D8.1 – Requirements and Interface

A. Schmitz, T. Schnitzler (Fraunhofer ISST)
J. Jürjens (Fraunhofer ISST & TU Dortmund)
D. Ríos (URJC)
J. Williams (ABDN)
S. Houmb (SecureNOK)

Pending of approval from the Research Executive Agency - EC

Document Information

Document Number	D8.1
Document Title	Requirements and Interface
Version	1.0
Status	Final
Work Package	WP 8
Deliverable Type	Report
Contractual Date of Delivery	31.01.2013
Actual Date of Delivery	31.01.2013
Responsible Unit	Fraunhofer ISST
Contributors	Fraunhofer, TUD, URJC, ABDN, UNITN, SNOK
Keyword List	Tool Platform Support
Dissemination level	PU

SECONOMICS Consortium

SECONOMICS "Socio-Economics meets Security" (Contract No. 285223) is a Collaborative project within the 7th Framework Programme, theme SEC-2011.6.4-1 SEC-2011.7.5-2 ICT. The consortium members are:

1	UNIVERSITÀ DEGLI STUDI DI TRENTO	Università Degli Studi di Trento (UNITN), 38100 Trento, Italy www.unitn.it	Project Manager: prof. Fabio MASSACCI Fabio.Massacci@unitn.it
2	DEEPBLUE	DEEP BLUE Srl (DBL) 00193 Roma, Italy www.dblue.it	Contact: Alessandra TEDESSCHI Alessandra.tedeschi@dblue.it
3	Fraunhofer ISST	Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V., Hansastr. 27c, 80686 Munich, Germany http://www.fraunhofer.de/	Contact: Prof. Jan Jürjens jan.juerjens@isst.fraunhofer.de
4	Universidad Rey Juan Carlos	UNIVERSIDAD REY JUAN CARLOS, Calle TulipanS/N, 28933, Mostoles (Madrid), Spain	Contact: Prof. David Rios Insua david.rios@urjc.es
5	UNIVERSITY OF ABERDEEN	THE UNIVERSITY COURT OF THE UNIVERSITY OF ABERDEEN, a Scottish charity (No. SC013683) whose principal administrative office is at King's College Regent Walk, AB24 3FX, Aberdeen, United Kingdom http://www.abdn.ac.uk/	Contact: Prof. Julian Williams julian.williams@abdn.ac.uk
6	TMB Transports Metropolitans de Barcelona	FERROCARRIL METROPOLITA DE BARCELONA SA, Carrer 60 Zona Franca, 21-23, 08040, Barcelona, Spain http://www.tmb.cat/ca/home	Contact: Michael Pellot mpellot@tmb.cat
7	Atos	ATOS ORIGIN SOCIEDAD ANONIMA ESPANOLA, Calle Albarracin, 25, 28037, Madrid, Spain http://es.atos.net/es-es/	Contact: Silvia Castellvi Catala silvia.castellvi@atosresearch.eu
8	SECURENOK	SECURE-NOK AS, Professor Olav Hanssensvei, 7A, 4021, Stavanger , Norway Postadress: P.O. Box 8034, 4068, Stavanger, Norway http://www.securenok.com/	Contact: Siv Houmb sivhoumb@securenok.com
9	SOÚ Institute of Sociology AS CR	INSTITUTE OF SOCIOLOGY OF THE ACADEMY OF SCIENCES OF THE CZECH REPUBLIC PUBLIC RESEARCH INSTITUTION, Jilska 1, 11000, Praha 1, Czech Republic http://www.soc.cas.cz/	Contact: Dr Zdenka Mansfeldová zdenka.mansfeldova@soc.cas.cz
10	nationalgrid THE POWER OF ACTION	NATIONAL GRID ELECTRICITY TRANSMISSION PLC, The Strand, 1-3, WC2N 5EH, London, United Kingdom	Contact: Dr Ruprai Raminder Raminder.Ruprai@uk.ngrid.com
11	ANADOLU ÜNİVERSİTESİ	ANADOLU UNIVERSITY, SCHOOL OF CIVIL AVIATION Iki Eylul Kampusu, 26470, Eskisehir, Turkey	Contact: Nalan Ergun nergun@anadolu.edu.tr

Document change record

Version	Date	Status	Author (Unit)	Description
0.1	2012-06-18	Draft	Andreas Schmitz (ISST)	Table of Contents
0.1	2012-06-18	Draft	Jan Jürjens (ISST)	Verify Table of Contents
0.1	2012-08-28	Draft	Fabio Massacci (UNITN)	Contribution for Section 2.6
0.1	2012-09-22	Draft	Siv Houmb (SecureNOK)	Contribution for Section 2.5
0.2	2012-12-05	Draft	Andreas Schmitz (ISST)	Full draft of deliverable
0.2	2012-12-05	Draft	Sebastian Pape (TUD)	Review draft
0.2	2012-12-05	Draft	Jan Jürjens (ISST)	Review draft
0.3	2012-12-19	Draft	Michela Angeli (UNITN)	Quality check
0.4	2013-01-11	Draft	Javier Cano (URJC)	Scientific review
0.5	2013-01-13	Draft	Julian Williams (ABDN)	Contribution
0.6	2013-01-13	Finalizing	Andreas Schmitz (ISST)	Finalizing deliverable
0.6	2013-01-18	Finalizing	Theodor Schnitzler (ISST)	Assistance
0.7	2013-01-23	Finalizing	Fabio Massacci (UNITN) Woohyun Shim (UNITN)	Scientific review
1.0	2013-01-29	Final	Andreas Schmitz (ISST)	Final deliverable

Table of Contents

D8.1 – Requirements and Interface 1
SECONOMICS Consortium 2
Document change record 3
Table of Contents 4
Table of figures 7
Executive summary 8
1 Introduction 9
1.1 Aim of WP8 9
1.2 Document Overview 10
1.3 Definition of the stakeholders of the tool platform 10
2 Evaluation of different available tools 11
2.1 Economic Model 11
2.1.1 Essential Features of a Mathematical Programming Features for Economic Models 11
2.1.2 An example of a data envelope problem for a Nash equilibrium 12
2.1.3 Meta-code for a general form of the Envelope Problem 13
2.1.4 Meta-code for the attack and defense game 14
2.1.5 Problems and development issues envisioned in SECONOMICS 16
2.2 Adversarial Risk Analysis 17
2.2.1 Basic ARA models 18
2.2.2 Game theory 18
2.2.3 Simultaneous Defend-Attack-Scenario 19
2.2.4 The ARA Approach 20
2.3 GeNIe 21
2.3.1 Description 21
2.3.2 Inability of GeNIe to solve the Sequential Defend-Attack problem 23
2.4 Bugs 27
2.4.1 Description 28
2.4.2 Inability of OpenBUGS to solve the SEQ-DA problem 28
2.5 SecInvest 30
2.6 SeCMER 31
2.7 Matlab Overview 33

2.8 Carisma 34
3 Techniques and possibilities of the tool platform 38
3.1 Matlab Java connection 38
3.1.1 JMI 38
3.1.2 RMI - Remote Method Invocation 38
3.1.3 Matlabcontrol 39
3.1.4 Matlab Builder 39
3.2 Eclipse 40
3.3 Plugin Interface 41
3.4 Creation of graphical editors 42
3.4.1 Eclipse Modeling Framework 42
3.4.2 ECore meta-model 43
3.4.3 Graphiti Framework 44
4 Design of tool platform 46
4.1 Overview of the design 46
4.2 Design of the integrated Frontend 46
4.2.1 General 46
4.2.2 Integration in detail 47
4.2.3 Different parts of each model 48
4.2.4 User interface of the integrated tool box 48
4.2.5 Level of Abstraction 49
4.3 Design of different tools 51
4.3.1 Influence Diagram Editor 51
4.3.2 Decision tree visualization 52
4.3.3 Adversarial Risk Analysis 52
4.3.4 Economic Visualization 54
4.3.5 Return on Security Investment 58
4.3.6 BPMN 58
4.3.7 Statistical Tool 60
4.4 Links to other work packages 61
4.4.1 Work package 4 61
4.4.2 Work package 5 61
4.4.3 Work package 6 61

5 Summary and conclusion 62
6 Bibliography 63
7 Appendix 1 65

Table of figures

Figure 1: a worked example with simple functional forms 17
Figure 2: Simultaneous Defend-Attack Modell 19
Figure 3: Example Influence Diagram showing a DA Model with private information 21
Figure 4: Defender-Attacker Example visualized with GeNIe Modeler 23
Figure 5: The Sequential Defend-Attack model 24
Figure 6: The Defender's decision problem 24
Figure 7: GeNIe model for the Defender's decision problem 25
Figure 8: The Defender's analysis of the Attacker's problem 26
Figure 9: GeNIe model for the Defender's analysis of the Attacker's problem 27
Figure 10: Snapshot of OpenBugs example 28
Figure 11: OpenBUGS model for the Defender' decision problem 29
Figure 12: Bayesian Belief Network topology of the SecInvest decision engine 31
Figure 13: Conceptual model of SeCMER 32
Figure 14: SeCMER arguments meta-model 33
Figure 15: The user interface of Carisma 35
Figure 16: Evolution analysis pipeline 36
Figure 17: Carisma Tool Architecture 36
Figure 18: Remote control a Matlab session with "matlabcontrol" 39
Figure 19: Execution of exported functions in Matlab Compiler Runtime 40
Figure 20: Relations between concrete classes and the Eclipse Plugin interface 42
Figure 21: Scheme of ECore meta-model 43
Figure 22: Feature concept of Graphiti 45
Figure 23: Schematic structure of the toolbox 46
Figure 24: Concept of the tool framework with Matlab integration 47
Figure 25: User interface of Carisma with a frontend for a Matlab implementation running in the background 49
Figure 26: DA Model in Influence Diagram Editor Prototype 52
Figure 27: Example decision tree with defender D and attacker A and State S 54
Figure 28: Kahnemann-Tversky type loss function with fixed points about the target levels of C, I and A 56
Figure 29: Loss Function Illustration: Expected loss from vulnerabilities versus investment penalities 56
Figure 30: The trade-off curve is the difference between these two losses 57
Figure 31: First row: Deviation from the equilibrium from a single unit shock in C; Second row: Evolution of Control variable; For example 1 and 2 57
Figure 32: Simple BPMN process with parallel sequence flows 59
Figure 33: Extended BPMN process with risks and mitigations 60

Executive summary

This report will evaluate the main tools and models in relation to the new toolbox provided by our partners. Furthermore the requirements of our partners regarding this toolbox will be gathered and analyzed. Some ideas for comfort increasing editors to support the models from our partners implemented in the toolbox will also be presented. An implementation strategy for the toolbox will be defined. Especially the implementation of the models, the implementation and design of the editors and the whole all surrounding framework will be shown. Further the links to other work packages will be pointed out. These are the presumable requirements, which will be implemented for the next deliverables. For this deliverable a prototype was implemented.

1 Introduction

This first report illustrates the requirements for and the interface of the SECONOMICS toolbox developed within WP8 at Fraunhofer ISST.

The report is divided into an evaluation of tools to be used for our platform, further techniques and possibilities our toolbox will make use of, and a presentation of the methodology and the design of the platform.

We will start this report with pointing out the aim of our work package in general, followed by an overview of this report and a definition of the different stakeholders using our integrated tool platform.

1.1 Aim of WP8

In this section we aim at giving a brief description of the targets of WP8. The main goal of WP8 is to develop an integrated toolbox for the SECONOMICS project. Why is an integrated toolbox needed? Different from developed methodologies, the models developed in this project are computer-assisted and computationally intensive. Due to this, computer implementations are needed. As will be explained in chapter 4, these implementations can be done with Matlab. The different models shall be integrated into one tool in WP8. The main advantage is that, rather than being just a collection of Matlab implementations; the tool is guided and easy to use, with all different models having the same interface with a homogeneous view. So the work done in WP8 is structured into the following steps. First, an overview of the existing tools used or developed by one partner of the consortium should be given and they should be evaluated. Within this evaluation, some of the tools can be used or integrated within this toolbox (for example Matlab), while from other alternative tools good ideas can be gained (for example GeNIe). After knowing the available tools, the requirements for such a platform have to be determined and the interfaces between the tool framework and the models have to be defined. After and while defining the requirements in WP8 the necessary technology has to be found and evaluated. This technology includes techniques to integrate the tools, especially Matlab, a modeling framework, Graphiti in our case, and a graph drawing engine like jMathtools or Gnuplot. Specifically, a good drawing engine has still to be found and evaluated and the best solution should be used. One of the most important tasks in WP8 is the design of the tool and the tool framework. It has to be specified how the different model implementations can be integrated, how they can be visualized, how editors can aid with the models and the interfaces between the tool framework and the models has to be designed. The tool should be easy to use, have an integrated and homogeneous view, the tools should be neatly arranged, while it should be still easy to expand. The second important task is the tool implementation. The tool framework has to be implemented and aid for the models will be given. After all, WP8 shall deliver an easy to use integrated tool box.

1.2 Document Overview

- First, we point out the different stakeholders of our tool in section 1.3.
- In chapter 2, we evaluate different tools used or developed by one partner of this project. We also evaluate third party tools that can be useful for the development of our toolbox. Therefore, we also analyze to what extent these tools meet the requirements of our toolbox. All the presented tools show the expertise and give general ideas for the implementation of specific parts of our toolbox.
- In chapter 3, we present different techniques we will use in our tool platform, and we also point out its resulting possibilities.
- In chapter 4, we explain the methodic, the general design of the platform and the integration of each available model. We also present approaches for different tools designed within our toolbox. Furthermore, we point out the links to the other work packages, respectively which concepts or parts of our toolbox are used by which partner.
- In chapter 5, we conclude this report by summarizing its main ideas.

1.3 Definition of the stakeholders of the tool platform

The tool that will be developed in WP8 of SECONOMICS will comprise an integrated tool, which will aid in using the mathematical models developed in this project, by providing different levels of abstraction, as well as an easy interface for using the models, like visual editors for specific instances of a model class, and easy entering of parameters by providing wizards.

Many people might be influenced by this tool when it is used for analyzing security and the results are to be implemented. So, we provide here some ideas about the different kind of stakeholders present in this project. The stakeholders will use the tool expecting to gain advantage out of the results. They can be structured by two different dimensions. The first dimension is hierarchical, i.e. according to their position in the target company. In the target company it is necessary to use this tool with different kinds of users at different hierarchical levels of the company, because the needed information and knowledge are only available at different levels. Hence, the tool is also structured in different levels of abstraction. On the lowest level we have designers of the model classes, who are mostly the universities within this project. At the second level we might have a model designer of a specific instance of one class. This might be a consultant, who knows the classes of models. Then, we have different groups of people, who provide statistical data for calculations of the models. After that, there might be some strategic decisions to be made. This can also be done by a consultant, with knowledge of the model class, in cooperation with the decision maker. In the last step, the decision maker has some key parameters which can be modified and analyzed.

The second dimension is an organizational way. Besides, the company itself, which should reach a better level of security in one specific aspect, there are several different stakeholders, who are different depending on the case study. In most cases, they will be regulators.

2 Evaluation of different available tools

In this chapter, already existing free and commercial tools are evaluated, with respect to their possible reusability within the tool platform of WP8. In addition, tools delivered from other work packages are also presented as well as third party tools.

2.1 Economic Model

At the heart of the SECONOMICS project is a tool that can evaluate security problems and make predictions on behavior using economic concepts. For this purpose the SECONOMICS tool needs to have a robust mathematical underpinning, grounded in an economic methodology. Generally we think of economic methodologies as modeling behaviors using of agents in an economic system with well specified pay-off functions that relate observed metrics to levels of welfare. Deliverable 6.1 reviews the relevant formulations of a variety of economic models that are applicable to the SECONOMICS domain, specified in the case study work packages WP1, WP2 and WP3.

2.1.1 Essential Features of a Mathematical Programming Features for Economic Models

For research purposes functional programming is a useful tool for defining and evaluating models based on mathematical functions. For example in adversarial game theory mathematical models can be used to evaluate the pay-offs for various agents acting strategically by maximizing their individual pay-off functions.

For particular types of games this can involve a complex series of optimizations that are often characterized by 'embedded envelope' problems. In an envelope problem a series of variables can also be a series of functions. For instance a hyper surface $g(.)$ in n dimensional space maybe represented by a series of parametric equations $\mathbf{x}(c)$, where $\mathbf{x}$ is a vector and c is a parameter (or vector of parameters in the general case). The envelope of a family of curves is when $g(\mathbf{x},c)=0$, whereby the curves that satisfy this condition have a point that satisfy an arbitrary point of tangency (usually describing an equilibrium).

The requirement of a modeling tool should be that is can solve well specified envelope problems of a general nature. Algebraically this is defined in the following steps. Consider a standard microeconomic problem:

$$\max_{\mathbf{x}} f(\mathbf{x},\mathbf{r}), \quad \text{s.t.} \quad \mathbf{g}(\mathbf{x},\mathbf{r}) = \mathbf{0}$$

Here f is a function translating control variables $\mathbf{x}$ under a set of constraints characterized by the vector function $\mathbf{g}$ and $\mathbf{0}$ is an appropriate length vector of zeros. We assume a Lagrangian view of systems, rather than a Kuhn-Tucker approach (whereby $\mathbf{g}(\mathbf{x},\mathbf{r})\geq 0$), as most Kuhn-Tucker problems can be expressed as simpler Lagrangian problems by basic rearrangement of the initial problem. Here $\mathbf{r}$ is a vector of parameters of state that characterize the economic interactions.

This simple setup characterizes the majority of economic mathematical problems, agents are represented by their payoff functions and seek to maximize their payoffs by acting strategically. The actions of other agents typically enter their payoff function through the constraints in $\mathbf{g}$.

Setting,

$$\mathbf{x}^*(\mathbf{r}) = \arg\max_{\mathbf{x}} f(\mathbf{x},\mathbf{r}), \quad \text{s.t.} \quad \mathbf{g}(\mathbf{x},\mathbf{r}) = \mathbf{0}$$

and

$$f^*(\mathbf{r}) = f(\mathbf{x}^*(\mathbf{r}),\mathbf{r})$$

We can restate the problem as a generalized Lagrangian constrained optimization:

$$\mathcal{L}(\mathbf{x},\mathbf{r}) = f(\mathbf{x},\mathbf{r}) - \lambda.\mathbf{g}(\mathbf{x},\mathbf{r})$$

where λ is a vector of Lagrange multipliers with elements equal to the length of $\mathbf{g}$ and $\cdot$ represents the dot product of two vectors.

Equilibrium is defined by:

$$\left.\frac{\partial \mathcal{L}(\mathbf{x},\mathbf{r})}{\partial r_i}\right|_{\mathbf{x}=\mathbf{x}^*(\mathbf{r}),\ \lambda=\lambda(\mathbf{r})} \equiv \nabla\mathcal{L}(\mathbf{x}^*,\mathbf{r}) \equiv \frac{df^*(\mathbf{r})}{dr_i}$$

2.1.2 An example of a data envelope problem for a Nash equilibrium

Consider the following problem. Let there be two representative agents in the system, attackers and targets. Attackers maximize their payoff

$$\pi_A = R\sigma(N_A, x_T) - C(N_A)$$

Where *R* is the reward for a successful attack, σ(.) is the probability of a successful attack as a function of the number of attacks N_A and the level of defensive effort x_T and *C*(.) is the cost of attacking as a function of the number of attacks N_A. Targets seek to minimize losses and as such have the following payoff function

$$-\pi_T = L\sigma(N_A, x_T) + D(x_T)$$

Where *L* is the loss to the target given a successful attack and *D*(.) is the cost of defense as a function of the chosen level of defensive effort. Attackers and targets are assumed to have representative budget constraints, such that:

$$C(N_A) - c = 0$$

$$D(x_T) - d = 0$$

where *c* and *d* are constants. The envelope problem in this case is derived from the following set of problems:

$$\max_{N_A} \pi_A \equiv \max_{N_A} R\sigma(N_A, x_T) - C(N_A), \quad s.t. \quad C(N_A) - c = 0$$

and

$$\max_{x_T} -\pi_T \equiv \min_{x_T} -L\sigma(N_A, x_T) - D(x_T), \quad s.t. \quad D(x_T) - d = 0$$

For notational purposes we also set:

$$N_A^* = \arg\max_{N_A} R\sigma(N_A, x_T) - C(N_A), \quad s.t. \quad C(N_A) - c = 0$$

and

$$x_T^* = \arg\min_{x_T} -L\sigma(N_A, x_T) - D(x_T), \quad s.t. \quad D(x_T) - d = 0$$

The equilibrium solution for this pair of equations is relatively simple.

First we set out the Lagrangian problem for the attacker:

$$\mathcal{L}_A(N_A, x_T) = R\sigma(N_A, x_T) - (1-\lambda_A)C(N_A) - c$$

and

$$\mathcal{L}_T(N_A, x_T) = -L\sigma(N_A, x_T) - (1-\lambda_T)D(x_T) - d$$

For the attacker the envelope problem is:

$$\left.\frac{\partial \mathcal{L}_A(N_A, x_T)}{\partial x_T}\right|_{N_A = N_A^*(x_T),\ \ \lambda_A = \lambda_A(x_T)}$$

and for the target the solution is of the form:

$$\left.\frac{\partial \mathcal{L}_T(N_A, x_T)}{\partial N_A}\right|_{x_T = x_T^*(N_A),\ \ \lambda_T = \lambda_T(N_A)}$$

For the attacker the solution is of the form, the optimal number of attacks N^*_A for a given level of exogenous representative defensive effort x_T. For the defender the solution is of the form of the optimal defensive effort, x^*_T, with respect to an exogenous level of attacking intensity N_A. There is a final equilibrium solution whereby:

$$\tilde{N}_A = N_A^*\left(x_T^*\right)$$
$$\tilde{x}_T = x_T^*\left(N_A^*\right)$$

This is the best response of attackers to the best response of targets, this is known as the Nash equilibrium and is the predicted level of attacking intensity and defensive effort, given a set of chosen functional forms of σ(.), *C*(.) and *D*(.) and the constants *R, L, c* and *d.*

2.1.3 Meta-code for a general form of the Envelope Problem

This section builds a set of meta-codes for the general envelope problem per agent. In this case we assume that we have a constrained optimization solver that uses a conventional approach such as sequential quadratic programming to obtain an unconstrained minimum.

First we construct a series of function declarations for the objective function and constraint.

```
function [f]=fun(x,r)
f=eval(arguments in, global variables)%this is an arbitrary func-
tional form


function [g]=con(x,r)
g=eval(arguments in,global variables)%this is an arbitrary con-
straint
function [x0]=init(r)
x0=eval(arguments in, global variables)%this function builds a set
of initial values for x, for the optimiser
```

next we build a new function that will run the optimization.

```
function [xstar]=envelopeeProblem(r_upper,r_lower,diff)
% r_upper is a list of upper bounds for the state variable r
% r_lower is a list of lower bounds for the state variable r
%diff is a scalar or equivalent length list of finite differences
for the optimisation.
%first check the consistency of r_upper and r_lower
n=length(r_upper)
if n~=length(r_lower)
     error(`upper and lower bounds are not consistent')
end
%we now build an array that will contain the optimal values of
rstar
R=cell(n,1);%a cell is an unstructured array
for i=1:n
     rarray=r_lower(i):diff(i):r_upper(i);
     R{i,1}=rarray;
     N(i)=length(rarray);
end
%this now contains the domains over which the envelope function
will be evaluated.
xstar = zeros(N);% the intergers in N present the dimensions of
the hypersurface that will be described by xstar. This is an n di-
mensional hypersurface.
ind=index(xstar); This creates an array that indexes the elements
of xstar
for ii=ind
%this creates a vector ii that cycles through the array R and con-
verts the loci of R into the array xstar
     sym x r l%declares x r l as algebraic symbols
     fun L=fun(x,r)-dot(l, con(x,r));
     for j=1:n
          r(j)=rarray{j,1}(ii(j));
     end
     x0=init(r);
     xstar(ii)=fmax(L,x0,r,l);in the order, function, initial val-
ues, state variables and lagrange multipliers
end
```

In this case we assume that we have the optimization functions `fmin`, `index`, `sym` and `fun` that respectively are: a functional minimization; a means of indexing a hyper dimensional array; a means of declaring function arguments and a function declaration that detects the pre-specified functions for the envelope problem.

The output array `xstar` may then be outputted and sent to another envelope function as its array of state variables r (as in a game theory problem).

2.1.4 Meta-code for the attack and defense game

The previous code allows for an n-dimensional envelope problem that solves a constrained min-max optimization problem. For most applications the dimension of **x** and **r** are usually unity, problems of higher dimension than two or three are difficult to con-

ceptualize and most economic applications agents would be considered unrealistically hyper-rational to solve such complex problems.

For the attack and defense game, N_A serves as **x** for the attacker and x_T is **r**. For the defender the situation is reversed, x_T serves as **x** for the attacker and N_A is the state variable **r.**

We shall now outline meta-code for solving such a game. First we must define a series of functions for the various elements of the payoffs for the attacker and the target.

```
function s=sigma(NA,xT)
s=eval(input arguments and any global variables)
function C=costAttack(NA)
C=eval(input arguments and any global variables)
function C=costDefence(xT)
D=eval(input arguments and any global variables)
function NA0 = initialAttackerEffort(xT)
NA0 =eval(input arguments and any global variables)
function xT0 = initialTargetEffort(NAT)
xT0 =eval(input arguments and any global variables)

%Note that we keep the functional forms of sigma, costAttack,
costDefence problem specific at this juncture.
```

Next we need to set the envelope problem solvers for the attacker and the targets.

```
function
[NAstar] = attackerEnvelopeProblem(xT_lower,xt_upper,diff)

xT_array= xT_lower:diff:xt_upper; builds an array of xT over a set
domain
NAstar=zeros(length(xT,array));
for i=1:length(xT_array)
     sym NA xT R c l%declares the variables
     xT=xT_array(i);
     fun LA = R*sigma(NA,xT)-(1 - l)*C(NA)-c%declare the Lagrangian
form of the optimisation.
     NA0= initialAttackerEffort(xT)
     NAstar(i)  =  fmax(LA,NA0,xT,l);one  dimensional  maximisation
problem.
end
```

In this step we compute the optimal defensive effort as a function of attacking intensity.

```
function
xTstar = targetEnvelopeProblem(NA_lower,NA_upper,diff)
%in this case a sensible starting point is the limits(NAstar) from
the first optimisation.
NA_array= NA_lower:diff:NA_upper; builds an array of xT over a set
domain
xTstar=zeros(length(NA_array));
for i=1:length(NA_array)
```

```
     sym NA xT L d l%declares the variables
     NA=NA_array(i);
     fun LT = L*sigma(NA,xT)+(1 - l)*D(NA)+d%declare the Lagrangian
form of the optimisation.
     xT0= initialDefensiveEffort(xT)
     NAstar(i)  =  fmax(-LT,xT0,NA,l);one  dimensional  maximisation
problem.
end
```

To compute the Nash equilibrium we simply need to interpolate and overlay the curves of x^*_T (`xTstar`) and N^*_A (`NAstar`) and compute points of intersection. If only one point of intersection exists then the problem is said to have a unique Nash equilibrium (i.e. the best response of attackers to the best response of defenders). See Figure 1 for a worked example with simple functional forms.
For most economic applications the form of the payoff and constraint functions are kept deliberately simple, to allow for a tractable solution. However, in SECONOMICS we envision that many of the problems faced by the security case studies (outlined in deliverables 1.3, 2.3 and 3.3) will be more complex and require the types of numerical solutions that may be solved using the types of mathematical approaches outlined above.

2.1.5 Problems and development issues envisioned in SECONOMICS

A major issue for practical use of economic methodologies for predicting security behavior is that the cost and reward functions have highly unusual properties. They normally exhibit substantial discontinuities and have functional forms not readily described by simple mathematical functions, even in a piecewise fashion.

This provides a substantial difficulty for conventional optimization engines based around quadratic programming and quasi-Newtonian methods. This affects the choice of solver, represented by `fmax` in the meta-code.

Several alternative approaches exist other than sequential quadratic programming solvers and their analogues. Most solvers that overcome the problems inherent in quasi-Newtonian solvers do so by negating the computation of the Hessian matrix of second order derivatives.
Two approaches that have been used are: Monte-Carlo methods and the related genetic algorithms.
Monte-Carlo approaches effectively randomly sample the space of the vector variable **x** and evaluate the function at each point. The basic optimization then records the function evaluation that achieves the maxima. The larger the number of trials the more likely the optimization is likely to find the optimal point.

An obvious drawback is that the space of **x** maybe very large and as such a huge number of repeat trials maybe needed to achieve any confidence in the result. Even for one dimensional optimization problems a simple Monte-Carlo with a fixed distributional drawn is incredibly inefficient.

An alternative approach that keeps the Hessian free nature of simple Monte-Carlo simulations are genetic algorithms. These importance sample the draws of the vector **x** and

derive new distributions of draws based on the history of observations from prior distributions in a Bayesian set-up.

For instance a simple genetic algorithm could work as follows: For a vector function *f*(**x**), start with an initial guess **x**0 population **X**0 of draws (usually greater than 20) of **x**, from a multinormal distribution with preset covariance matrix Ω and centered on **x**0 (unfortunately few solvers are assumption free). Find the draw from **X**0, denoted **x**1 that achieves the maxima of *f*(**x**). Centre the next set of draws **X**1 at **x**1 with covariance matrix (**x**0- **x**1) (**x**0- **x**1)´, where ´ denoted the conjugate transpose. Each generation retains the best moment attributes of the prior generation. If the function has a tight global minima, then with enough generations and a large enough number of draws per generation, the algorithm will converge tightly to the global minima.

An example of matching reaction functions for a data envelope problem for an attack and defense game is shown in Figure 1. In this case the attackers and targets have both a numerical (unbroken line) and analytic solution (dashed line) to the envelope problem. In this case we assume that costs of security investment and attack are linear and the probability of successful attack follows Gordon and Loeb's 1/e rule. See Deliverable 6.1, section 9 for a more detailed explanation of this approach.

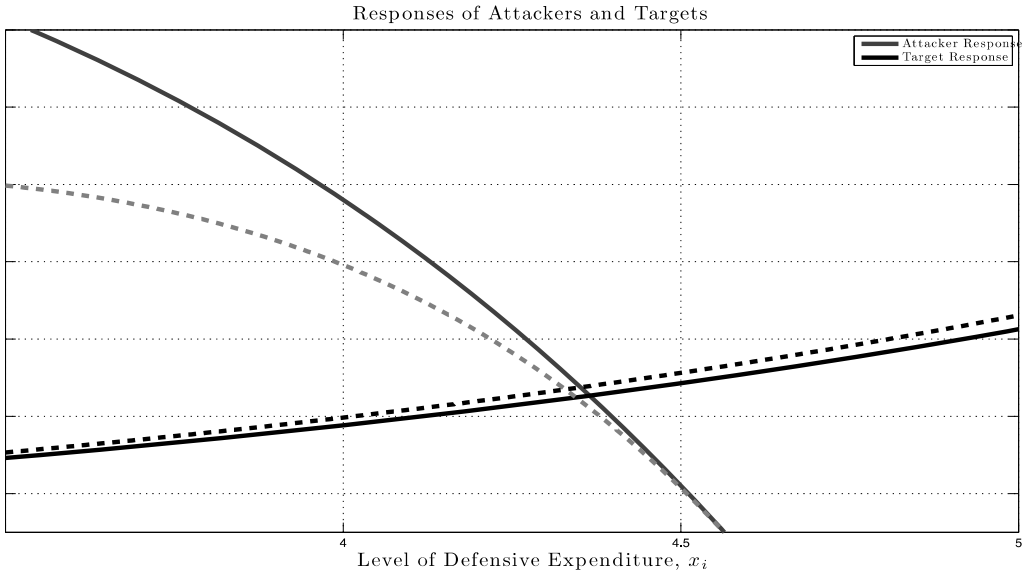


Figure 1: a worked example with simple functional forms

2.2 Adversarial Risk Analysis

Adversarial Risk Analysis is a concept to describe and evaluate attacks on systems. This concept is described in detail in [12] and is based on game theory, but this theory was modified in various points to match the attack-defend-scenario. ARA aims at providing one-sided prescriptive support to one of the intervening agents, the Defender, based on a subjective expected utility model, treating the adversary's decisions as uncertainties. In order to predict the adversary's actions, it models his decision problem and try to assess his probabilities and utilities. Assuming that the adversary is an expected utility

maximizer, we can predict the adversary's actions by finding his maximum expected utility action. Uncertainty about the adversary's probabilities and utilities is propagated over to the adversary's decision. Sometimes, such assessment may lead to a hierarchy of nested decision problems. They apply the ARA framework to five prototypical models relevant in security risk analysis.

2.2.1 Basic ARA models

The Basic ARA models are:

- Sequential Defend-Attack (SEQ-DA). In this model, the Defender first makes a decision about his defend policy and after that the Attacker chooses his attack, depending on the Defender's strategy.
- Simultaneous Defend-Attack (SIM-DA): In this model, the Defender and Attacker choose their policy independent from each other, i.e. the Attacker does not now the chosen defense policy.
- Sequential Attack -Defend (SEQ-AD). In this model, the Attacker first performs an attack. Then, having suffered it, the Defender chooses a defense
- Sequential Defend-Attack-Defend (SEQ-DAD): In this model, the Defender first deploys defensive resources. Then, the Attacker, having observed such decision, performs an attack. Finally, the Defender tries to recover from the attack as best as she can (for example, by calling for support).
- Sequential Defend-Attack with Private Information (SEQ-DA-PI). In this model, the Defender wants to keep secrecy about vulnerabilities of sites she is trying to protect, as this information can be used by the Attacker to increase the chances of success and the expected impact of an attack. In this model, the Defender moves first by choosing a defense and, then, having observed it, the Attacker moves by choosing an attack.

2.2.2 Game theory

In the following, a brief explanation of game theory will be given. Game theory is an economic approach to model decision making situations. The result of the game depends not only on the decisions made by one actor, but on the decisions of all actors in the game. Therefore all actors affect each other. A game is defined by a set of actors, actions, possible final states and, associated with them, a reward distribution for each actor. An actor is a participant of the system. He has different possible actions at choice. A basic assumption of game theory is that an actor always tries to maximize his utility function, given as the reward of the game, or to minimize his loss function. The reward of the game depends on the achieved final state, which in turn, depends with different probabilities on the chosen actions of all actors. There are different information distributions. The first possibility is that an actor knows the action of the other one, which is called a sequential game. This means, that the actor, whose decision is known, has to choose first. Another possibility is that no one knows the action of the other one. Then it is a simultaneous game.

The simplest scenario is the Simultaneous Defend-Attack-Scenario, abbreviated as SIM-DA. This is a game theoretical scenario, where the defender first has to choose his defending strategy. With the knowledge of the defenders choice, the attacker makes his decision about his attack strategy.

2.2.3 Simultaneous Defend-Attack-Scenario

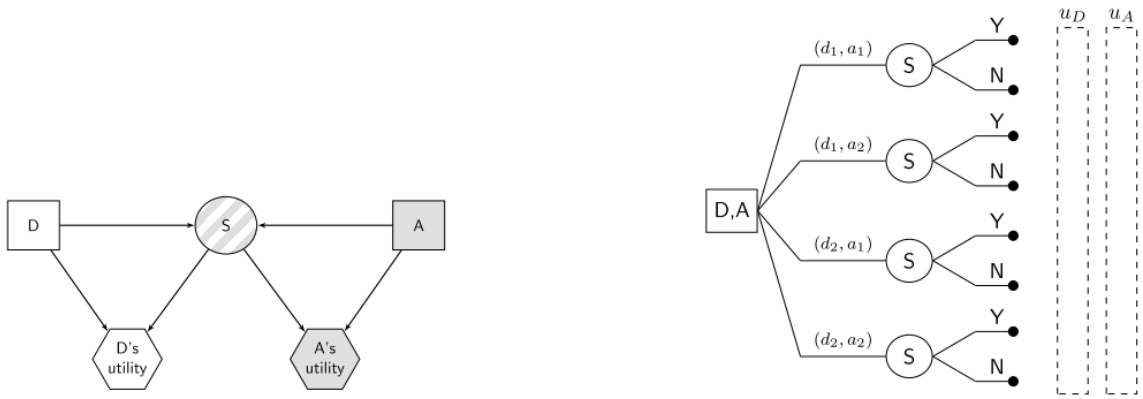
This approach is described in [11]. There are two different players in this game theoretical approach: The defender D and the attacker A. Both choose their strategy simultaneously without any information about the choice of the other. Let the defender have the discrete set of choices $D = \{d_1, d_2, .., d_m\}$ and the attacker $A = \{a_1, a_2, .., a_k\}$. The chosen choice of the defender is $d \in D$ and the choice of the attacker is $a \in A$. The only uncertainty in this game is the outcome of the system S, with $S = 1$ for success and $S = 0$ for failure of an attack. Both the Attacker and the Defender have their own assessment of the probability of the result of an attack which depends on the chosen defense and attack strategy: $p_D(S = s|d, a)$ and $p_A(S = s|d, a)$. Additionally, both have a utility function which depends on the own chosen strategy and on the result of the attack. The utility function of the Defender is denoted as $u_D(d, s)$ and the one of the attacker is denoted as $u_A(a, s)$. So the reward of each player depends on their choice and on the outcome of the system. All these distributions are known by each other under the common knowledge assumption. This assumption is very common in game theory. It assumes that the utility functions and probability distributions are known by each player, so they are able to optimize their own utility function and can anticipate adversary decisions. Therefore, each player knows the expected return that both would get from the system under a given strategy pair $(d, a) \in D \times A$. It can be computed as $E[u(d, a)] = p_D(S = 0|d, a) \cdot u_D(d, S = 0) + p_D(S = 1|d, a) \cdot u_D(S = 1)$ for the defender and similarly for the attacker. A Nash equilibrium (d^*, a^*) for this game maximizes the expected return for both:

$$E[u(d^*, a^*)] \geq E[u(d, a^*) \; \forall \; d \in D$$

and

$$E[u(d^*, a^*)] \geq E[u(d^*, a) \; \forall \; a \in A$$

In [11] it is stressed out, that there can be several equilibriums and finding them may require the use of randomized strategies.



(a) Influence diagram (b) Game tree ($m = n = 2$)

Figure 2: Simultaneous Defend-Attack Modell

One improvement to the game theoretical approach is to drop the common knowledge assumption and use probability distributions instead of the utility function and other

knowledge about the attacker. This makes the calculations more realistic and can be well used in these case studies.

2.2.4 The ARA Approach

More realistically, we weaken the common prior knowledge assumption. We assume that we support the Defender in solving the Simultaneous Defend-Attack model. As reflected in Figure 2, the Defender has to choose a defense $d \in D$, whose consequences depend on the success of an attack $a \in A$ simultaneously chosen by the Attacker, which is, therefore, uncertain for the Defender at the time she makes her decision. By standard decision theory, the Defender should maximize his expected utility. He knows his utility function $u_D(d,s)$ and his probability assessment p_D over S, conditional on (d,a). However, he does not know the Attacker's decision a at node A. He expresses his uncertainty through a probability distribution $\pi_D(A=a)$. Then he should find the optimal defend decision d^*. To be able to solve the equitations for that, the Defender needs to sess $\pi_D(A)$. To do so, suppose he thinks that the Attacker is an expected utility maximize. The Attacker would look for the attack $a^* \in A$ providing him maximum expected utility. In general, the Defender will be uncertain about the Attacker's utility function and probabilities (u_A, p_A, π_A) required to solve such problem. Suppose that we model all information available to the Defender about (u_A, p_A, π_A) through a probability tion (U_A, P_A, Π_A). Then, and this will aid us in assessing $\pi_D(A)$.

Although (U_A, P_A) could be directly elicited from the Defender, eliciting $\Pi_A(D)$ may require further analysis, leading to the next level of recursive thinking: the Defender would need to think about how the Attacker analyses her problem. Note that $\Pi_A(D)$ incorporates two sources of uncertainty:

- the Attacker's uncertainty about the Defender's choice, represented through his beliefs $\pi_A(A)$, and
- the Defender's uncertainty about the probabilistic model π_A used by the Attacker topredict what the Defender will choose, assessed from her perspective through $\pi_A \sim \Pi_A$.

In the above, the Defender presumes that the Attacker thinks that he is an expected utility maximizer trying to solve a decision problem like the one described. Therefore, in order for the Defender to assess the distribution $\Pi_A(D)$, he will elicit $(U_A, P_A) \sim F$ from his viewpoint, through the analysis of his decision problem, as thought by the Attacker. This reduces the assessment of $\Pi_A(D)$ to the computation of the distribution

$$D|A^1 \sim argmax_{d \in D} \sum_{a \in A} \left[\sum_{s \in \{0,1\}} U_D(d,s) P_D(S=s|d,a) \right] \Pi_D(A^1 = a),$$

assuming the Defender is able to assess $\Pi_D(A^1)$, where A^1 represents the Attacker's decision within the Defender's second level of recursive thinking: the nested decision model used by the Defender to predict the Attacker's analysis of his decision problem. To assess the distribution above, the Defender needs to elicit $(U_D, P_D) \sim G$, representing his probabilistic knowledge about how the Attacker may estimate the Defender's utility function $u_D(d,a)$ and the corresponding probability p_D over $S|d,a$; when he analyses how the Attacker thinks about his decision problem. Again, the elicitation of $\Pi_D(A^1)$ might require further recursive thinking from the Defender. This would lead to recursive assessments.

To simplify the discussion, we have assumed that the recursive decision models used to assess A_i and D_i are a reflection of each other. Moreover, the choice sets for the Defender and the Attacker are the same in all the recursive models: D and A, respectively. This hierarchy of nested models would stop at a level in which the Defender lacks the information necessary to assess the distribution F^i or G^i associated with the decision analysis of A^i and D^i, respectively. At this point, the Defender would holistically assign an unconditional probability distribution over A^i and D^i, respectively, without going deeper in the hierarchy, summarizing all remaining information he might have through the direct assessment of $\Pi_{D^{i-1}}(A^i)$ or $\Pi_{A^i}(D^i)$, as might correspond.
This should only give a short introduction. For further knowledge of this model or the other ones mentioned read the corresponding deliverable D5.1.

2.3 GeNIe

This chapter first we want to describe the GeNIe tool, which has some good ideas and then explain why it is inappropriate for this project to use.

2.3.1 Description

The GeNIe (Graphical Network Interface) [13] software package can be used to create decision theoretic models intuitively, using a graphical click-and-drop interface. GeNIe is the graphical interface to SMILE, a fully portable Bayesian inference engine developed by the Decision Systems Laboratory at University of Pittsburgh and thoroughly tested in the field since 1998. GeNIe 2.0 is the latest version of GeNIe. GeNIe 1.0, released to the community in 1998, has received a wide acceptance within both academia and industry. It is free of charge for academic purposes.

GeNIe permits dealing with decision influence diagrams and allows computing optimal policies. It is also possible to create probabilistic influence diagrams that allow propagating evidence and performing inference and forecast. Both exact and simulation methods are included.
In this section GeNIe is described in detail, because it provides some basic features needed in our toolbox. GeNIe enables to define influence diagrams. An influence diagram shows how a target result value is being influenced by decisions and probability states, and how these states influence each other. In Figure 3 the schematic illustration of an exemplary DA model with private information is given.

Figure 3: Example Influence Diagram showing a DA Model with private information

These diagrams consist of three main node types. The most important one is the decision node. It is displayed as a rectangle and represents a decision which has to be made by a decision maker. Most times, each decision corresponds to a unique actor but this is not necessary in general. In the example shown in Figure 3 the two nodes called *"Defender"* and *"Attacker"* are decision nodes as the actor with the same name has to choose an action. The second node type is the value node, which represents the reward of the system for each actor, who has to maximize this reward with his decisions. In this example, there are two concurrent value nodes, one for each actor. One of the limitations of GeNIe is that it is not possible to create concurrent actors, as the system always tries to optimize the total outcome. The third important node type is the chance node, shown as an ellipse. A chance is in general a system state, either probabilistic or constant depending on another node. In our example, there are several chance nodes. The node called *"private information",* for instance, is a probability distribution, stating which type of system we have to protect. It could be either stable or unstable. The last very important structure in an influence diagram is the arc. An arc shows a dependency between two nodes. A decision can depend on a chance, if the decision is allowed to depend on the given state of the chance. So when the decision has to be made, the current state is already available. A chance can depend on another chance. This is possible for two states depending on each other, or a representation of imperfect information. This modeling of imperfect information can also be done for decisions. In our example, the node called *"Defender Strategy"* represents such imperfect information for the attacker. The attacker has full knowledge of the node called *"Defender Strategy".* This node depends on the defender's decision, but it just propagates the correct information in combination with likelihood; otherwise wrong information is delivered. Finally, arcs can connect chance and decision nodes with value nodes. The value node then depends on the input nodes connected to them. In the example of Figure 3, the value node *"Value for Defender"* is representing the outcome for the defender. It depends on the choice made by the defender (for example, costs to implement the choice) and the result of the system state, which is a probability distribution.

Every node has properties connected to it. These properties define the behavior of the nodes. One disadvantage of GeNIe is that it only has discrete probability distributions and just discrete amount of decisions.

A decision node has a set of actions defined. The input nodes are for the calculations of returns for each combination of states of the input nodes. Chance nodes are defined by states and a probability for each state and each combination of states of the input nodes. So it has several dimensions, one for each input node, and one for the output state to be filled. An example for a chance node is *"Private Information"*, which represents an internal state with two possible values: *"stable"* or *"unstable"*, each combined with a probability of occurrence. The *"Defender Strategy"* node is a model of imperfect information. It uses the choice of the defender as input and then gives the correct information to the attacker with a certain probability, otherwise wrong information. The attacked system is modeled as a chance node, too. It has to states *"broke"* and *"hold"* and two input nodes connected with an arc. Therefore it has a three dimensional grid filled with the associated probabilities. It describes the probability given the two choices of attacker and defender, with which the system fails. The value node has no state itself. It describes the outcome for a decision depending on different input nodes. In our

example *"Value for Defender"* is a value node. It receives to arcs from defender and system, which represent the decision of the defender and the status of the system. With respect to this, it defines a discrete outcome.

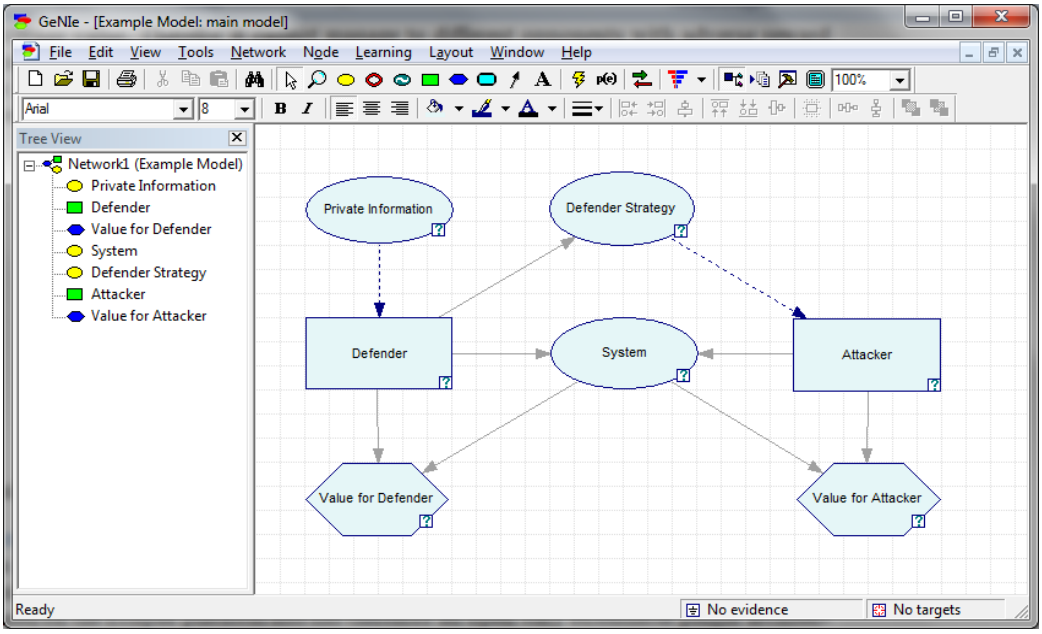


Figure 4: Defender-Attacker Example visualized with GeNIe Modeler

In Figure 4, the example of the DA model described above is visualized with the GeNIe modeler. GeNIe has some disadvantages in order to simulate defend-attack models, which shall be improved in our tool. First of all, it can model different outcome values and different decisions but tries to optimize the average value. Therefore it cannot manage two different opponents with adverse reward value functions. Therefore, it is impossible to model Defend-Attack models. The second disadvantage is the limitation to discrete sets of actions and probability functions. So it is impossible to model investment decisions.

2.3.2 Inability of GeNIe to solve the Sequential Defend-Attack problem

In game-theoretical analysis, a common assumption is that the Defender knows how the Attacker will solve his problem, i.e., she accurately knows the Attacker's true p_A and u_A. However, in the ARA analysis this common knowledge assumption is weakened: the Defender does not actually know (p_A, u_A) in the Sequential Defender-Attacker problem displayed in Figure 5.

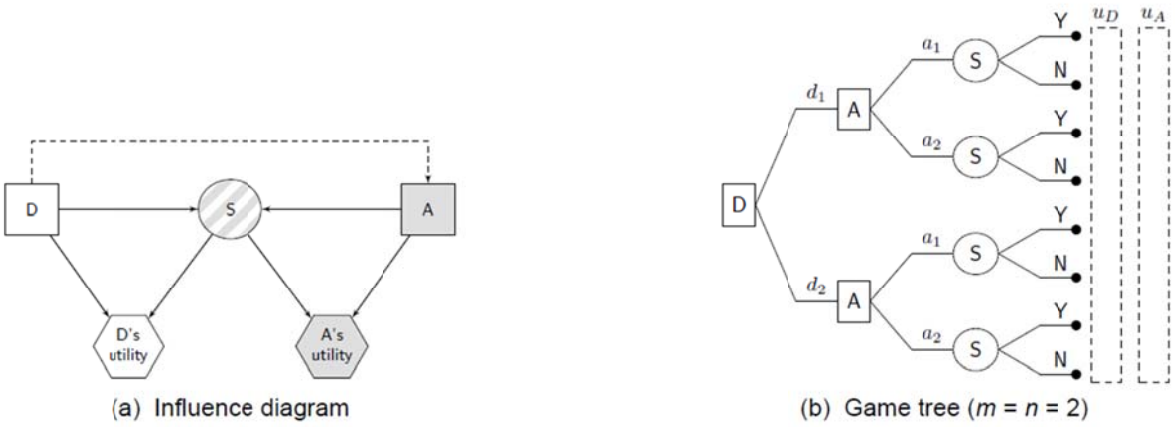


Figure 5: The Sequential Defend-Attack model

We thus consider the Defender's problem as a standard decision analysis problem: the Defender's influence diagram in Figure 6, no longer has the hexagonal utility node with the Attacker's information and his decision node is perceived as random variable.

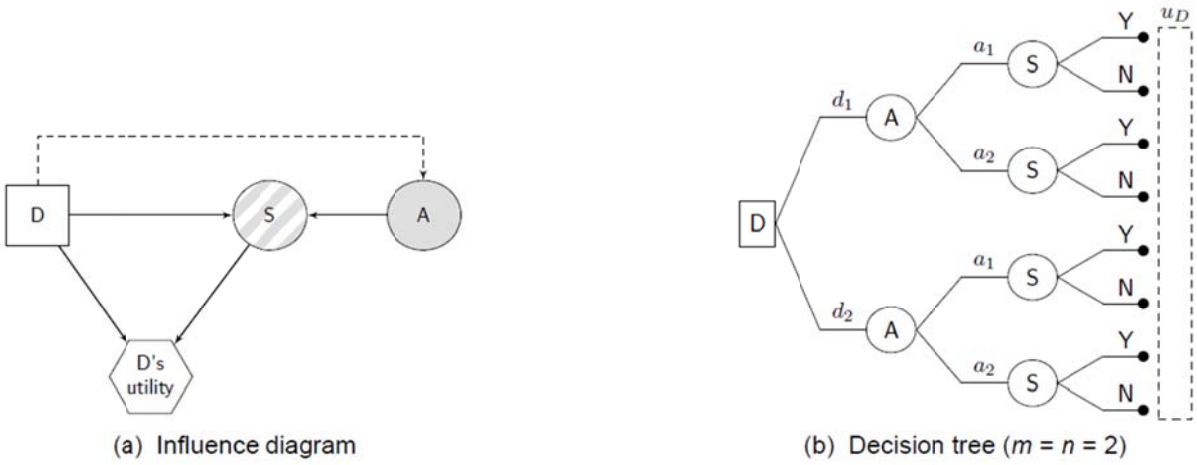


Figure 6: The Defender's decision problem

Similarly, her decision tree denotes uncertainty about the Attacker's decision by replacing [A] with (A) and including a reference only to the Defender's utility function. However, as we shall see, she may have beliefs about (p_A, u_A), which will be relevant in our analysis. This problem is implemented in the GeNIe model `Seq_D_A_prob_D.xdsl`, see Figure 7.

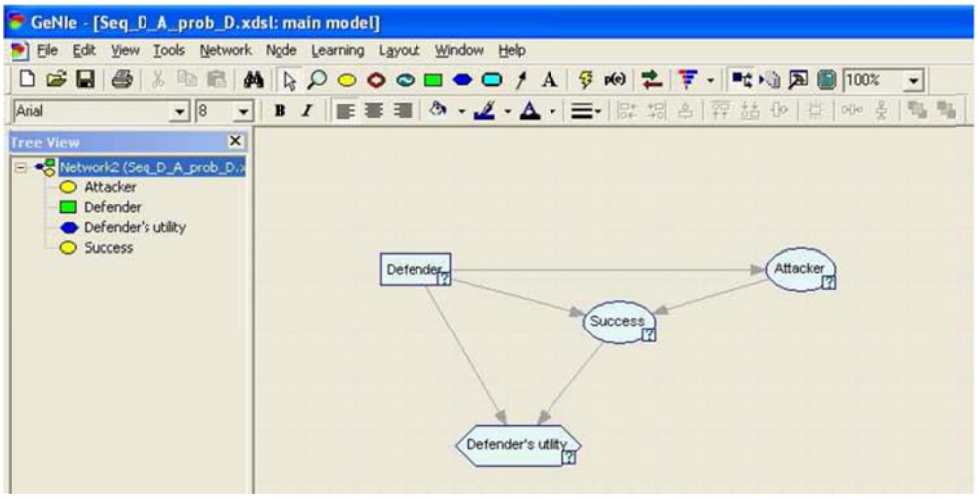


Figure 7: GeNIe model for the Defender's decision problem

In this particular example, we have introduced the following data from p. 19 of deliverable *D5.1 - Basic Models for Security Risk Analysis* into the nodes:

- Her expected utilities $u_D(d, s)$ into node $\langle u_D \rangle$;
- Her estimated probabilities on the success of the attack $p_D(S|d,a)$ into node (S);
- Node (A) should contain the information about her estimated probabilities about what attack will choose the Attacker once he has observed her defense $\hat{p}_D(A = a_j|d)$. In this case, and for illustration purposes, we have included the values in Table 2 of deliverable *D5.1 - Basic Models for Security Risk Analysis*.

After *Updating* (solving) the model, the estimated expected utilities $\hat{\psi}_D(d)$ will be stored in the Results tab of the [D] node. We obtain the same results than in Table 3 of deliverable *D5.1 - Basic Models for Security Risk Analysis*.
By observing the influence diagram, note that in order to solve her decision problem, the Defender has already assessed $p_D(S|d, a)$ and $u_D(d, S)$, but she also needs $p_D(A \mid d)$, which is her assessment of the probability that the Attacker will choose attack *a*, after observing that the Defender has chosen defense *d*. This assessment requires the Defender to analyze the problem from the Attacker's perspective, possibly as we describe.
First, the Defender must place herself in the Attacker's shoes, and consider his decision problem. Figure 8 represents the Attacker's problem, as seen by the Defender.

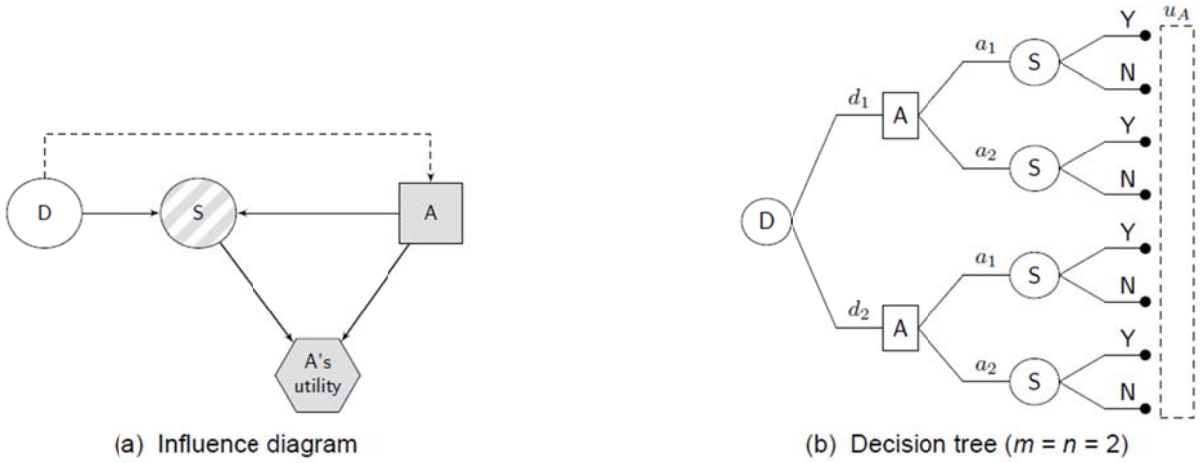


Figure 8: The Defender's analysis of the Attacker's problem

In this model, the Attacker observes the (random for him) defense of the Defender, and chooses an attack. To solve this problem, the Defender can proceed as follows:

- She elicits a distribution $F \sim (U_A, P_A)$.
- This induces a probability distribution

$$\Psi_A(d,a) = \sum_{s \in \mathcal{S}} \underbrace{U_A(a,s)}_{u_A} \underbrace{P_A(S|d,a)}_{S}$$

  This random expected utility is, for each (d, a), a weighted average of the random utilities $U_A(a, s)$ over all possible outcomes of the attacks $s \in S, p_D(S|d,a)$.

- Then, the Defender can estimate probabilities $p_D(A|d)$ through MC as

$$\underbrace{\hat{p}_D(A = a|d)}_{A} = \frac{\#\{a = \arg\max_{x \in \mathcal{A}} \psi_A^k(d,x)\}}{N}, \quad \forall a \in \mathcal{A},$$

  where $\{\psi_A^k\}_{k=1}^N \sim \Psi_A$

We could try to solve this problem with GeNIe, as shown in Figure 9, proceeding as follows:

- $p_D(d)$ should be placed on node (D);
- $U_A(a, s)$ should be placed on node (u_A);
- $P_A(S \mid d, a)$ should be placed on node (S);

and (if we were able to solve the problem) the estimated probabilities $\hat{p}_D(A|d)$ (unknown for her) would be stored in the Results tab within the [A] node. But the problem

here is that GeNIe is not able to handle continuous probability distributions (as $U_A(a, s)$ and $P_A(S \mid d, a)$). So this problem should be solved with a different tool e.g. OpenBUGS or Matlab.

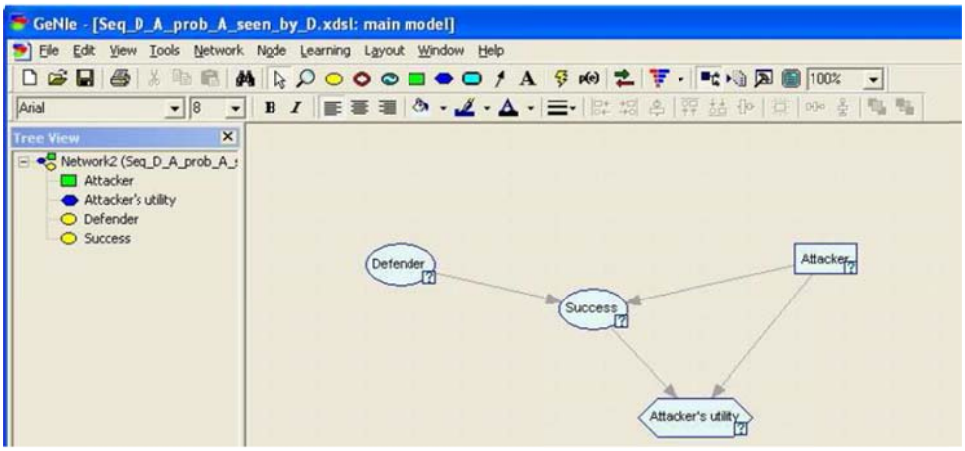


Figure 9: GeNIe model for the Defender's analysis of the Attacker's problem

Provided we have been able to solve the Attacker's problem with any of these tools (i.e., once with $p_D(A|d)$), we could go back to the model in Figure 6, and try to solve the Defender's problem with GeNIe, see Figure 7. Her expected utilities at node Ⓐ in Figure 6 for each $(d, a) \in D \times A$ are

$$\psi_D(d, a) = \sum_{s \in S} u_D(d, s) p_D(S|d, a).$$

For each $(d, a), \psi_D(d, a)$ is a weighted average of the utilities $u_D(d, s)$ over all possible outcomes of the attacks $s \in S, p_D(S|d, a)$. Then, her estimated expected utilities at node [D] for each $d \in D$ are

$$\hat{\psi}_D(d) = \sum_{j=1}^{n} \psi_D(d, a_i)\, \hat{p}_D(A = a_i|d).$$

For each $d, \hat{\psi}_D(d)$ is a weighted average of the expected utilities $\psi_D(d, a)$ over the estimated probabilities $\hat{p}_D(A = a_i|d)$. They will be stored, as previously mentioned, in the Results tab in the [D] node of the GeNIe in Figure 7.
Finally, her optimal decision would be $d^* = argmax_{d \in D} \hat{\psi}_D(d)$. The above discussion illustrates the inability of GeNIe to handle even the simple ARA models.

2.4 Bugs

This chapter first we want to describe the Bugs tool, which is another tool with good ideas, and then explain why it is inappropriate for this project to use.

2.4.1 Description

Bugs [14] is a statistical software for modeling **B**ayesian inference **u**sing the **G**ibbs **S**ampling algorithm. The user can define an almost arbitrarily complex statistical model. Therefore it is only necessary to declare the dependencies between related variables. The great advantage of applying the Gibbs Sampling algorithm is in fact the possibility to compute the combined distribution of a statistical model, even when only the separate distributions of the variables are known.

There are two versions of Bugs originated from the same source: OpenBugs and WinBugs. WinBugs is an established and stable version, but it will not be further developed. On the other hand, OpenBugs is open source software, which is at least as reliable as WinBugs in the newest versions, and the continuing development work is still in progress. A typical example visualized with OpenBugs is visualized in Figure 10.

One could possibly deduce a strategy to implement some part of the template ARA models within OpenBugs/WinBugs. Indeed, we could use it to create the Attacker problem with uncertainty in the assessments and use a simulation strategy to obtain the distribution over Attacker policies. We would then need to externally develop the Defender model with the produced distribution as key input. This would be quite cumbersome, as an approach to general security risk models.

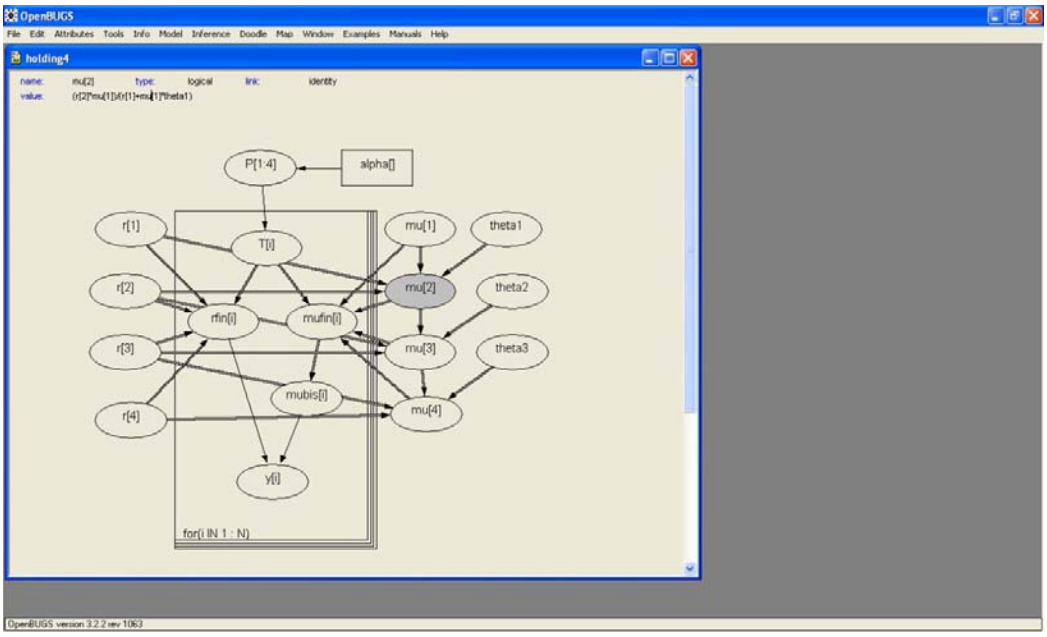


Figure 10: Snapshot of OpenBugs example

2.4.2 Inability of OpenBUGS to solve the SEQ-DA problem

In order to solve the Attacker's decision problem, we need to implement the first step of the algorithm on p. 20 of deliverable *D5.1 - Basic Models for Security Risk Analysis* for each possible defense.

Simulation for the Sequential Defend-Attack problem

1. Estimate $\hat{p}_D(a_j|d_i)$ for each d_i

For $i = 1$ to m

  For $k = 1$ to N

    For $j = 1$ to n

      Draw $(u_A^k, p_A^k) \sim (U_A, P_A) = F$ for all possible outcomes $s \in \mathcal{S}$

      Compute $\psi_A^k(d_i, a_j) = \sum_{s \in \mathcal{S}} p_A^k(s|d_i, a_j)\, u_A^k(a_j, s)$

    Compute $a^* = \arg\max_{x \in \mathcal{A}} \psi_A^k(d_i, x)$

    $\hat{p}_D(a^*|d_i) = \hat{p}_D(a^*|d_i) + 1$

  $\hat{p}_D(a_j|d_i) = \hat{p}_D(a_j|d_i)/N, \ \forall j$

2. Compute optimal defence

For $i = 1$ to m

  Compute $\psi_D(d_i) = \sum_{j=1}^{n} \hat{p}_D(a_j|d_i) \left[\sum_{s \in \mathcal{S}} p_D(s|d_i, a_j)\right] u_D(d_i, s)$

Compute $d^* = \arg\max \psi_D(d_i)$

The layout of the model in OpenBUGS would be as shown in Figure 11.

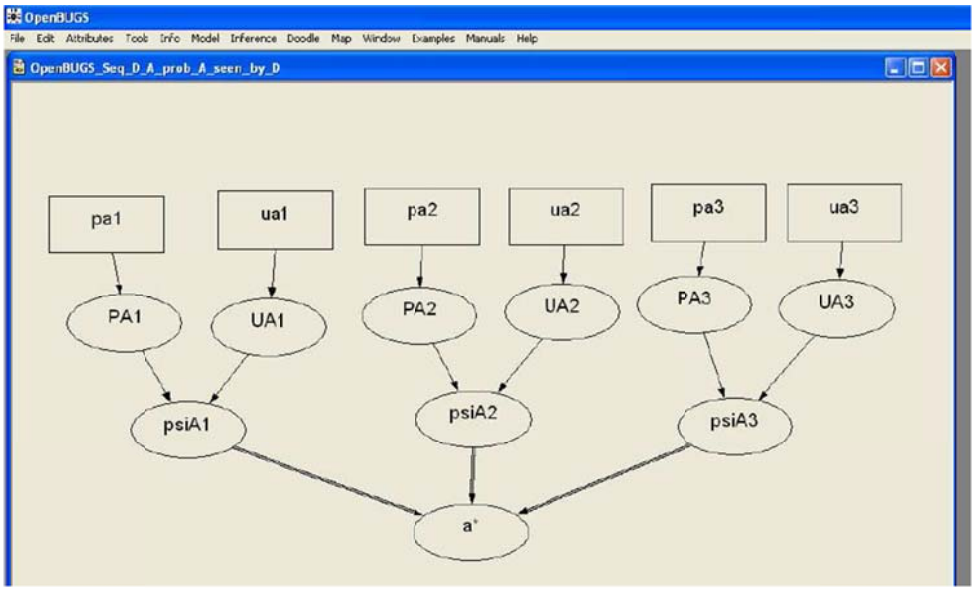


Figure 11: OpenBUGS model for the Defender' decision problem

On the upper level of the model we should have to specify the parameters of the involved distributions, as can be seen on Table 1 of deliverable *D5.1 - Basic Models for Security Risk Analysis*. Then, after sampling from the distributions of the probabilities $P_A(s|d, a)$ and the utilities $U_A(a, s)$, we would obtain the expected utilities $\psi_A(d, a)$, from which we could finally estimate the optimal attack from the point of view of the Attacker.

The drawback is that, even for this small problem, the specification of the model becomes cumbersome in OpenBUGS, and there is a lack of flexibility in the definition of the distributions and of the quantities involved. Besides, a different model should be defined for each possible defense, with the additional limitation that the models are not easy connectable in OpenBUGS. Hence, OpenBUGS is not an adequate tool to deal with our ARA analysis.

2.5 SecInvest

SecInvest is a security investment tool that emulates the presence of a security expert. The tool is described in detail in [3], [4] and [5]. It takes decision makers through the evaluation of investment alternatives in a step-by-step manner, without requiring the decision maker to be an expert. SecInvest does this with the help of a number of knowledge and experience repositories, both can be company confidential and publicly available. The public repositories are made up of information from sources like open vulnerability websites and risk analysis report providers (e.g., NIST and ENISA). The repositories also incorporate vendor-specific exploit and vulnerability information. To capture regional aspects like country-specific threat situations, that may affect the investment initiative, SecInvest includes an additional regional risk repository.

SecInvest uses a trust-based information aggregation technique to combine the disparate information and help select and link information of relevance for a particular investment decision. The tool also takes into account, whether the decision maker is risk-averse, risk-taking or in-between, and lets the decision maker actively take part in the investment alternative evaluation process.

The development of this new process of decision making becomes necessary, although there are a variety of already existing economic approaches like Return on Security Investment (RoSI), Net Present Value (NPV) and Annualized Loss Expectancy (ALE) models. Each of these approaches uses a specific form of cost-benefit analysis. As these components are often too difficult to be estimated exactly, it is in practice rarely achievable to set up a budget decision making process that rests solely on results of these purely rational and economic models. Therefore, some more categories of affecting parameters have been integrated into the decision making process.

There are four categories of variables involved in evaluating security investments alternatives, based on fitness score: (a) Cost variables, (b) Risk variables, (c) Context variables, and (d) Benefit variables. In addition, there are priority-variables modeled as utility functions across the other variable sets. The cost category includes the variables: (a1) Monetary cost, (a2) Billing model, and (a3) Cost coverage. In SecInvest these three variables are defined in terms of a qualitative relational scale and all are ranked internally and in respect to the other cost variables using conditional probability expressions. The same applies to the risk variables: (b1) New risks, (b2) Compliance, and (b3) Liability; the Context variables: (c1) Time-to-market (TTM), (c2) B2B trust (hereafter called Trust), (c3) Cultural issues; and the Benefit variables: (d1) Cost savings, and (d2) Control retained. The risk and context variables are used to compare alternatives from a security perspective, while the cost and benefit variables hold the financial and business constraints.

The SecInvest decision engine is implemented as a Bayesian Belief Network (BBN) topology. BBN is a powerful tool for reasoning under uncertainty and has shown effective for both assessing the safety and the security of systems.

SecInvest merely focuses on security evaluation and cost-benefit analysis and does not support a policy-driven security evaluation. Furthermore, SecInvest supports private and public medium sized businesses, and does not target critical infrastructure and national

security. Figure 12 shows a Bayesian Belief Network topology of the SecInvest decision engine.

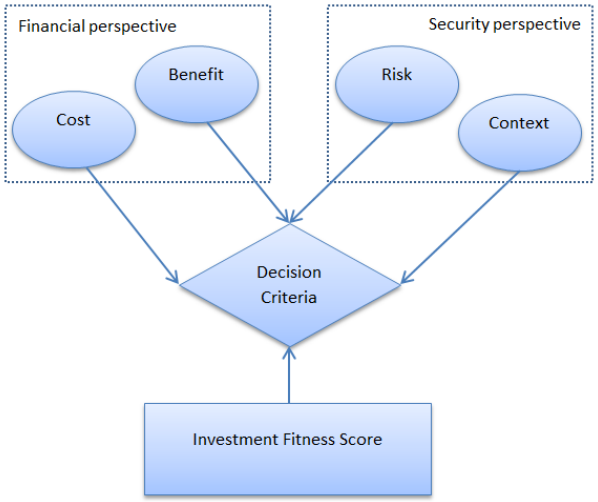


Figure 12: Bayesian Belief Network topology of the SecInvest decision engine

2.6 SeCMER

The SeCMER tool, described in [2], is specialized on the qualitative management of structured security requirements. This becomes necessary as modern software systems are increasingly complex, and the environments where they operate are increasingly dynamic. Security requirements change continuously, making the traceability of requirements difficult and monitoring of requirements unreliable. Changing requirements can also have impact on security properties of a system. Some older properties can become invalid, or new properties have to be satisfied, due to system changes. Therefore, it is necessary to apply analysis techniques that check the satisfaction of requirements in a system that has been changed.

SeCMER supports the automatic detection of changes in requirements and violation of security properties. The tool also identifies new security properties, due to old properties being affected by evolutions of the system. SeCMER provides a methodology that supports a conceptual model of security requirements, with a process for the elicitation of security goals, a light-weighted approach to formalizing, and reasoning about changing security goals, and an approach based on argumentation, and model transformation to reason about the impact of change. The tool's output either ensures that the evolution of a system did not affect any security properties, or it offers modified security properties that have to be applied by the changed system.

For modeling requirement changes, SeCMER produces two models: The before model and the after model. These models link the empirical security knowledge such as information about assets, security goals and threats to the stakeholders' security goals. Therefore, concepts from Problem Frames (PF) and SI* requirements technology are combined with traditional security concepts, such as security goal and asset. SI* is an extension of the I* framework that provides the ability to model a project's stakeholders, their goals and their social inter-dependencies, whereas PF analyzes the physical domains acting in the problem's context. There is certain similarity between problem concepts modeled with

SI* and PF, especially within the notion of, for example, biddable domain on the one hand, and the actors on the other hand.

Merging SI* and PF produces a quite powerful security goals engineering approach, as an SI* analysis can identify attacker's intentions through the social interactions of the involved participants. PF instead allows identifying valuable assets that lie in the system boundary through explicit traceability of shared phenomena among physical domains and the machine itself. The concept of SeCMER is visualized in the model shown in Figure 13.

To detect changes affecting security properties in a system, the SeCMER methodology makes use of security principles, which are specified by an extensible set of security patterns. Security patterns define situations that cause violation of a security property in the system. Whenever the application finds one of these patterns in the model, it can automatically be reported. It is also possible to apply pre-defined default actions automatically when a security property is violated, or at least to suggest some quick fix solutions when there is no automatic remedy defined.

In the following, some default security patterns and properties are introduced. The main security patterns, the SeCMER methodology makes use of, are trust, access and need.

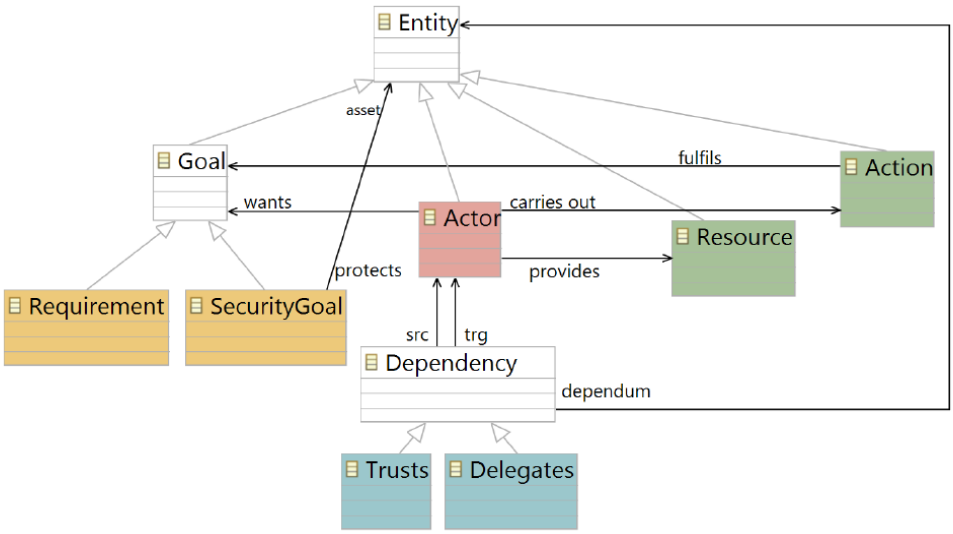


Figure 13: Conceptual model of SeCMER

Trust and access are patterns, which can also be modeled and interpreted transitively. In the patterns, only assets that are protected by security goals are considered. When an actor performs an action that does not violate any system properties, there is no need for further investigations. If an actor gets access to an asset without trust, then the "trusted path property" is violated. If an actor gets access to an asset without need, then the "least privilege property" is violated.

Security analysis in SeCMER can also be driven argumentation based. As a result of changes in the requirements model, the developers check whether there are new security properties to be added or to be removed. For argumentation based analysis, the argumentation meta-model is deployed. This is shown in Figure 14.

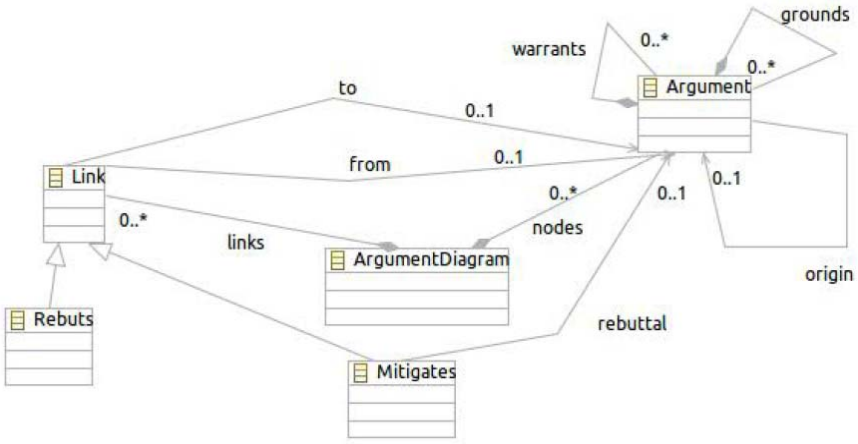


Figure 14: SeCMER arguments meta-model

An argument diagram consists of multiple arguments linked together. Each of these arguments has exactly one claim. It also consists of facts and warrants. A claim is a predicate with a truth value that is established by an argument. A fact is a true proposition, represented by an argument that only contains a claim. A warrant links the facts of an argument to its claim. As facts and warrants can themselves also be arguments, arguments can be nested. When an argument is introduced to the model, it is possible to indicate the time when it is introduced by means of an optional timestamp.

Argumentations should not be re-executed after every small change in the requirements model. Whenever some arguments may become invalidated by a system change, then the argument is marked for re-examination. This relies on the traceability that can be established between the argument and the requirement model.

2.7 Matlab Overview

The Matlab programming language offers an excellent environment for developing models that mix economic models of interacting agents (distributed or representative) with mathematically consistent models of the architecture of a system with security requirements.

The Matlab programming language has been developed over a number of years to serve as a means for accessing computationally intensive tasks written in languages such as C, C++, VBA, Java and Fortran.

From the Matlab Product Description [18] the key features are

- High-level language for technical computing
- Development environment for managing code, files, and data
- Interactive tools for iterative exploration, design, and problem solving
- Mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, and numerical integration
- 2-D and 3-D graphics functions for visualizing data
- Tools for building custom graphical user interfaces

- Functions for integrating Matlab based algorithms with external applications and languages, such as C, C++, Fortran, Java, COM, and Microsoft Excel.

The usefulness of the language is in serving as an integrator for models developed in the languages listed in the key features. For deployment purposes the Matlab programming language offers three very helpful major features.

1. Proof of concept and robustness checks.
2. Development of graphical user interfaces.
3. Compilation and deployment.

The first feature is reviewed in deliverable D6.1. Most mathematical formulations can be described by meta-code and in most circumstances the expressiveness of basic Matlab code replicates the meta-code with only minor adjustments.

The second feature is useful in deployment and is related to future work to be found in Deliverable D8.4. Graphical User Interfaces (GUIs) allow for easy interaction with models, allowing users to change parameter configurations and run experiments for different types of policy.

The third feature is very important for distribution of the policy modeling tool that the SECONOMICS project will produce through WP8. The Matlab compiler allows the modeler to build standalone executable applications that can then be installed on machines with appropriate operating systems.

From the Matlab Compiler product description [19]:

"Matlab Compiler lets you share your Matlab application as an executable or a shared library. Executables and libraries created with the Matlab Compiler product use a runtime engine called the Matlab Compiler Runtime (MCR). The MCR is provided with Matlab Compiler for distribution with your application and can be deployed royalty-free."

The features of Matlab Compiler will be discussed in detail in section 3.1.4.

2.8 Carisma

Carisma is a framework for compliance, risk and security analysis in models. This section about the framework is based on [1]. Carisma is developed at Fraunhofer ISST and TU Dortmund, and it has been used there successfully for a few years in several projects. It is based on the Eclipse platform and therefore has an extendable open plugin-architecture. Carisma is not bound to any specific model type, but can easily be extended to the support of any possible model type. For now, it supports the Eclipse Modeling Framework (EMF), UML models and BPMN models. Furthermore, Carisma offers extension points facilitating the contribution of functionality of other plugins to provide different checks. Such a Carisma check is the implementation of checking a security property. A check can also contain miscellaneous help functionalities. A Carisma analysis is a set of Carisma checks that are applied to a model in a pre-defined order. As Carisma is provided as an open plugin-architecture, additional analysis routines can be integrated into

the framework easily. Compared to other tools, Carisma supports more types of models and offers a broader range of analyses.

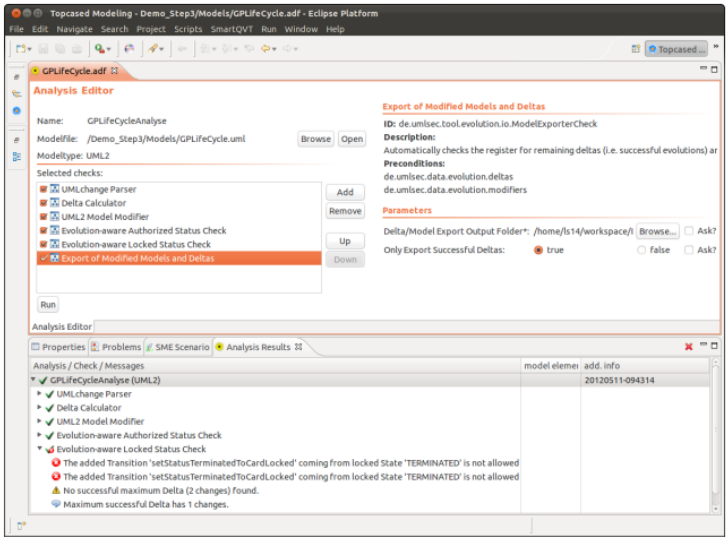


Figure 15: The user interface of Carisma

It is important to mention, that it is also possible to analyze only the evolution of a model that has been changed. So no complete re-verification is necessary after a model has been re-generated. To perform the analysis of a model in evolution, it is necessary to compute the difference between the original and the changed model. This difference is called the delta model.
Figure 15 shows the user interface of Carisma. It provides different views to create and execute analyses, and to display the corresponding results. On the top left side in the Analysis Editor, the user has to choose a model that has to be analyzed, and also the checks that are supposed to be applied to the model in the analysis. It is possible that a check requires certain parameters for its execution. These must be either primitive data types such as String, Integer, Float or Boolean or can also be files or folders. Executing the analysis by clicking the Run button initiates that Carisma loads the specified model and triggers all checks contained in the selected analysis routines. The results are shown in the Analysis Result View at the bottom of the figure. All performed checks are grouped by the analyses they belong to. For each check a key like Success or Failure shows the check result. Furthermore, detailed information about each check execution is given. Afterwards, a full textual report, that contains all information about the analysis, is generated. The information consists of detailed test results, but can also contain descriptions of scenarios that would fulfill security requirements in case of an actual check failure.

Certain checks support the analysis of evolution models that are changed. The process of this analysis is visualized by the pipeline in Figure 16. The steps always have to be processed in the order that is given there. As the first step, the changes in a model have to be identified. This can either be done by computing the differences – which will be

called the delta model – or by manual user-entered specifications of all changes. The second step of the analysis is the computation of sets of all possible delta models from the given changes. When one or more security requirements are violated by the evolution, all partial applications of changes in the model are analyzed. This is done with the help of the sets that have been computed before. In the next step, the security checks are performed actually either on the delta models, or the modified model itself. All valid delta models can be stored afterwards.

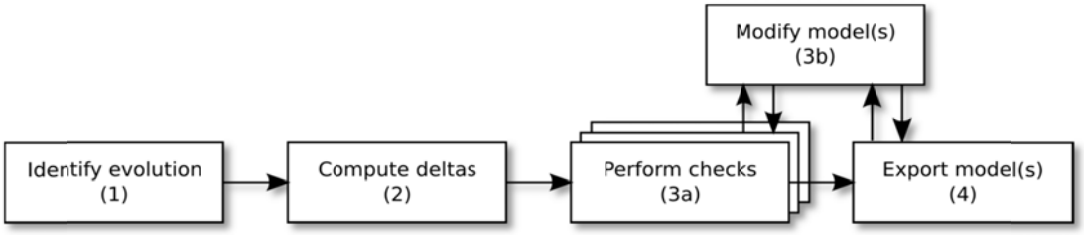


Figure 16: Evolution analysis pipeline

The tool architecture of Carisma is visualized in Figure 17. Carisma is designed as an open plugin architecture that can easily be extended. It is available as both RCP standalone application and also integrated into Eclipse. Therefore, the Carisma tool is divided into several Eclipse features, which can be installed altogether, but also separately. The framework and the core components of Carisma are independent from the model type that has to be processed. On the other hand, Carisma checks are mostly model type specific. For example, checks using UMLsec stereotypes can only be applied to UML models. But the realization of model type independent checks is also possible. The model processed in a check or analysis is accessed via the Eclipse Modeling Framework (EMF).

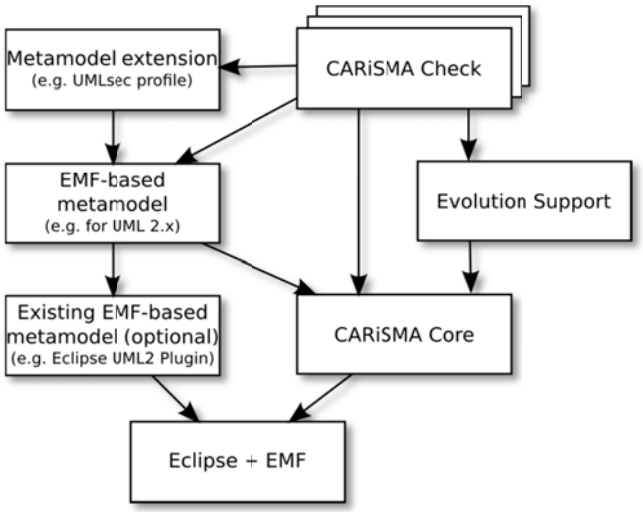


Figure 17: Carisma Tool Architecture

For integrating new checks into Carisma, there are two things necessary: First, meta-information about the check, such as name, identifier and supported model types have

to be declared. As the second step, the check itself has to be realized. Therefore, the CarismaCheck interface has to be implemented. The central operation is called “perform” and contains the actual model analysis of a check. Its return value must be a Boolean, which indicates whether the check was executed successful, or resulted in failure. As a help feature, Carisma also offers a wizard that supports the contribution of new checks by creating a template-plugin project and necessary meta-information.

3 Techniques and possibilities of the tool platform

In the following paragraphs, some important techniques that will be used in the implementation of the toolbox are described. The evaluation of different techniques to gain access from Java to Matlab is based on [6].

3.1 Matlab Java connection

The ability to evaluate Matlab functions, models and simulations from a Java environment are essential for an integrating framework, which has several possibilities to visualize and design editors for an easy usability of the analytic models developed in WP5 and WP6. In principle, there are two fundamentally different options. The first one is that the integration platform can run and remote control a Matlab session, whereas the second alternative is that the code can be transformed in Java code. The first method can be realized with the help of the Java Matlab Interface (abbreviated as JMI) and the Java remote method invocation RMI. A tool which combines them, called "matlabcontrol", was found in [6]. The second method makes use of the Matlab Builder JA. It "compiles" the Matlab code to Java executive code which can directly be called from Java but needs to have a runtime environment installed. The two methods will be now reviewed in more detail.

3.1.1 JMI

The Java package `com.mathworks.jmi` is provided by Mathworks who are the developers of Matlab to JAVA. With the help of this package, a Matlab session can be remote controlled from a Java program. There are different classes and methods included in the package that allow the control of the session. The central class, which is the most important one for the developer, is the `Matlab` class. Official documentations on this package do not exist, but help can be accessed via the Matlab command line prompt for information about the methods of the `Matlab` class. The command `methods com.mathworks.jmi.Matlab` provides a list of names of all methods contained in the class. For obtaining further information on the methods, the command `methodview com.mathworks.jmi` can be entered in the Matlab command line. Then, for each method, the return value, the required arguments and possible exceptions are specified. The most important methods are `mtEval` and `mtFeval`. The method `mtEval` receives a string as a parameter that is sent directly to Matlab and evaluated there. The string and the result of the analysis are displayed in the Matlab console, as if they had been typed in there. With the method `mtFeval` the functions are passed separately with name and parameters to Matlab: `mtFeval (name, param1, param2, ...)`. These functions can also return values back to Java. The package `com.mathworks.jmi` only serves to interact between Java and Matlab with an existing connection between Java and the Matlab session. To prepare such a connection more packages are needed.

3.1.2 RMI - Remote Method Invocation

The RMI package is an interface that allows connections between different virtual Java machines running different Java-based applications. Furthermore, RMI defines its own client-server-based communication protocol with which these connections are realized. The Java Virtual Machine running Matlab corresponds to the server, to which the client virtual machine running the framework connects. First, an interface is created, which defines the methods offered by the server. This interface is then implemented by the

server. Objects and methods provided by the server have to be registered in a RMI registry. The client can then access these objects and methods with the help of the registration name.

3.1.3 Matlabcontrol

Matlabcontrol is a well-documented [8] package that allows a connection from Java to Matlab, using JMI and RMI. This package is an open source project under BSD license, which is not directly supported by Mathworks. The program code, as well as a simple example application, is available online [8]. First, `MatlabProxyFactory` is used to create a proxy server for Matlab, which starts a new session. With the method `eval` commands can be sent to this proxy.

This method is exactly the same as `mtEval` from the JMI package. The input string is evaluated in Matlab and the result is returned to Java. The counterpart to the JMI method `mtFeval` in Matlabcontrol is `feval`. With it, the name of the Matlab function and the corresponding arguments are passed separately. In addition, with Matlabcontrol the values of variables can be set and read. The following example is from the online walkthrough of Matlabcontrol taken from [8].

The method `setVariable ("a", 5)` creates a new variable in Matlab and assigns a value of 5 to it. With `getVariable ("a")` the contents of the variable can be retrieved from Matlab. It must be noted that return values of Matlab are always returned in form of arrays, even if they contain only one element. Calling `Object result = proxy.getVariable ("a")` thus provides not the desired result. For the correct result a type conversion has to be done. The corresponding call for this is `double result = ((double []) proxy.getVariable ("a")) [0]`. Below the relationship between the different components in the Java-Matlab-remote control is illustrated schematically in Figure 18.

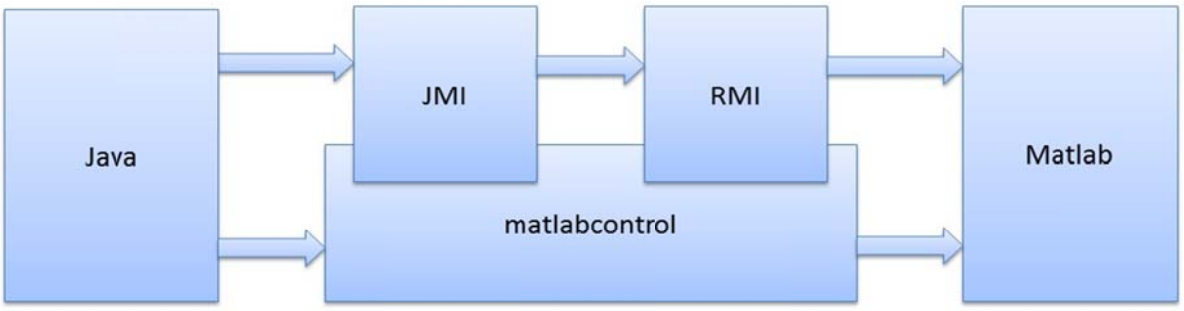


Figure 18: Remote control a Matlab session with "matlabcontrol"

3.1.4 Matlab Builder

Matlab Builder is a tool included in Matlab that offers the ability to use Matlab code in other applications outside Matlab. Matlab Builder JA covers Matlab Code with additional Java code and embeds it into Java classes. These classes can be accessed via predefined methods whereas the Matlab code itself is encrypted. The advantage of using

Matlab Builder JA is that no Matlab distribution is needed to execute Matlab functions. Java Packages created by Matlab Builder can be executed on every machine – only the Matlab Compiler Runtime has to be installed to use the functions. This runtime can be obtained without charge on the Mathworks website.

The Matlab Compiler Runtime also includes a graphical output window for visualizing graph plots, but there the functionality for handling these plots is limited to a certain degree.
To use the package created by the Builder in a Java application, it must be imported like any other Java package. Furthermore, an additional Builder Java package that comes along with Matlab has to be integrated into the Java application.

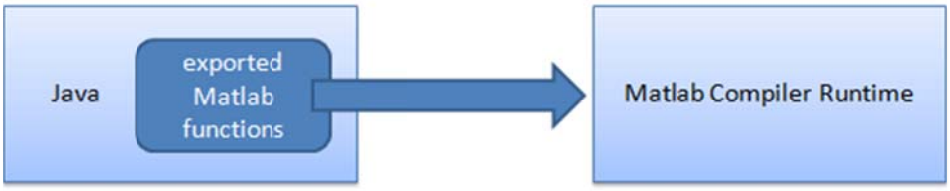


Figure 19: Execution of exported functions in Matlab Compiler Runtime

Figure 19 visualizes that the exported Matlab functions integrated into the Java environment call the Matlab Compiler Runtime. The function execution is processed inside the runtime. There is no possibility to gain direct access to anything that is processed in the runtime. Only the results of the computation can be handled in Java, after the execution has finished. A detailed instruction manual for Matlab Builder JA can be found in [9].

3.2 Eclipse

Eclipse is an open source development tool for various types of software. Being an integrated development environment (IDE) only for Java language in the early years, the use of Eclipse in other fields of action has increased more and more because of its extendibility. Eclipse itself is based upon Java technology, since release 3.0 on the OSGi framework called Equinox, which makes it available for multi-platform use – Eclipse runs on Windows as well as on Linux systems.
Eclipse is also almost completely based on plugin architecture. Only a little part is provided as the core, which can be extended to a high degree as there are a lot of plugins for Eclipse, both open source and commercially provided. It is easy to embed any Java application into the Eclipse environment – everything that runs in Java can also be included in Eclipse.

Because of its free availability, Eclipse is a widely spread development environment in computer science and so offers a bunch of substantial manuals and help documents, which abundantly ease the first steps inside Eclipse for new beginners. A lot of basic tools, such as different types of editors, are already available as plugins for Eclipse, but if this is not enough, it is also easy to create new tools, for example by means of the Graphiti toolkit, which will be described in detail in sections 3.4.

The Carisma tool is also implemented in Eclipse, which makes it a regularly-used environment and so there is a large expertise among the workgroup members at Fraunhofer ISST.

The layout of the Eclipse window is fully customizable; any part of any view can be moved and docked almost arbitrarily and can be aligned to every user's individual requirements.

3.3 Plugin Interface

Plugins are self-contained software components which can be created in Java. Eclipse itself is built up from a plugin-architecture. The main IDE can be extended to additional menus and editors by the use of plugins.
A plugin is a set of code which is put in a modular, extendable and distributable package. Modular means in this case, that some of the code is already provided. Moreover, it is specified whether there are dependencies to other packages or plugins, and also which packages are from the newly created plugin. A Java application can be composed of multiple plugins which can be added, replaced or deleted to modify its functionalities.
A plugin is provided extendable, as so-called extension points can be used by other plugins to extend them to further functionalities. Of course, it is also possible to define own extension points, so that other plugins can extend the own features. Extension points are typically defined as a combination of XML and Java interfaces. A plugin using an extension point has to stick to a pre-defined structure exactly. The definition of an extension point is some kind of agreement between the two components.
The distribution of plugins is realized by an export to a folder or a jar file. These data can be used by other applications instantly. Moreover, there is a possibility to merge multiple plugins to a so-called feature. These can be installed to the application entirely.
To add a new extension point to a plugin, there are a few steps necessary. At first, the extension point has to be defined in the manifest file. The definition's content states the name of the extension point, its unique ID which the extension point can be referenced at by other plugins, and its internal structure that has to be inherited by plugins. This structure consists of multiple so-called elements. These are defined by a name and a data type. When a plugin is supposed to extend any functionalitiy, at least one element of the type Java must exist. This element describes an interface which is implemented by the plugin. The interface's name is also registered in the element. By the use of these information from the manifest file, the structure of the extension point can be identified, and the user gets to know which classes have to be implemented to extend the target application.
To use plugins that extend one's own application the application itself is responsible for that. In an Eclipse registry file, all plugins which implement the new plugin's own interfaces can be found. These can be then instantiated and method calls are possible.
When writing a new plugin the first step is to create a manifest file in this new plugin. Therein, the user adds information about the extension point the plugin makes use of. Elements can be attached to these extension points, whereas the elements contain specific classes which implement the interface of the element from the target application. Figure 20 shows this relation in a simplified way.

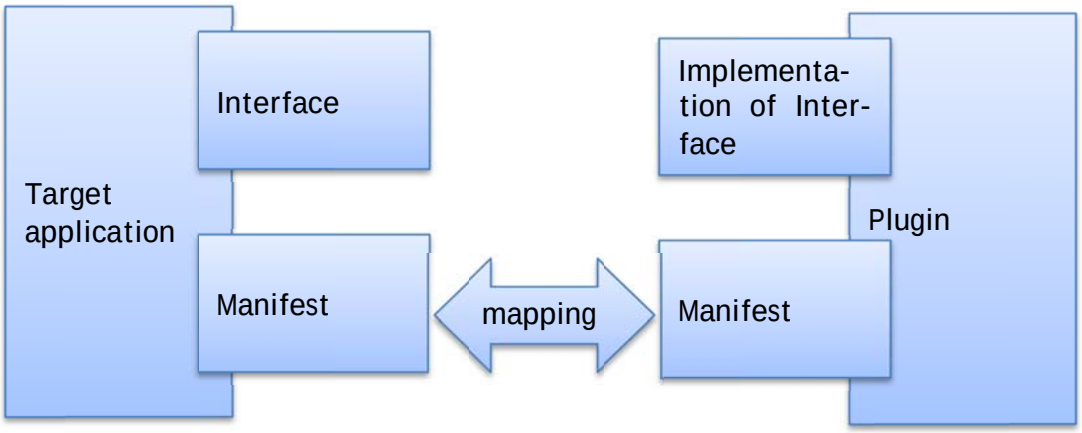


Figure 20: Relations between concrete classes and the Eclipse Plugin interface

3.4 Creation of graphical editors

One part of the Eclipse Framework is the Eclipse Modeling Framework (EMF). It is a large-sized package of different tool packages. One major part of EMF is the Graphiti package. It is used to build graphical editors to create, edit and visualize models, like BPMN or UML-diagrams. In this project, Graphiti provides a Java API to create visual editors for modeling influence diagrams or decision trees. With this package, it is easy to build such visual modeling tools. Using a visual editor, it is easy to create models within a graphical user interface. Such created models can then be post processed and then pushed to the Matlab engine to be processed by the mathematical toolboxes there. The result can be returned to the framework and then visualized in the visual model. The Graphiti package uses domain-models to know the structure of the creatable models. These domain-models are described in a special language introduced in the Eclipse Framework called ECore.

To understand the context of Graphiti it is necessary to get some background information. This includes the Eclipse Modeling Framework (EMF), which is described in section 3.4.1, and the ECore meta-model, described in 3.4.2.

3.4.1 Eclipse Modeling Framework

The Eclipse Modeling Framework (EMF) is an open source project of the Eclipse open source community. It is, for instance, used to create Java code based on a structured model. EMF uses the XML Metadata Interchange format (XMI) to define these models. The domain-model of EMF precisely defines the various components a specific instance of the model may consist of. These components may be, for example, classes, data types, class attributes, or associations between individual elements of the model. Since the purpose of this framework is to generate Java code, the scope of the modeling capabilities is not as comprehensive as e.g. UML. But the previously discussed components are sufficient in most cases. There are several ways to create such a model to produce code using the EMF code generator therefrom. The most basic method is to define the model by hand in an XML or text editor, where care must be taken on the precise structure of the document. Another option is to use existing modeling tools, such as Rational

Rose, which also uses the XMI format to define their charts. Additionally, there is the possibility of providing ad hoc written Java interfaces to be provided with modeling properties. If these interfaces are annotated on java classes, the EMF can evaluate this and produce the missing implementations.

The generated code can then be extended with instance variables and methods by the user. In addition to the interfaces and implementations, the code generator of EMF creates getter and setter methods and ensures the validity of two-way references. For example, the two classes "book" and "Author" refer to each other, so the EMF makes sure, that if an author is assigned to a book by the user the book is assigned to the author, too.

In addition to model interfaces and implementations, two more interfaces and implementations are generated: Factory and package. Factory contains methods to create instances of the domain-model. Therefore, a create method for every class in the domain-model is provided. Package supports static data about the domain-model. Therein, access to static integer variables is possible, as well as access to methods which return the basic data types of the EMF meta-model, which have been used for the domain-model. This is, for instance, for the class “book” the type “EClass”. This information is needed to create specific instances of the domain-model. The generated classes use this information to identify objects in the domain-model.

3.4.2 ECore meta-model

ECore is a meta-model which the Eclipse Modeling Framework is built upon, thus ECore represents the core of this framework. The meta-model is also a toolbox for EMF, as the user can work with its pre-defined elements when creating new domain-models. The most important entities therein are classes (EClass), attributes (EAttribute), references (EReference) and data types (EDataType).

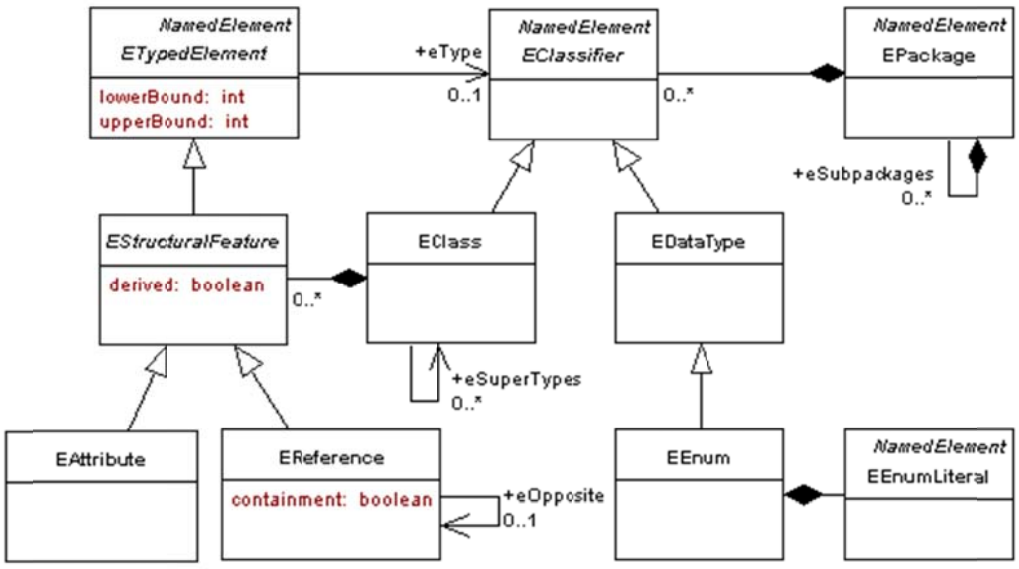


Figure 21: Scheme of ECore meta-model

The basic structure of the ECore meta-model is set up hierarchically and will be described below. On the top level there is a root object, which represents the whole model. There are packages attached as children to the root object. On the next level there are classes, whereas the bottom level contains attributes that are assigned to these classes. Figure 21 shows a simplified scheme of the ECore meta-model.

3.4.3 Graphiti Framework

The basis for any editor developed with the Graphiti framework is an existing domain model. This contains the data which has to be visualized. These domain models can either be models from the Eclipse Modeling Framework (EMF) such as the previously introduced ECore meta-model, but also domain-models from any other Java based objects, or meta-models in business process management (BPM) notation.

Models created with a Graphiti based editor always consist of two different types of elements: Objects and references. ECore already provides a meta-model with generic classes for these elements: EClass (for objects) and EReference (for references). For a newly instantiated domain model these classes can be subclassed several times, depending on how many different types of nodes and edges have to be provided in the new domain-model.

In the context of Graphiti the Link Model and the Pictogram Model are also important. The Pictogram Model contains all information about the graphical representation of a diagram created with a Graphiti based editor. As a consequence of this, some data might be redundant, as they are available both in the Domain Model and the Pictogram Model (e.g. the name of a class in an ECore model). This can sometimes cause synchronization problems. For this case, Graphiti offers a mechanism that visualizes these problems. These can be resolved manually by the user with an update feature.

Objects and references in the domain model have to provide the so-called features. Typical features are, for example, a create feature, to create elements in the domain-model; an add feature, to add the graphical representation of the element to the model; a delete feature, to delete an element from the visualization; and a remove feature, to remove an element from the model. These features have to be provided by the Graphiti user. It is very important that a create feature also has to call an add feature, to be able to link model and visualization with each other. This also applies to the remove feature, which must always contain a delete feature.

In Java code, features are implemented as Java classes extending applicable super classes. If desired, these classes can be extended to specific functionalities. It is also possible to define constraints within the model, for example only to interdict the creation of an edge between certain objects of a specific type. Therefore, it is necessary to provide a feature which only permits the specified connections.
All the features in the domain-model have to be added to the Feature Provider of the model. This central control class is used to call all the necessary features at any time. The feature concept is illustrated in Figure 22.

For each instance of an EClass or EReference all the provided features can either be implemented separately, or some features can be shared by two or more objects of the

domain-model. This can be useful, when there are only little differences between these objects, e.g. in their graphical representation.

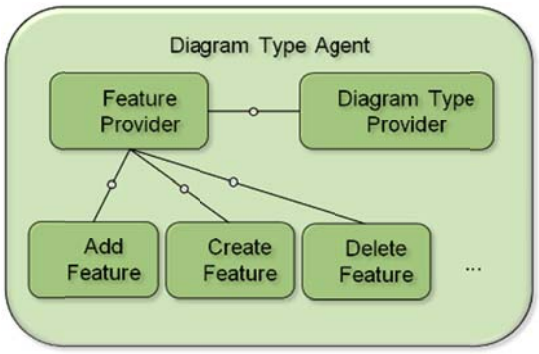


Figure 22: Feature concept of Graphiti

The graphical user interface of a Graphiti based model editor always includes a palette which contains the objects of the domain-model. From these objects the user can create specific models. To add a node object to an existing diagram in the editor, the user has to choose the object in the palette and then drag a new rectangle in the draw area of the diagram. Performing this action makes the Diagram Feature Provider class call the create feature of the object. This creates a new instance of the object in the model, adds its graphical representation to the diagram, and creates a link which contains the relation between these objects from domain- and diagram model.

4 Design of tool platform

In the following sections, the design of the tool platform will be presented, and a prototype of the user interface can be seen. After that, it will be shown how different models can be integrated.

4.1 Overview of the design

The toolbox integrates different separate parts. There are analysis models and supporting components like editors. The models are intended to be implemented as functions in Matlab, and can be used in this toolbox then. For the integration, the tool platform Carisma is used. This has already been used for many years at Fraunhofer ISST and TU Dortmund, and is characterized by many interfaces and thus good expandability. Generally, the tool should receive input data from the case studies WPs 1-3 and parameters of the models as input and generate analyses, graphs and reports from this data.

4.2 Design of the integrated Frontend

4.2.1 General

All Models will first be developed in Matlab and published as separate Matlab toolboxes. Some visualization will be implemented as far as possible. Additionally, these toolboxes can be integrated in one single platform built with Java. This enables the use of visual editors to create graphical models usable with the Matlab toolboxes. For example, there could be an editor for decision trees and influence diagrams. Additionally, this platform could be able to display some graphs and diagrams (with the help of the Matlab plotter).

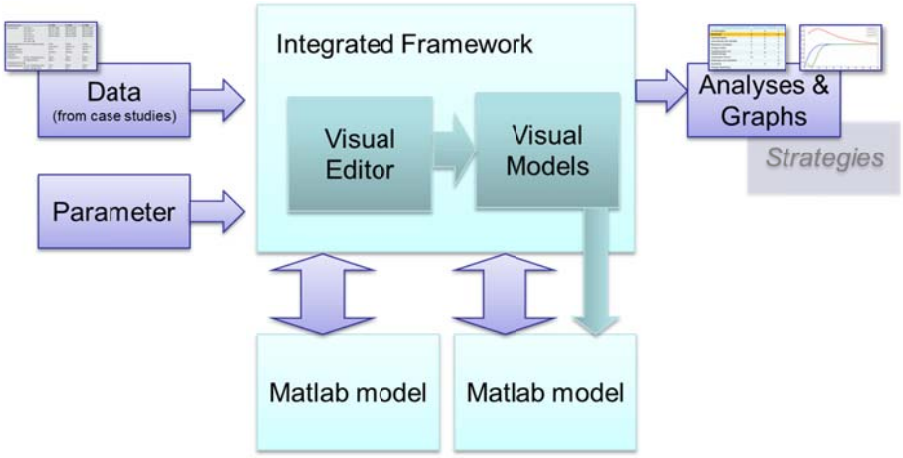


Figure 23: Schematic structure of the toolbox

During an evaluation, the analysis is selected. This analysis can then be adjusted with parameters and performed on the data provided to the tool. The Matlab implementations are previously compiled with the Matlab Builder JA, and can then be called with the aid of the Matlab runtime, to evaluate the analysis getting the data finally back from the runtime. For these calls no Matlab license is needed. Graphs and reports are then displayed. Figure 23 visualizes all these processes schematically. Data from case studies are provided to the tool, the model will be chosen, parameters are adjusted and with the aid of visual editors, some visual models might be designed. This data will be trans-

ferred to Matlab and the specific methods are called and the results are calculated in Matlab. Finally, the result is transferred back and displayed as tables, graphs and reports.

4.2.2 Integration in detail

We explain now in detail the structure of the framework, presented in Figure 24. From the case studies, empirical data is provided as external data to the tool box. As mentioned in section 3.3, the framework is able to be extended with additional plugins. The models themselves are provided as Matlab toolboxes. Each analysis will be displayed in an analysis selector, which is a list box icon showing all available analyses connected with a description of the analysis and its purpose. The analysis selector will provide two different interfaces to integrate a tool.

- The easiest one is to place an analysis description file into the specific toolbox directory. It has to be connected with either a compiled Matlab file, or even with an uncompiled Matlab file, which is then just able to be used together with a live Matlab installation running on the workstation. In this description file, parameters have to be defined, which can be accomplished with a default parameter editor, and will be passed to Matlab before analysis execution.
- The second possibility is to design specific parameter interfaces and other visual editors. These editors are built as Java Eclipse plugins and integrated into the framework. After filling in the Parameter the plugin itself decides which Matlab function should be called.

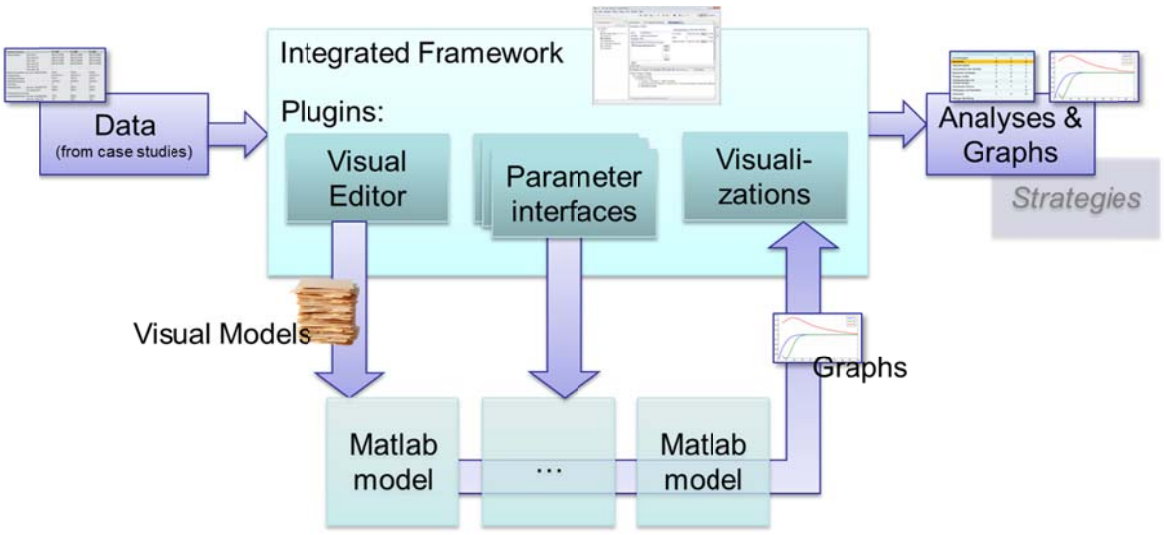


Figure 24: Concept of the tool framework with Matlab integration

There are other possible, much more complex, editors, which can build visual models like decision trees or influence diagrams. These can be transferred as files to the Matlab function. With this specific parameter interfaces, much more useful user interfaces can be designed, where some logic can be hidden behind the edit boxes to check the parameters or pre-calculate parameters. The visual editors are built-in with Graphiti. There are two possibilities to call Matlab functions, as described in Matlab Java connection in 3.1. The default, for most built-in analysis, will be to use compiled Matlab functions created with the Matlab Builder JA, because it does not need any Matlab license owned

by the toolbox user. After calculating the analysis the data and graphs can be returned to the framework.

4.2.3 Different parts of each model

As described in the project proposal, there are three different parts defined: the Security Problem Structurer, the Security Problem Modeler and the Security Problem Solver. Each model may consist of these parts, but not necessarily if, for example, no visualization is needed.

4.2.3.1 Security Problem Structurer

The Security Problem Structurer supports the modeler by adding specific values or parameters to a given model created and visualized by the Security Problem Modeler. Therefore, the Security Problem Structurer corresponds to the lowest layer in the overview of the Levels of abstraction.

4.2.3.2 Security Problem Modeler

The Security Problem Modeler is a model designer for a class of models, to create a specific instance of a model to match a problem. If possible, the designer is built as a visual editor. For example, for the adversarial risk analysis an influence diagram designer can be devised. Additional editors can be a utility function designer or decision tree visualization. To implement such visual designers, Graphiti will be used as explained in 3.4.

4.2.3.3 Security Problem Solver

The Security Problem Solver of each model is a calculation function of the model implemented in Matlab. These are included in the tool as explained in section 4.2.2. The solver therefore makes use of the Java/Matlab-Interface, as the interactions with the models are processed in Java.

4.2.4 User interface of the integrated tool box

All modules should be implemented into an integrated interface. At Fraunhofer ISST and TU Dortmund, Carisma is used as a tool platform. Carisma is based on Eclipse and provides large-scale extendable interfaces. All modules will be implemented for this platform as a tool plug-in. This creates an integrated impression while keeping components separated. The degree of integration depends, among other issues, on the form of input data that is provided by the case studies. It can be used just with quantitative statistics, or with statistics linked with structure, or process models. In the latter case, a tighter integration with the existing tool can be achieved and the representation of UML and BPMN is supported natively in the tool. For example, the topology of the Metro can be presented as a system diagram. The state of every module can be saved at (nearly) any time. Some templates could be possibly generated at later stages.

In Figure 25, the user interface of Carisma is shown. It is running a frontend for a Matlab implementation of Return on Security Investment as an integrated plugin. On the left side of the toolbox, a model can be selected from the choice of models. This choice is created by reading some configuration files. After selecting a model, the corresponding plugin is loaded and shown in the main frame. Each Model can have its own individualized parameter interface implemented as an own plugin and additional visual editors, or it can simply use a standard form for filling in some parameters. After loading data and

filling in the parameters, the form can be submitted and evaluated in Matlab. After that is finished, a result form is shown, displaying the graphs and analyses.

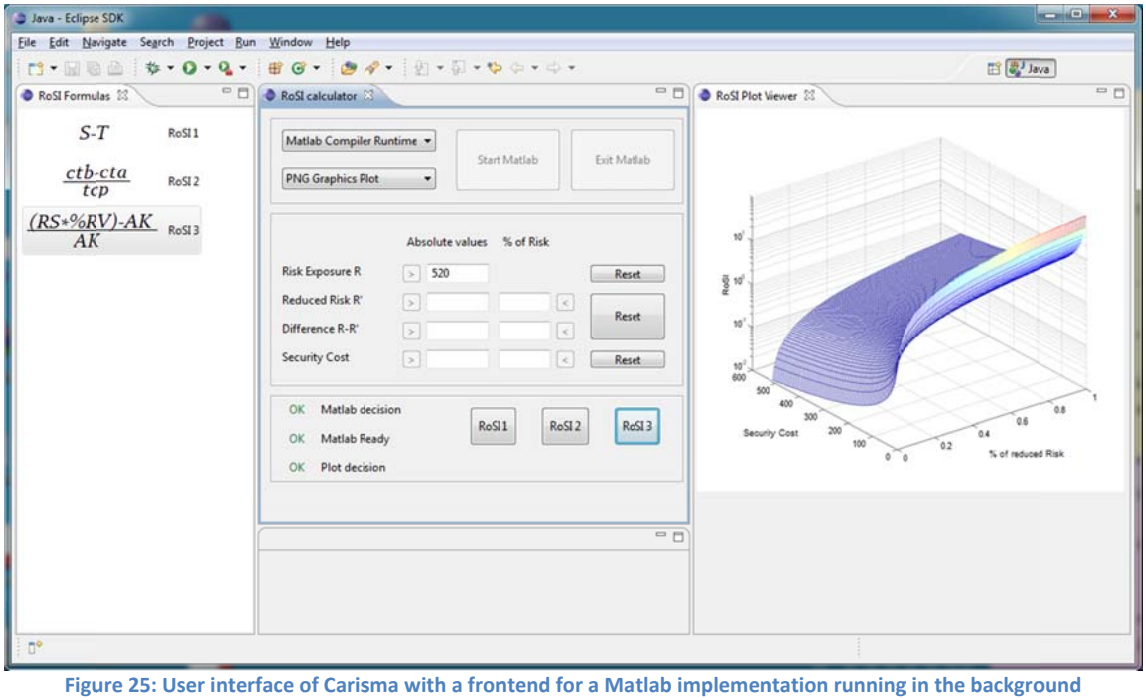


Figure 25: User interface of Carisma with a frontend for a Matlab implementation running in the background

4.2.5 Level of Abstraction

In order to make the tool usable for as many different kinds of applications as possible, it should also support different kinds of user. These users will have, in general, different knowledge and access to different information, something that should be taken into account by the tool. For this purpose, we have designed the concept of levels of abstraction. Each level needs different knowledge of the user and different information, too. To reach this goal, a level is an abstraction layer to the level below. Each level defines a specified amount of variables. A level has less possibilities of modification, and, besides, less low level information is needed compared to the level below. But the upper level has more and further reaching decisions to be made. Specifically, the lowest level is the model creation level. It can only be done by people having deep knowledge about the special kind of model. On the other hand, the last level has a few specified control variables, which influence all lower levels. Different choices of values for these control variables can be made and the results can be shown. But, in the end, these values should be chosen wisely. There can be as many levels as needed. The definition of the different levels can be done in the model creation level, the model definition level and, perhaps, in the model parameter level. It is done by defining open parameters for higher levels and specifying visualizations and graphs for these levels. The definition will be saved in configuration files for each model.

Now we give an overview of different possible levels:

- Model creation and implementation level

The lowest level will be the model creation and implementation level. At this level, the models are classes of models, like utility function models. To implement one class of a model, this level has to be well understood. Therefore, the implementation can only be done from people with much knowledge about a specific model class. In most of the cases, these people have to be from scientific background, like universities. The implementation is done, as shown in section 4.2.2, with Matlab. Therefore, some knowledge about the implementation language is also needed.

- Model definition level
  The second level is the model definition. A model definition can be thought of as a specific instance of a class of models with a specific structure. If the implementation of a model allows an arbitrary structure or, at least, a kind of customization of the structure of the model, it can be designed at this level. It can also be possible that this level is hard-coded in Matlab, if an abstraction to more customization is impossible. Then, different predefined models will be given. Otherwise, at this level there will be an editor created to design the structure of the model in an easy way. This is already done as a prototype for the Security Problem Structurer. To use this level, the user must have deep knowledge about the class of the model, and also of designing such a model. This kind of users should include scientists from universities and consultants from consultant companies. On this level, many different model parameters, which parameterize the model and allow modifying it extensively, are specified.
- Model parameter level
  Parameters defined in the model definition level can be set in the model parameter level. Besides, some parameters can be left out or, perhaps, filled with formulas depending on a new aggregation variable. These parameters and variables can be determined as parameters for the next level. Some parameters might be depending on statistical information and might require tables of empirical information about the environment and company. This information might only be gathered by different employees of the company with access to low level information, because it is only available there, whereas it is only available in aggregated form on higher levels. Therefore, the users of this level might be employees with access to low level information.
- Policy parameters level
  There are possibly parameters defined for this level by the model designer or parameters that were left out on the model parameter level. This parameter level should be used if many (more than five) parameters have to be filled in. These parameters might define some characteristics of the company, like the importance of different goals and the target values of the goals. This level needs the user to have knowledge about the global strategy of the company and to have rights of decisions. So, the users of this level can be seen as low-level decision makers.
- Control parameter level
  The highest level can be seen as the control parameter level. This level should not include more than five parameters, in the most cases only one or two different parameters. These parameters highly depend on the strategy of the company, so the user will be a high-level decision maker.

-

Level	Tasks	Users
Model creation and implementation	Create and implement classes of models	Scientists
Model definition	Design the structure of an instance of the model	Scientists and consultants
Model parameter	Fill in environment and company specific information	Consultants and employees
Policy parameters	Fill in strategy policy parameters	Employees and decision makers
Control parameter	Analyze and define the value of the control parameter	High level decision makers

Table 1: Definition of the different levels of abstraction usable in the tool framework

The advantage of this approach shall be summarized here. The main gain of the approach is that many different kinds of users can make easy usage of the tool, from the model designer and consultant up to the decision maker himself. Besides, it is important to note that needed knowledge is only available for different users. The tool aggregates the required information, as well as shown information in the way the structure of the company does. Perhaps, this can be done by allowing determining arbitrary levels of parameters.

4.3 Design of different tools

In this section we point out how the different tools in our toolbox are designed and what features they provide. We also compare them to the existing tools introduced in chapter 2 and expose the improvements made in our tools with respect to fulfill our requirements.

4.3.1 Influence Diagram Editor

This tool is a Graphiti based prototypic editor for defining, modeling and analyzing decision theoretic influence diagrams. The editor's structure and functionality are basically inspired by GeNIe editor (see section 2.3), but it is further extended to meet those of our requirements that GeNIe does not fulfill. A great difference with respect to GeNIe is that there can be two or even more stakeholders acting in one model. First of all, the representations of the three model elements Chance, Decision and Value were implemented, and a hypercube data structure to handle all the different input combinations for each model element was developed. Further possible features of the editors are diagrams inside each element, which visualize the internal distribution inside the element. Ideally, these distributions can be both discrete and continuous. Besides the graphical representation in a diagram, there will be also tabular visualizations for empiric distributions. The actions which are executed by the decision makers can also be both discrete and continuous. A graphical representation of the Defender-Attacker model that has already been handled in section 2.3 has been developed with the new Influence Diagram Editor and can be found in Figure 26.

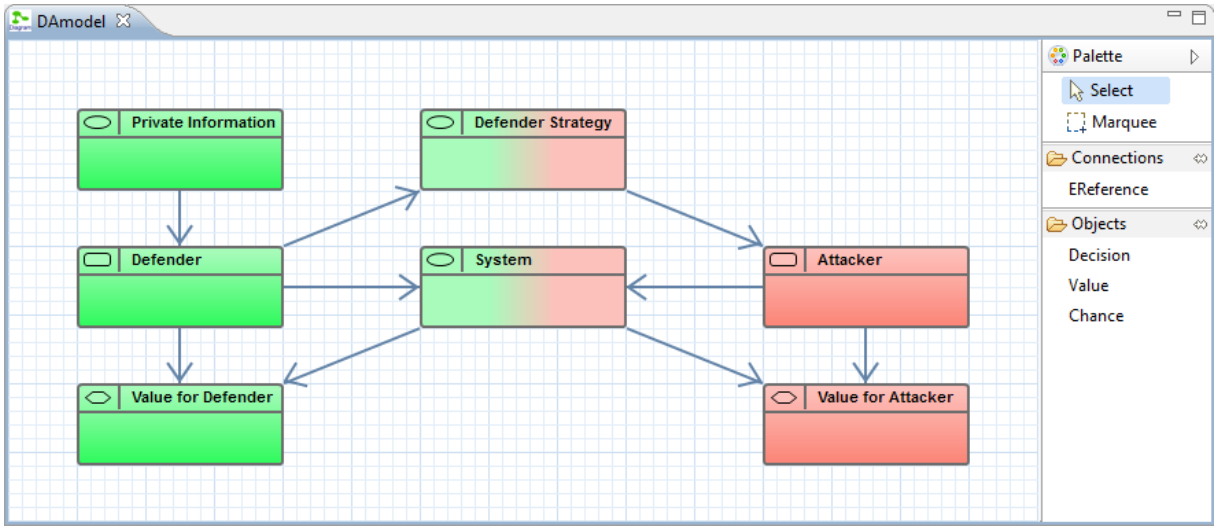


Figure 26: DA Model in Influence Diagram Editor Prototype

4.3.2 Decision tree visualization

With the help of this tool, we can visualize probability theoretic decision trees. This is necessary due to the sometimes confusing display of tables from influence model elements, when there are a lot of parental elements which influence a single element. The visualization as a decision tree makes it easier for the operator to understand the relations in complex influence diagrams. Such a decision tree is illustrated in Figure 27.

4.3.3 Adversarial Risk Analysis

The Adversarial Risk Analysis (ARA) can build a security decision problem from the perspective of the organization which is protecting an infrastructure. While this may be done with influence diagrams for the defender and for the attacker, the ARA is further able to identify potentially effective countermeasures. Thus the model includes a database of countermeasures with relevant features, such as cost, difficulty in implementation, efficiency and more. It helps identifying and modeling the constraints relevant for the problem including the maximum budget available or human resources available.

4.3.3.1 General

ARA is based on a game theoretical approach. There are two actors, a defender and an attacker. As an input, the model can use statistical data from the case studies, e.g., probabilities or probability distributions. The model can be customized by selecting one of the predefined types (although there can also be complicated games, such as Sequential Defend-Attack-Defend-Attack, SEQ-DADA) and by defining actors and actions for them. Actions can be associated with specific probability distributions and costs. Then the tool calculates the optimal decisions for the defender. The module is divided into four tasks.

- First, the problem can be modeled as one of the predefined types as the one explained above, for example SEQ-DA, SEQ-DAD, SIM-SAD. At a later stage, there could be more complex types, like SEQ-DADA. Perhaps, some types can be modeled with a specific model editor, which is a decision at a later stage. In the default there are two actors defined, the attacker and the defender. As described above, perhaps there is a possibility to add multiple actors.

- After defining the actors, the possible actions have to be declared. These actions can be displayed as a decision tree, like the one shown in Figure 27. In this graphical presentation, it is easy to add new possible actions. At each transition, a probability distribution can be added, especially for the last transition to the final state. First, there are only predefined phases available, perhaps in a later stage, there could be the possibility to add further phases to the decision tree.
- After that, the problem information has to be provided, i.e. information about the defender's preferences and beliefs, and the defender's opinion about the attacker's preferences and beliefs.
- After this input, the problem can be started to get solved. This happens in two phases. In phase one, the probability of various attacks depending on the problem and the given defense strategy will be identified with Monte Carlo. All these distributions will be shown as a graph. Possibly, some sensitivity analysis will be performed by the defender. In the second phase, the resulting problem of the defender will be solved. Some kind of automatic report will be generated to describe why this is the optimal resource allocation.

4.3.3.2 Parameters

As an input for the Matlab analysis toolbox, there are several parameters needed. Some of them can be only defined in later stages, but some of them are already clear, and will be described here. The first parameter, which chooses the Matlab function to execute, is the selected model. As mentioned in section 4.2.2, it can be selected in an analysis selector. Any predefined model has several assumptions. These are the number of players/actors, and the number of turns of the game. These parameters cannot be changed as they are essential for the calculations of the model. Perhaps, in a later stage, these assumptions might be softened. But the possible actions for each player have to be defined. There are also parameters to be defined for the preferences, and beliefs of the defender and the probability distributions for the preferences and beliefs of the attacker from the point of view of the defender, as they have uncertainty. Furthermore, the probabilities of the possible states of the system depending on the choices of actions of the defender and attacker have to be determined. All probabilities and probability distributions used in the model can be changed.

4.3.3.3 Editors

Although the models are predefined, they can be visualized with influence diagrams like the one in Figure 26. Within this visualization, it is easy to change the parameters. The model can be checked if all parameters are entered correctly, before it is sent to the Matlab calculation engine. Decisions of actors can be displayed as decision trees, as shown in Figure 27. In this way, a complete view over the model can be gained.

4.3.3.4 Graphs and Outputs

The model calculates the probability distributions for the defender (and also for the attacker). With these distributions, the optimal policy can be determined. The output will thus be the graphs of the probability distributions and the optimal policy.

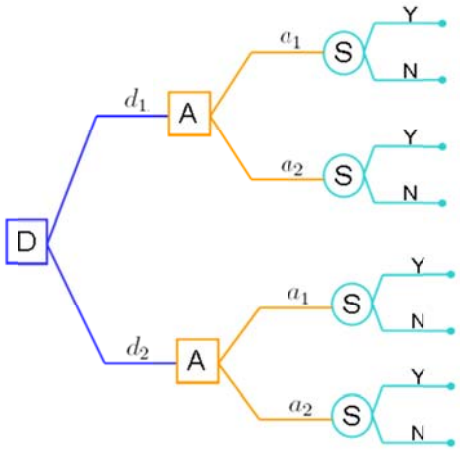


Figure 27: Example decision tree with defender D and attacker A and State S

4.3.3.5 Solving the Attacker's problem using Matlab

The tool we have chosen for the implementation of the ARA models is Matlab. The ARA models can be implemented in an intuitive and flexible manner in Matlab, allowing for the inclusion of new pieces of code, or modules, when the model becomes larger and/or more complex. Scripts for the resolution of the Sequential Defender-Attacker and the Sequential Attacker-Defender problems have been already written and tested, with excellent results. We have already developed templates to solve the basic models introduced in deliverable *D5.1 - Basic Models for Security Risk Analysis*. Following, we show the Matlab code for the Sequential Defender-Attacker problem. See Appendix 1 for details on Matlab code implementation.

4.3.4 Economic Visualization

For a chief information security officer understanding the relative impacts on the security attributes of a system by adjustments to security policy and investment are critical. Consider the following dynamic stabilization model whereby the system state is described by a vector model where:

$$y_t = \left[C_t, I_t, A_t\right]', \quad \bar{y} = \left[\bar{C}, \bar{I}, \bar{A}\right]'$$

Here, *C, I* and *A* represent measurements of confidentiality, integrity and availability. The subscript *t* represents time and the bar symbol represents a target level based on a set of policy preferences. The dynamic evolution of *y* is assumed to follow a stochastic process, with general functional form:

$$y_{t+\Delta t} = F\left(y_t, x_t, \int_t^{t+\Delta t} \varepsilon_t \, ds\right)$$

Here, *F*(.) is a vector function mapping a series of control variables *x* and a noise process ε. A specific functional form is proposed in [16] and uses a two dimensional Brownian motion, with non-linear adjustment in Confidentiality and Availability to trade function-

al priority between these two features. Policy is implemented by minimizing the policy makers expected loss function,

$$x_t^* = \arg\min_{x_t} L(\beta, x_t)$$

By adjustments of the control variable *x*, the loss function in its most general form is:

$$L(\beta, x_t) = \int_t^{t+\Delta t} \exp(-\beta t) f(y_t - \bar{y} | x_t, \varepsilon_t) ds$$

Here, *f* is a real valued function that translates expected deviations from target levels of confidentiality and availability into an expected loss for the policy maker and β is a discount factor. In [17], we present a methodology for exact identification of this loss function under a variety of assumptions of the process driving the future stochastic evolution of *y*.

4.3.4.1 Visualization of the expected investment profile

The mathematical models will be implemented in Matlab. These can be integrated into Java using the Matlab Builder JA so that no Matlab license for the actual execution of the calculation is needed. The model receives statistical data as input with which the derived functions can be evaluated. Among these data model parameters are required. These are coefficients for the cost or utility functions. The data and parameters are entered in the interface and then passed to the Matlab component (see Figure structure). There, the implemented functions are evaluated, the optimum of the cost or utility function is calculated and the result is returned to the interface. The functions can be presented as a graph using the actual values from the case studies. These together with the calculated results are given in a report.

4.3.4.2 Example 1: Fixed versus variable costs and choosing between regular security cycles versus security adjustment on arrival of threats.

In this short subsection we present visualizations of a series of results on the timing of security investment (in section 4.3.5 of this deliverable we address the measurement of security investment in detail).

This example is based on the framework developed in [17], in this instance we consider a firm deciding on the timing of security investment. We assume that the loss function is of the Kahnemann-Tversky type with fixed points about the target levels of *C*, *I* and *A*, see the following Figure 28.

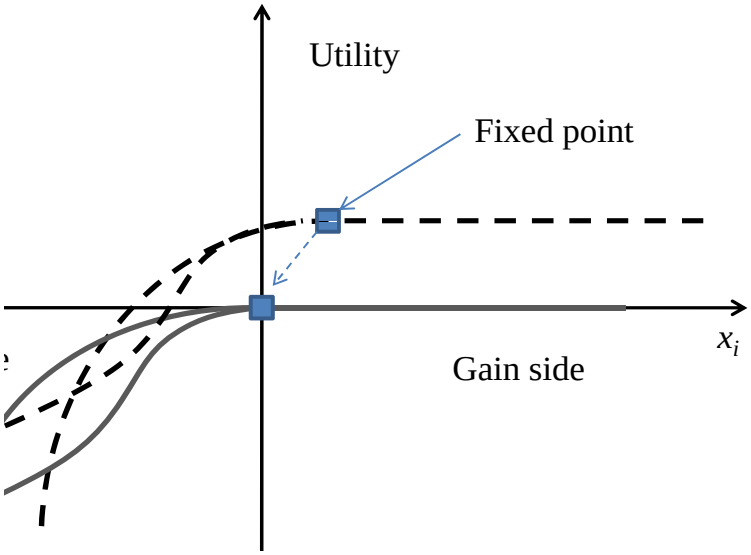
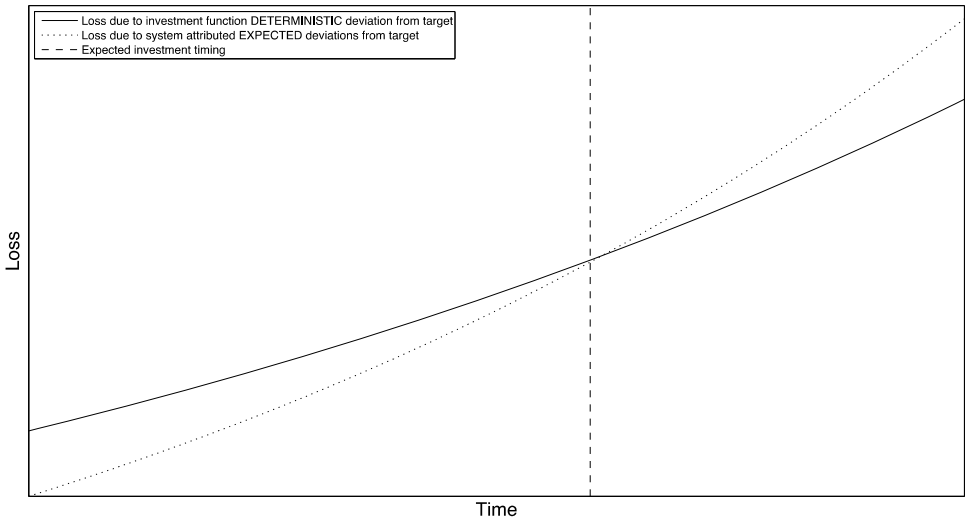


Figure 28: Kahnemann-Tversky type loss function with fixed points about the target levels of C, I and A

The policy maker then trades of future discounted loss over some fixed time horizon (that may or may not be infinity). The speed of investment is then determined by the weighted trade-off between degradation of the systems security attributes and the fixed and variable cost of investment in mitigating them. This trade-off is captured by the expected time profile of two integrals, subtraction yields a trade-off curve, traversing the zero axis represents the expected time of investment. For instance in a typical model, these loss functions evolve over the forecast horizon, see Figure 29:



The trade-off curve is the difference between these two losses and can be seen in Figure 30:

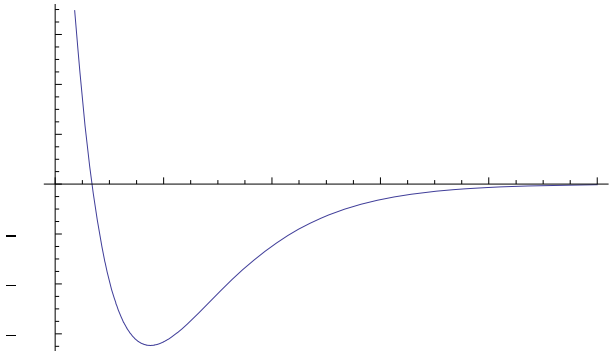


Figure 30: The trade-off curve is the difference between these two losses

This model predicts that implementation of security mechanisms will take place on a weekly basis (the line crosses the abscissa axis at roughly 7 days). The plots above are constructed using the Matlab programming language and maybe embedded in webpages or applets designed to illustrate the choices available to the CISO. In this case we can adjust discount rates to demonstrate the impact on the decision to adjust the investment profile and the timing of security investment.

4.3.4.3 Example 2: Firm stabilization profile

As the firm responds to security threats the adjustment profile outlined above will affect the investment profile and other potential control variables. For instance if security investment is one control variable and the degree of activity (from partial to complete shutdown of IT systems) is another, then we can simulate a firms return to a normal operating profile using an impulse response analysis.

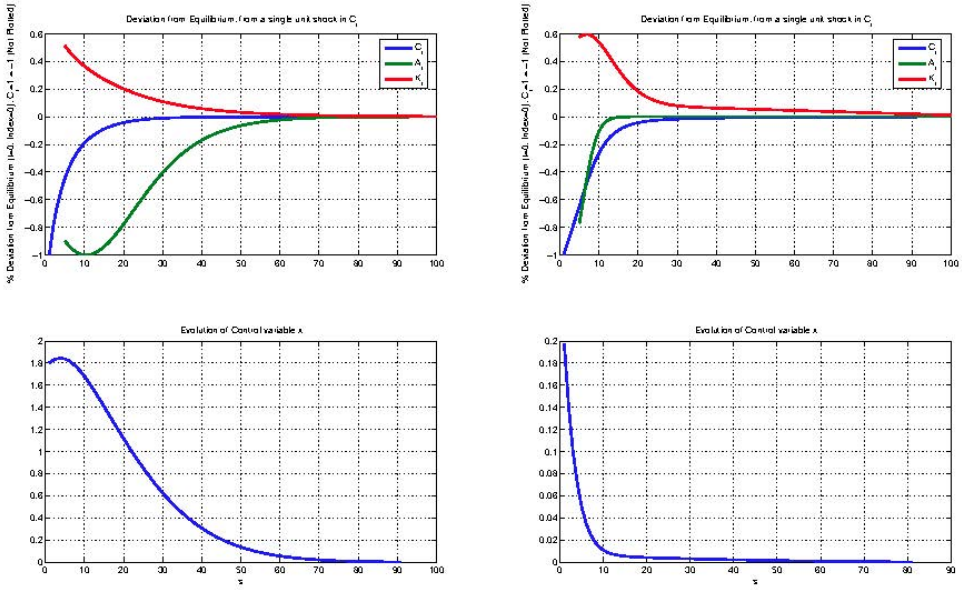


Figure 31: First row: Deviation from the equilibrium from a single unit shock in C; Second row: Evolution of Control variable; For example 1 and 2.

The usefulness of this is in indicating the length of time compared to the size of the security shock. This approach may be aggregated across firms to provide an aggregate security model for public policy.

The time profiles for investment and system activity may then be used to estimate total cost of security incident over the life-cycle of the security incident.

4.3.5 Return on Security Investment

The cost of security incidents, caused by accidents, errors or attacks, can be reduced by employing security applications. Return on Security Investment, abbreviated as RoSI, provides information about the benefits of these applications. The costs of the security measures have to be set in relation to the original costs caused by the incident and the reduction of the actual probability for an incident, or the reduction of the incident costs, respectively. It is often hard to identify these measurements; although there are various approaches to obtain them as, for example through the application of Bayesian Belief Networks or other statistical metrics. But when all the necessary parameters for a RoSI analysis are well known, the problem is reduced to the application of a simple formula. There are different formulas leading to different types of results, such as an absolute amount of the benefit, or a factor dependent on the security investment, among others. One of these formulas will be introduced exemplarily in the following. For this formula, which is described in detail in [10], is introduced as an illustration

$$RoSI = \frac{CT_{before} - CT_{after}}{TCP}$$

where three parameters have to be known

- The cost of threats before the application of security measures (CT_{before})
- The cost of threats after the application of security measures (CT_{after})
- The total cost of protection by a security measurement (TCP)

The difference of the cost of threats caused by security incidents before and after the application of security measures is set in relation to the security measure's total costs. The RoSI resulting from this formula has to be interpreted as a factor, by which the benefits gained with the security application exceeds its costs. When RoSI > 1, the security investment is useful, as its cost is smaller than the expected vulnerability reduction.

4.3.6 BPMN

Business Process Modeling Notation (BPMN) is a graphical notation to visualize business processes. The basic element for each model is a pool consisting of one or more lanes. The whole business process is represented by the pool, whereas the lanes model different accountability areas. Processes consist of different components, such as tasks, gateways and associations between them. A task represents an activity that has to be executed in the process. Activities are associated with each other by flows that model the order of execution. Flows can be absolute, when a sequence of activities should always be executed in the pre-defined order, but they can also be dependent on different parameters. Therefore, the gateways are used to model decisions in a flow. By means of a dependent gateway, only one flow can be processed (OR gateway), but it is also possible to execute even more flows, to model parallel processes (AND gateway). The de-

pendencies can be checked for each flow separately, so that even the number of flows that are actually processed can also vary.
Gateways can also be applied to merge certain sequence flows. There are also different options concerning how to trigger the outgoing flow. This can be done, for example, when the first flow reaches the gateway but, alternatively, after all ingoing flows have reached it.

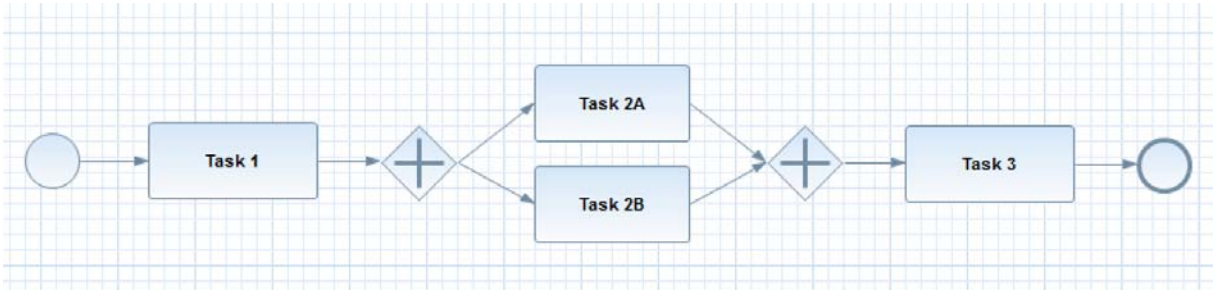


Figure 32: Simple BPMN process with parallel sequence flows

Figure 32 shows a simple example for a business process modeled with BPMN. After executing Task 1, the gateway splits the control sequence up into two parallel flows and the tasks 2A and 2B have to be processed in parallel. After both executions have finished, the second gateway merges the sequence flows and Task 3 can start.

At Fraunhofer ISST, an extension for this BPMN specification is being developed, with detailed information in [7]. With the extension, it is possible to apply risk analyses to BPMN models. Based upon BPMN Modeler 2.0, Risk and Mitigation are integrated as additional model elements into the specification. The Risk element requires two parameters: the loss that a certain risk causes when it occurs, and the probability of occurrence of the actual risk. Risk elements are associated with process tasks, as well as with other risk elements. The required parameters can be set in the risk element itself once, but can also be overwritten in each association individually. This makes it possible that Loss and Probability of a Risk can vary for each task. Risk elements can also influence each other's parameters. It is possible that a risk can increase or decrease the probability of occurrence and the loss caused by another risk to a certain task.
The second additional element of the BPMN extension, Mitigation, is a counter-measure to risk. Mitigations are associated with risks but they can also affect other mitigations. As input parameters, a Mitigation element needs actual costs of its implementation, and values for probability, and loss reduction to a certain risk. The costs of Mitigations are fixed, whereas the probability reduction can vary for each associated element. By means of loss reduction and probability reduction, Mitigations can influence losses caused by a risk. They can also influence the parameters of other Mitigation elements both positively and negatively. Another possibility is that a Mitigation element can exclude the application of certain other mitigations. It is also conceivable that mitigations depend on the existence of another specified mitigation.
After such an extended BPMN model has been created with all its risk and mitigation elements and the corresponding parameters, it is possible to perform security analyses on this model. Therefore, Return on Security Investment analyses can be applied to it. Then all possible combinations of mitigations in the model are evaluated and compared. In the end, the combinations that lead to the best results represented by the minimized total estimated risk are highlighted within the analysis view.

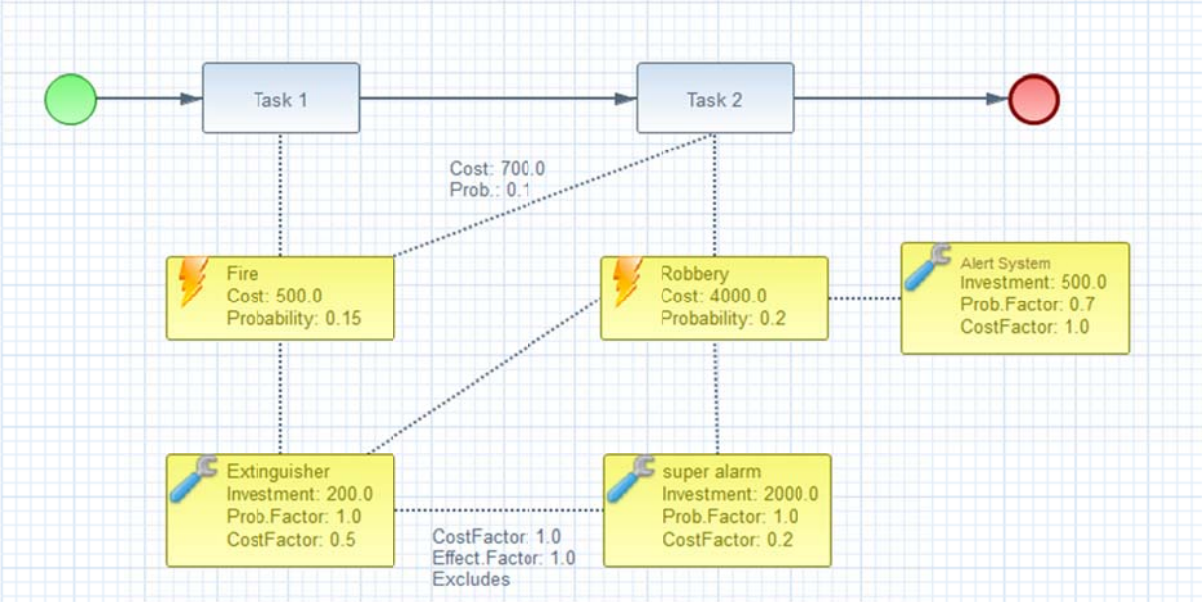


Figure 33: Extended BPMN process with risks and mitigations

Figure 33 shows an extended BPMN model, where the two sequential tasks, Task 1 and Task 2, are threatened by the Risks Fire and Robbery, which are influenced by the Mitigations Alert System, Extinguisher and Super Alarm.

4.3.7 Statistical Tool

The models developed in the context of this project are partially based on statistical information, gained from the case studies. These statistical models might be implemented as part of the toolbox. Then the toolbox will have a data importer to import the statistical data and show them in tabular form. Together with this information, some parameters can be entered, like e.g. thresholds. Then, the statistical analytics can be started and the visualizations are shown. The analytics should be implemented also in Matlab. Customary statistical visualizations are, in principle, to be used, as e.g. histogram, boxplots, or pie charts. The toolbox will provide a statistical tool, if needed. This tool will enable the user to input statistical data on various places. Other tools and models can use this package. The data can be imported from tables in different formats if needed. After importing the data can be displayed in tables and edited, if necessary. With this data, an empirical distribution is associated. Another possibility is to select a distribution from a portfolio of different pre-implemented distributions like Gaussian distribution, logistic distribution, Beta distribution, Uniform distribution, etc. Furthermore, this tool enables the user to do some basic analytics, as calculating key figures for the empirical data like median, weighted arithmetic average, geometric average, harmonic average, standard deviation, variation coefficient, quantiles, among others. If specific functionalities are needed while researching the different parts in the other work packages, these functionalities will be provided here.

4.4 Links to other work packages

In this section we contour the relations of our tool in WP8 to the other work packages of SECONOMICS. Therefore, we point out which collaborating partners make use of specific parts of the tool developed within WP8.

4.4.1 Work package 4

Work package 4 is led by the SOÚ Institute of Sociology of the Academy of Sciences of the Czech Republic. Managed by P. Guasti, Z. Mansfeldova and J. Hronesova they use the sociological analysis of security threats. Therefore, they will use mainly the statistical tool. The implementation of risk analyses from sociological perspectives is mainly accomplished by the developers in SOÚ Prague, with support by Fraunhofer ISST.

4.4.2 Work package 5

Work package 5 which is led by David Ríos Insua from Rey Juan Carlos University in Madrid, uses the adversarial risk analysis of security threats. Therefore, the adversarial risk analysis tool and also the statistical tool will be mostly used. Fraunhofer ISST supports URJC Madrid in implementing the risk analyses.

4.4.3 Work package 6

Work package 6 uses the risk analysis from an economic point of view. Led by Julian Williams from the University of Aberdeen, most of their tools are developed by themselves with little support from Fraunhofer ISST. Mostly the financial tool will be used in the analyses from the members of work package 6.

5 Summary and conclusion

In this report we have seen how the toolbox will be created. At the end of the project, an easy to use, and seamless integrated toolbox will be developed. After an introduction, defining the main targets and stakeholders of the toolbox, several tools developed and used in the consortium are evaluated. Specifically two tools should be emphasized here. Matlab is used to build the mathematical calculations for models. Then Matlab functions will be integrated into the toolbox. The second tool is GeNIe. Although GeNIe does not meet our requirements, some good ideas can be reused, mainly for the influence diagram designer. Furthermore, the techniques used in the implementation are explained as far as possible at this stage. The most important techniques are the integration of the Matlab calculations into the toolbox. This can be done either live with a running Matlab instance or compiled as a Java library. The second technique is the framework Graphiti used for building visualizations in editors. With this framework it is relatively easy to build such visual editors for our purposes. The last technique, mentioned here in the summary, is the plugin strategy. All models and tools used in the toolbox are built as a modular plugin.

The design of the toolbox is explained. The tool gets data from the case studies as input. A specific model is created by adjusting the parameters, as far as needed, and assisted by wizards, or by building a specific model with a visual editor. After the calculation has been run in Matlab, the results are visualized, for example as graphs. This is the output of the model.

Summarizing, an easy to use, seamless toolbox is created, where all mathematics is hidden behind the same user interface, and the use of the parameters is guided.

6 Bibliography

[1] Sven Wenzel, Daniel Warzecha and Jan Jürjens: „CARiSMA – Evolution-aware Tool Support for Model-Based Security and Compliance Analysis". In: *The 27th Intl. Conf. on Automated Software Engineering*, 2012.

[2] Gábor Bergmann, Fabio Massacci, Federica Paci, Thein Than Tun, Dániel Varró and Yijun Yu: "A Tool for Managing Evolving Security Requirements". In: *Lecture Notes in Business Information Processing 107*, pp. 110 – 125, 2012.

[3] Siv Hilde Houmb, Virginia N.L. Franqueira and Erlend A. Engum: "Quantifying security risk level from CVSS estimates of frequency and impact". In: *The Journal of Systems and Software 83*, pp. 1622 – 1634, 2010.

[4] Virginia N.L. Franqueira, Siv Hilde Houmb and Maya Daneva: "Using Real Option Thinking to Improve Decision Making in Security Investment". In: *Lecture Notes in Computer Science 6426*, pp. 619 – 638, 2010.

[5] Siv Hilde Houmb, Indrajit Ray and Indrakshi Ray: "SecInvest – Balancing Security Needs with Financial and Business Constraints". In: *Dependability and Computer Engineering – Concepts for Software-Intensive Systems*, pp. 306 – 328, 2012.

[6] Theodor Schnitzler: "Entwicklung eines Eclipse-Plugins für Matlab-unterstützte Auswertungen des Return On Security Investment", Bachelor thesis. Dortmund, Germany: TU Dortmund, 2012.

[7] Dominik Thalmann: „Erweiterung der Business Process Modeling Notation für Return on Security Investment Analysen", Diploma thesis. Dortmund, Germany: TU Dortmund, 2013.

[8] Joshua Kaplan. Matlabcontrol - A Java API to interact with MATLAB. 2012. URL: code.google.com/p/matlabcontrol/.

[9] MATLAB Builder JA User's Guide. 2.2.4. The MathWorks, Inc. 3 Apple Hill Drive, Natick, MA 01760-2098, 2012.

[10] Vicente Aceituno. Return On Security Investment. In: *ISSA Journal - The Global Voice of Information Security* pp. 6-9, 2012.

[11] Jesus Rios and David Rios Insua. Adversarial risk analysis for counterterrorism modeling. In: *Risk Analysis* pp. 894–915, 2012.

[12] David Rios Insua and Javier Cano. Basic Models for Security Risk Analysis. SECONOMICS D5.1

[13] Marek J. Druzdzel. SMILE: structural modeling, inference, and learning engine and genie: A development environment for graphical decision-theoretic models. In:

Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99), pp. 342-343, Orlando, Florida, USA, 1999.

[14] David J. Lunn, Andrew Thomas, Nicky Best and David Spiegelhalter. WinBUGS – A Bayesian modelling framework: Concepts, structure, and extensibility. In: *Statistics and Computing 10*, pp. 325–337, 2000.

[15] D. Pym, C. Ioannidis and J. Williams, Information Security Trade-Offs and Optimal Patching Policies, (European Journal of Operational Research, Volume 216, Issue 2, 16 January 2012, Pages 434-444)

[16] D. Pym, C. Ioannidis and J. Williams, Investments and Trade-offs in the Economics of Information Security, in Financial Cryptography (2009), International Financial Cryptography Association (IFCA), Springer Lecture Notes in Computer Science (LNCS)

[17] D. Pym, C. Ioannidis and J. Williams, Fixed Costs, Investment Rigidities, and Risk Aversion in Information Security: A Utility-theoretic Approach. Proc. WEIS 2011, Springer-Verlag George Mason University, Fairfax, Virginia, 14-15 June, 2011: WEIS 2011.

[18] Matlab Documentation 2012, Key Product Description

[19] Matlab Compiler Documentation 2012, Key Product Description

7 Appendix 1

Matlab code for implementing a sequential defend-attack-problem:

```
% SEconomics project
% Script for the simulation algorithm for the sequential defend-attack
% problem

% Version 0.1. 2012/10/09. General structure

% See also TRIANGRND.M, DRCHRND.M

clear all
N=1e4; % size of the MC sample
% The first part of the algorithm deals with the estimation of pD(aj|di),
% i.e., the probability (from the point of view of the defender) that the
% attacker will perform an attack aj, given that the defender has
% performed a defense di.

% We assume that we have m possible defenses, n possible attacks, and ns
% possible outcomes of the attacks, following the notation in the
% deliverable 5.1.

m=4;  % Number of possible defenses
n=3;  % Number of possible attacks
ns=3; % Number of possible outcomes of the attacks

% We define the possible outcomes of the attack s\in{0,1,...,ns-1},
% although here we use their cardinals: 1,2,...,ns.
S=1:ns;

% We define an (m x n) matrix PDhat containing all the \hat{p}_D(a_j|d_i).
% This matrix does not depend on the number ns of possible outcomes of the
% attacks

PDhat=zeros(m,n);

% We define the utilities uD for the different success of the attack. This
% is an (m x ns) matrix, being its first column s=1 and the rest of the
% columns s=2,..., s=sn
uD=[200 50 10;
100 20 10;
80 10 0;
50 0 0];

% We define the probabilities pD(S=s|d,a). When there are ns outcomes for
% the attack, we have to define all the probabilities pD(S=s|d,a),
% s\in{1,...,ns}, so we create an (m x n x ns) array with such
% probabilities

pD(:,:,1)=[0.45 0.2 0.1;
      0.55 0.45 0.3;
      0.65 0.55 0.45;
      0.85 0.75 0.65];
pD(:,:,2)=[0.25 0.45 0.5;
      0.2 0.25 0.35;
      0.15 0.2 0.25;
      0.1 0.15 0.2];
pD(:,:,3)=[0.3 0.35 0.4;
```

```
      0.25 0.3 0.35;
      0.2 0.25 0.3;
      0.05 0.1 0.15];

% DOUBLE CHECK. The probabilities pD(S=s|d,a) for fixed d,a must sum up to 1
if any(any(sum(pD,3)-1>1e-10))
      warning('Double check the probabilities pD. There are inconsistencies')
      pause
end

% We define now the parameters of the (triangular) distributions for the
% utilities UA(a,s), for the different outcomes of attacks.
% We need to define a specific function to sample from it, as it is not
% implemented in Matlab. a, b, c are the minimum, mode and maximum of the
% distribution, respectively, It is defined in the M-file TRIANGRND.M.
UApar(:,:,1)=zeros(3); % For the case s=0 (failure)
UApar(:,:,2)=[50,60,80; % For the case s=1 (moderate damages)
      40,50,60;
      25,40,50]';
UApar(:,:,3)=[80,100,100; % For the case s=2 (severe damages)
      60,80,90;
      60,70,90]';

% We define now the parameters of the Dirichlet distributions for the
% probabilities PA(S=s|a,d), for the different types of attacks and
% defenses. We define it as an (m x n) cell array, and, in the (i,j) cell
% position, we store the parameters of the Dirichlet distribution
% corresponding to the i-th defense and j-th attack.
% We need to define a specific function to sample from it, as it is not
% implemented in Matlab. It is defined in the M-file DRCHRND.M.

PA=cell(m,n);
for i=1:m
      for j=1:n
            probaux=pD(i,j,:);
            PA{i,j}=probaux*10;     % We multiply by 10, so they have smaller
                                    % variance
      end
end
% We define an (m x 1) matrix to store the expected utility for the defender

K=1; % Number of replicas to obtain boxplots for the utilities psiD
% This part can be removed if no boxplots are needed setting K=1.
psiDall=zeros(K,m,1);
doptimall=zeros(K,1);

for k=1:K
      for i=1:m % For each possible defense. In each of these loops, we compute
            % all the probabilities \hat{p}_D(a_j|d_i) for fixed i
            % (N x n x ns) matrix to store the samples for the probabilities for
            % each defense and each attack
            PAsample=zeros(N,n,ns);
            % (N x n x ns) matrix to store the samples for the utilities
            UAsample=zeros(N,n,ns);
            % (N x n) matrix to store the samples for the expected utilities
            psiAsample=zeros(N,n);
            % We sample from the distributions of UA and PA, which are the
            % assessments of the Defender about the probabilities and utilities
            % of the Attacker
            for j=1:n % For each possible attack
                  for s=1:ns
                        UAsample(:,j,s)=triangrnd(N,UApar(:,j,s));
```

```
                    end
                    PAsample(:,j,:)=drchrnd(squeeze(PA{i,j})',N);
                    % We compute the expected utility for the defense i (fixed)
                    and all
                    % possible attacks j=1:n
                    psiAsaple(:,j)=
                          sum(squeeze(UAsample(:,j,:)).
                          *squeeze(PAsample(:,j,:)),2);
              end % end of for j=1:n
              % We compute which attack provides the maximum expected utility
              [psiAmax,aoptim]=max(psiAsample,[],2);
              for a=1:n
                    PDhataux(a)=length(find(aoptim==a))/N;
              end
              % We annotate the probabilities \hat{p}_D(a_j|d_i) for fixed defense
              % di
              PDhat(i,:)=PDhataux;
        end % end of for i=1:m

        % In this auxiliar matrix, we compute the products uD x pD for all
        % possible outcomes of the attack
        psiDaux=permute(repmat(uD,[1,1,n]).*permute(pD,[1,3,2]),[1,3,2]);

        for i=1:m
              % We now compute the optimal defense
              psiD(i)=sum(PDhat(i,:).*sum(squeeze(psiDaux(i,:,:)),2)');
        end

        psiD';
        disp(['The expected utilities are ',num2str(psiD)])
        psiDall(k,:)=psiD;
        [psiDmax,doptim]=max(psiD);
        doptimall(k)=doptim;
        disp(['The optimal defense is d',num2str(doptim)])
end

% We plot the boxplot only if we have more than one replication (K>1)
if K>1
        mean(psiDall)
        boxplot(psiDall,'labels',{'d_1','d_2','d_3','d_4'})
        ylabel('Expected utility')
        title('Expected utilities for the Sequential Defend-Attack problem')
end
```