch-and-bound algorithms.

6 Compositional and Incremental Model Analysis

The TRESPASS modelling language is organized into three layers — the architecture layer (the socio-technical model), attack DSL (Attack trees (AT) and Attack-defense trees (ADtrees)) and analysis models – automata based lower level formalism such as Priced timed automata (Kumar, Ruijters, & Stoelinga, 2015), Markov automata (Timmer, Katoen, van de Pol, & Stoelinga, 2012), I/O interactive Markov chains (Boudali, Crouzen, & Stoelinga, 2007). All these layers are tightly bound to each other through interchange of data formats (The TRESPASS Project, D3.2.1, 2015; The TRESPASS Project, D5.4.2, 2016; The TRESPASS Project, D6.1.2, 2015). The choice of attack trees and attack defense trees as the attack DSL is driven by the fact that they have an intuitive hierarchical multistep representation of attack scenarios. These models of reality allow brainstorming and it can be useful to document, brainstorm, and analyze system security. Further they are flexible and modular. Hence, these artefacts can be easily reused by saving them in an Attack Pattern Library (The TRESPASS Project, D5.3.2, 2015) .

As the layers in the TRESPASS modelling language are tighly coupled, we do not embed dynamic updates directly into the attack DSLs or the analysis models. However, the analysis models are themselves compositional which provides a lot of flexibility in terms of extensibility.

By compositionality, we mean that a model can be built by dividing it into several smaller sub-models. This keeps the model modular, yielding significant benefits in terms of efficiency and comprehensibility. Thus, even if there is an additional sub-system expressed as a sub-attack tree, all we need to do is to translate it into a automata and use it in the larger system.

Compositionality is important in sociotechnical security scenarios as the considered scenarios are complex and constructing and analysing them directly usually suffers from state space explosion (Hahn, Hartmanns, Hermanns, & Katoen, 2013; Crouzen & Lang, 2011). Thus, instead of composing the whole attack tree at once, we compose smaller sub-trees in a stepwise fashion and then minimise the state space after each composition using notions of strong, weak and branching bisimulation, as studied in process algebra. This keeps the model modular, flexible and easy to extend. Therefore, we circumvent the omnipresent state space explosion problem by not constructing a large stochastic model in the first place. A graphical representation of the approach is shown in Figure 6.4. Here we see that we first translate each atomic leaf and gate into an automata which is then composed iteratively.

In WP3, we developed several automated extraction techniques (The TRESPASS Project, D3.2.1, 2015) to extract a small analysis model for each element in the attack-defense tree. The complete analytical model is then obtained as composition of all stochastic models derived from the components (The TRESPASS Project, D3.2.1, 2015). The reason for having different formalisms is to analyse different aspects, like time, cost and probability of attacks. The general idea behind all these techniques is to represent the system as an automata enumerating all the states and stating the condition of reaching a goal state in temporal logic formulae. Model-checking algorithms (Baier & Katoen, n.d.) verify the reachability of the desired states and also provide an example sequence of how to reach the desired state. Since the extraction method consists of a one to one translation of elements in the attack tree to behavioural transition diagrams, if a future application requires a refinement to increasing granularity, one needs just to add the new stochastic model for the new model– rather than changing the entire model extraction process. Model transformation techniques are then used to trace back the output to input models, i.e., attack DSL and sociotechnical model itself.

This incremental approach also provides the means for an efficiency increase in case we want to propagate a change: for unchanged parts of the attack tree, we can in principle reuse previously generated stochastic models, which can be composed with the partial stochastic model computed for the changed component. This would increase the response time of the tool, when it is used in an interactive, design explorative mode.

Example. In order to show our compositional analysis methodology, we consider an ADTree sub-tree (Figure 6.3).The attack defense tree has been manually constructed to model an ATM compromise. The entire case description is provided in (The TRESPASS Project, D7.4.2, 2016).

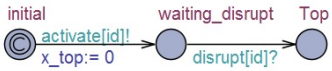

Figure 6.1: STA for top_event.

Consider the two tree nodes: 1) a leaf node (BAS) for example, *Blackbox attack* and 2) a top_event- *ATM crime* as shown in the ADTree given in Figure 6.3. Here by top_event we refer to the root of the attack-defense tree. We translate each of these two nodes into an equivalent automata, *stochastic timed automaton* (STA). Here, we use the STA as our underlying formalism just to showcase our approach. The viability of different extraction techniques has already been discussed in (The TRESPASS Project, D3.2.1, 2015).

The stochastic timed automata (STA) representing a BAS is shown in the Figure 6.2 and consists of the location {Initial, Wait, potentially_undetected, potentially_detected, ongoing, activated, execution, wait_success, success, Failure, stop} and two types of transitions: *Time delays* governed by probability distributions (here put as invariants lamda and lambda1 over the locations {activated, ongoing} respectively), and *probabilistic transitions* whose weights, like w1 and w2, are specified over the dotted edges to specify the probability distributions of the discrete transitions.

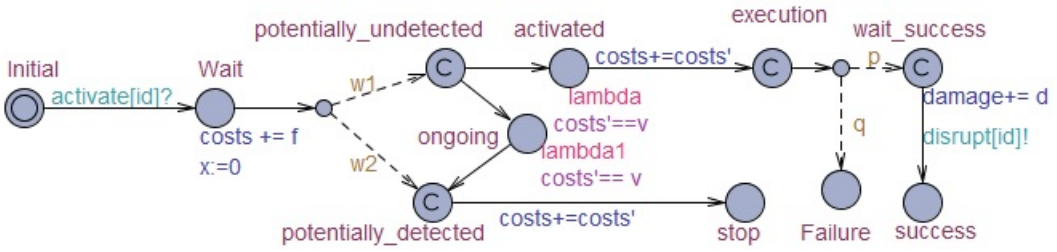

Figure 6.2: STA for the template of a BAS. Here id is a unique identifier for the BAS and x is a clock to track the duration of BAS. $costs$ is a global variable to keep track of all the accumulated costs, $costs'$ represents the variable costs per time unit spent in the location and $damage$ is a global variable to keep track of the accumulated damages. f, v and d are suitable constant values.

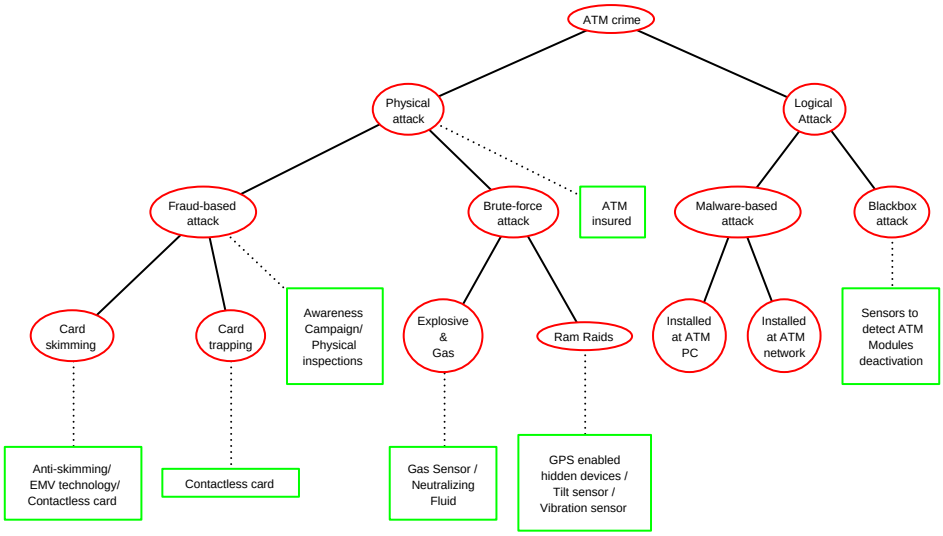

Figure 6.3: An excerpt of an ADTree on ATM crime.

The STA representing the top_event node (shown in Figure 6.1), consists of the locations – {Initial, waiting_disrupt, Top}. It initializes the system by emitting a broadcast signal (activate[id]!) and then waits for a broadcast signal disrupt[id]? from its child node. After receiving that signal, it makes a transition to the 'Top' location, which indicates the disruption of the AFT. Here, we use a clock x_top that keeps track of the global time.

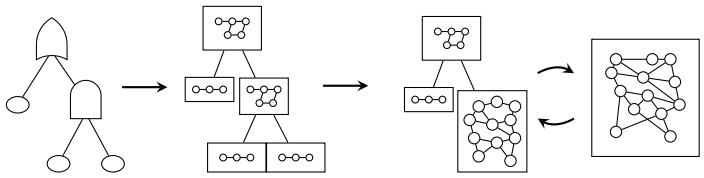

Figure 6.4: Graphical overview of compositional aggregation for AT models.

Thus, by translating each element of the ADTree into an equivalent STA and composing them together iteratively using proper broadcast signals, we obtain a network of STAs (NSTA) (see Figure 6.4). The resulting NSTA is then used to perform model checking, i.e. we verify the satisfiability of the security metrics formalized as queries over the resulting NSTA.

At present these automata have been drawn manually, however as explained in section 2, they can also be generated automatically from the attack DSLs. The output of the analysis can then be either visualized directly or can be traced back to the attack DSLs (Brandt, 2016).

Tool support Compositional analysis technique is used in the ATCalc and the ATtop tools. Both these tools have been extensively discussed in (The TRESPASS Project, D3.4.2, 2016). A video demo of ATCalc can be found at `https://vimeo.com/145070436`. Detailed methodology behind the tools is elaborated in (Kumar et al., 2015; The TRESPASS Project, D3.2.1, 2015; Arnold, Guck, Kumar, & Stoelinga, 2015).

7 Conclusion

In this deliverable, we have documented several aspects of dynamics that are embedded at the different stages in the TRE$_S$PASS language. We established metamodelling and the model transformation techniques as one of the key methodologies to establish both the forward and the backward linkages between the analysis tools and the attack DSLs.

Further, based on the user interactions with the ANM during design exploration (as developed in WP5 – the TRE$_S$PASS process), we summarised how these changes propagate through all models and data generated in WP3, how they change the analysis, and how the results are traced back to the ANM. So, any update to the socio-technical model or a change in the data parameters affect the analysis methodology properly, changes the analysis results, and can be traced back to the original socio-technical model.

In order to cope with the complexity of the socio-technical models we report on the modularity of the attack DSLs and its compositional analysis. Additionally, we report in this deliverable on the sensitivity analysis with respect to data parameters, which can systematically and robustly reduce the number of spurious and irrelevant attack scenarios, e.g., small variations over the same basic attack.

References

Arnold, F., Guck, D., Kumar, R., & Stoelinga, M. (2015). Sequential and parallel attack tree modelling. In F. Koornneef & C. van Gulijk (Eds.), *Computer safety, reliability, and security: Safecomp 2015 workshops, assure, decsos. isse, resa4ci, and sassur, delft, the netherlands, september 22, 2015, proceedings.* Retrieved from http://dx.doi.org/10.1007/978-3-319-24249-1_25 doi: 10.1007/978-3-319-24249-1_25

Aslanyan, Z., & Nielson, F. (2015a). Pareto efficient solutions of attack-defence trees. In R. Focardi & A. C. Myers (Eds.), *Principles of security and trust* (Vol. 9036, pp. 95–114). Springer. Retrieved from http://dx.doi.org/10.1007/978-3-662-46666-7_6 doi: 10.1007/978-3-662-46666-7_6

Aslanyan, Z., & Nielson, F. (2015b). Pareto efficient solutions of attack-defence trees. In *Principles of security and trust - 4th international conference, POST 2015, held as part of the european joint conferences on theory and practice of software, ETAPS 2015, london, uk, april 11-18, 2015, proceedings* (pp. 95–114). Retrieved from http://dx.doi.org/10.1007/978-3-662-46666-7_6 doi: 10.1007/978-3-662-46666-7_6

Baier, C., & Katoen, J.-P. (n.d.). *Principles of model checking*. MIT Press.

Behrmann, G., David, A., & Larsen, K. G. (2004). A tutorial on uppaal. In M. Bernardo & F. Corradini (Eds.), *Formal methods for the design of real-time systems, international school on formal methods for the design of computer, communication and software systems, SFM-RT 2004, bertinoro, italy, september 13-18, 2004, revised lectures* (Vol. 3185, pp. 200–236). Springer. Retrieved from http://dx.doi.org/10.1007/978-3-540-30080-9_7 doi: 10.1007/978-3-540-30080-9_7

Boudali, H., Crouzen, P., & Stoelinga, M. (2007). Dynamic fault tree analysis using input/output interactive Markov chains. In *The 37th annual IEEE/IFIP international conf. on dependable systems and networks proc.* (pp. 708–717). IEEE Computer Society. doi: 10.1109/DSN.2007.37

Brandt, J. (2016). Understanding attacks: Modeling the outcome of attack tree analysis. In *25th twente student conference on it* (Vol. 25). University of Twente. (BSc Thesis; see http://referaat.cs.utwente.nl/conference/25/paper)

Crouzen, P., & Lang, F. (2011). Smart reduction. In *Fundamental approaches to software engineering - 14th international conference, fase 2011* (Vol. 6603, p. 111-126). Springer.

Czarnecki, K., & Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Systems Journal*, *45*(3), 621–646. Retrieved from http://dx.doi.org/10.1147/sj.453.0621 doi: 10.1147/sj.453.0621

Gerking, C., Schäfer, W., Dziwok, S., & Heinzemann, C. (2015). Domain-specific model checking for cyber-physical systems. In M. Famelis, D. Ratiu, M. Seidl, & G. M. K. Selim (Eds.), *Proceedings of the 12th workshop on model-driven engineering, verifi-*

cation and validation co-located with ACM/IEEE 18th international conference on model driven engineering languages and systems, modevva@models 2015, ottawa, canada, september 29, 2015. (Vol. 1514, pp. 18–27). CEUR-WS.org. Retrieved from http://ceur-ws.org/Vol-1514/paper3.pdf

Gu, M., Aslanyan, Z., & Probst, C. W. (2016). Understanding how components of organisations contribute to attacks. In *21st nordic conference on secure it systems, nordsec 2016, oulu, finland, november 2-4, 2016.* Springer.

Hahn, E. M., Hartmanns, A., Hermanns, H., & Katoen, J.-P. (2013, October). A compositional modelling and analysis framework for stochastic hybrid systems. *Form. Methods Syst. Des.*, *43*(2), 191–232. Retrieved from http://dx.doi.org/10.1007/s10703-012-0167-z doi: 10.1007/s10703-012-0167-z

Huistra, D. (2015). *A unifying model for attack trees.* Research Project, University of Twente http://essay.utwente.nl/69399/.

Kolovos, D., Rose, L., García-Domínguez, A., & Paige, R. (2016). *The epsilon book.* Available on http://www.eclipse.org/epsilon/doc/book.

Kordy, B., Pièrre-Cambacédès, L., & Schweitzer, P. (2014). Dag-based attack and defense modeling: Don't miss the forest for the attack trees. *Computer Science Review*, *13-14*, 1–38. Retrieved from http://dx.doi.org/10.1016/j.cosrev.2014.07.001 doi: 10.1016/j.cosrev.2014.07.001

Kumar, R., Ruijters, E., & Stoelinga, M. (2015). Quantitative attack tree analysis via priced timed automata. In *13th int. conf. on formal modeling and analysis of timed systems, FORMATS* (pp. 156–171).

Lenin, A., Willemson, J., & Sari, D. P. (2014). Attacker profiling in quantitative security assessment based on attack trees. In K. Bernsmed & S. Fischer-Hübner (Eds.), *Nordic conference on secure it systems* (Vol. 8788, pp. 199–212). Retrieved from http://dx.doi.org/10.1007/978-3-319-11599-3_12 doi: 10.1007/978-3-319-11599-3_12

Mauw, S., & Oostdijk, M. (2005). Foundations of attack trees. In D. Won & S. Kim (Eds.), *Information security and cryptology - ICISC 2005, 8th international conference, seoul, korea, december 1-2, 2005, revised selected papers* (Vol. 3935, pp. 186–198). Springer. Retrieved from http://dx.doi.org/10.1007/11734727_17 doi: 10.1007/11734727_17

Mens, T., & Gorp, P. V. (2006). A taxonomy of model transformation. *Electr. Notes Theor. Comput. Sci.*, *152*, 125–142. Retrieved from http://dx.doi.org/10.1016/j.entcs.2005.10.021 doi: 10.1016/j.entcs.2005.10.021

Object Management Group (OMG). (2002). *OMG Unified Modeling Language (OMG-UML), Version 2.5.* OMG Document Number formal/2015-03-01 (http://www.omg.org/spec/UML/2.5).

Object Management Group (OMG). (2006). *Meta-Object Facility (MOF) Specification, Version 2.0.* OMG Document Number formal/2006-01-01 (http://www.omg.org/spec/MOF/2.0).

Object Management Group (OMG). (2015). *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.2.* OMG Document Number formal/2015-02-01 (http://www.omg.org/spec/QVT/1.2).

Probst, C. W., & Hansen, R. R. (2013). Reachability-based Impact as a Measure for Insiderness. In *5th International Workshop on Managing Insider Security Threats*

(MIST 2013).

Rushdi, A. M. (1985, December). Uncertainty analysis of fault-tree output. In *Ieee transactions on reliability* (Vol. 34, pp. 458–462).

Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *IEEE Computer*, *39*(2), 25–31. Retrieved from http://dx.doi.org/10.1109/MC.2006.58 doi: 10.1109/MC.2006.58

Schneier, B. (1999). Attack trees. *Dr. Dobb's Journal*.

Stahl, T., Voelter, M., & Czarnecki, K. (2006). *Model-driven software development: Technology, engineering, management*. John Wiley & Sons.

Steinberg, D., Budinsky, F., Paternostro, M., & Merks, E. (2009). *Emf: Eclipse modeling framework 2.0* (2nd ed.). Addison-Wesley Professional.

Stevens, P. (2007). A landscape of bidirectional model transformations. In R. Lämmel, J. Visser, & J. Saraiva (Eds.), *Generative and transformational techniques in software engineering ii, international summer school, GTTSE 2007, braga, portugal, july 2-7, 2007. revised papers* (Vol. 5235, pp. 408–424). Springer. Retrieved from http://dx.doi.org/10.1007/978-3-540-88643-3_10 doi: 10.1007/978-3-540-88643-3_10

The TRESPASS Project, D2.4.1. (2016). *TRESPASS information system.* (Deliverable D2.4.1)

The TRESPASS Project, D3.2.1. (2015). *TRESPASS extraction methods for stochastic models.* (Deliverable D3.2.1)

The TRESPASS Project, D3.3.2. (2015). *TRESPASS methods for stochastic analysis.* (Deliverable D3.3.2)

The TRESPASS Project, D3.4.1. (2014). *Attack generation from socio-technical security models.* (Deliverable D3.4.1)

The TRESPASS Project, D3.4.2. (2016). *Methods for attack generation, preventive measures, and ranking.* (Deliverable D3.4.2)

The TRESPASS Project, D4.2.2. (2016). *Methods for visualization of information security risks.* (Deliverable D4.2.2)

The TRESPASS Project, D4.3.3. (2016). *Visualizations of socio-technical dimensions of information-security risks.* (Deliverable D4.3.3)

The TRESPASS Project, D5.3.2. (2015). *Best practices for model creation and sharing.* (Deliverable D5.3.2)

The TRESPASS Project, D5.3.3. (2016). *Best practices for model maintenance.* (Deliverable D5.3.3)

The TRESPASS Project, D5.4.2. (2016). *The integrated TRESPASS process.* (Deliverable D5.4.2)

The TRESPASS Project, D6.1.2. (2015). *Final requirements for tool integration.* (Deliverable D6.1.2)

The TRESPASS Project, D6.4.3. (2016). *TRESPASS deployment and maintenance plan.* (Deliverable D6.4.3)

The TRESPASS Project, D6.4.4. (2016). *The integrated TRESPASS tools.* (Deliverable D6.4.4)

The TRESPASS Project, D7.2.2. (2016). *Final report case study a.* (Deliverable D7.2.2)

The TRESPASS Project, D7.4.2. (2016). *Final report case study c.* (Deliverable D7.4.2)

Timmer, M., Katoen, J., van de Pol, J., & Stoelinga, M. (2012). Efficient Modelling and Generation of Markov Automata. In *23rd International Conference of Concurrency Theor,y CONCUR* (Vol. 7454, pp. 364–379). Springer.

Wolters, N. (2016). *Quantitative analysis of attack trees with timed automata*. Unpublished master's thesis, University of Twente. (See http://fmt.cs.utwente.nl/education/master/297/)