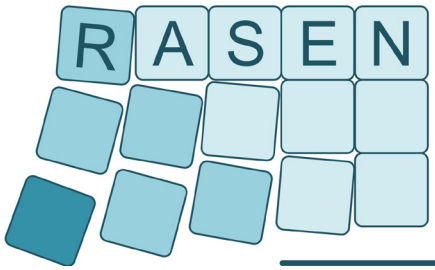

Compositional Risk Assessment and Security Testing of Networked Systems

Deliverable D5.2.1

Risk Assessment and Security Testing Toolbox Requirements and Design

Project title:	RASEN
Project number:	316853
Call identifier:	FP7-ICT-2011-8
Objective:	ICT-8-1.4 Trustworthy ICT
Funding scheme:	STREP – Small or medium scale focused research project

Work package:	WP5
Deliverable number:	D5.2.1
Nature of deliverable:	Report
Dissemination level:	PU
Internal version number:	1.0
Contractual delivery date:	2013-03-31
Actual delivery date:	2013-03-22
Responsible partner:	FhG FOKUS

Contributors

Editor(s)	Jürgen Großmann (FOKUS)
Contributor(s)	Fredrik Seehusen (SINTEF), Fabien Peureux, Bruno Legeard (Smartesting), Jürgen Großmann, Martin Schneider (FOKUS).
Quality assuror(s)	Fabien Peureux (Smartesting), Arthur Molnar (Info World), Fredrik Seehusen (SINTEF).

Version history

Version	Date	Description
0.1	13-02-25	Initial DRAFT with input from the RASEN GA
0.2	13-02-28	Updated integration requirements and CORAS tool integration interface description
0.3	13-03-06	Restructuring and integration of partner contributions from SAG, SINTEF and UIO
0.4	13-03-07	Integration of contribution from SMA
0.5	13-03-12	Input from SINTEF
0.6	13-03-19	Generic testing model (FOKUS)
0.7	13-03-19	SMA review
0.8	13-03-20	IW review
1.0	13-03-22	Final quality check (SINTEF)

Abstract

The overall idea of the RASEN project is to combine security risk assessment with security testing. Typically security risk assessment and testing are supported by different tools, often multiple tools. In contrast, the RASEN risk assessment and security testing toolbox aim explicitly to the integration of data from security risk assessment and security testing. This deliverable outlines the integration requirements and the initial design of the RASEN risk assessment and security testing toolbox.

Keywords

RASEN, Security Risk Assessment, Security Testing, Tool Integration

Executive Summary

The RASEN risk assessment and security testing toolbox provides tool support for the RASEN approach to risk-based security testing and test-based security risk assessment. This deliverable specifies the initial integration requirements and the initial integration design for the tools that compile the toolbox. The deliverable starts with a short overview on tool integration approaches and tool integration platforms. Tool integration itself is approached from two directions. Generic data models, so called conceptual models, are used to model the data and artefacts that are relevant for security testing, security risk assessment and security risk management. Moreover, so called integration use cases describe cross-tool data exchange scenarios and are used to identify the data to be exchanged between the individual tools of the toolbox. The final section outlines the integration design. It starts with the description of the tools that are already available from the partners and outlines conceptual integration interfaces for the different tool categories of the toolbox.

Table of Contents

1 Introduction

A central innovative idea of the RASEN project is to combine security risk assessment with security testing. Risk assessment is used to improve the direction of security testing and security testing can be used to provide confidence on the assumptions made during risk assessment. Typically security risk assessment and testing are supported by different tools, often multiple tools. In contrast, the RASEN techniques aim explicitly to the integration of data from security risk assessment and security testing and thus tool support for an integrated approach is necessary. The RASEN risk assessment and security testing toolbox provides such an integrated approach. It is initially based on the already existing tools that have been introduced by the various partners of the RASEN project and provides additional tool support that is specific for the techniques that are to be developed in RASEN. The table below provides a conceptual overview on the tools that build the basis of the tool box and the extensions that are foreseen to be developed in RASEN. The table has been introduced with the RASEN DoW and will be refined during the project.

Conceptual tool	Concrete tool	RASEN development
Security risk assessment tool for supporting compositional risk assessment	CORAS risk assessment tool (developed by SINTEF) ARIS Risk & Compliance Manager (developed by Software AG).	Extend existing risk assessment tools in such a way that they can be used in combination with the test derivation and the test result aggregation tools, and such that compositional and continuous assessment is supported
Security testing tool for specifying security test cases (in detail) and generating executable tests	CertifyIt for Security Testing (developed by Smartesting) Fokus!MBT (developed by Fraunhofer)	Extend existing testing tools in such a way that they can be used in combination with the test derivation and the test result aggregation tools, and such that compositional and continuous assessment is supported
Test derivation tool for supporting the derivation and prioritization of test cases based on the risk assessment	None	This tool needs to be developed from scratch using security testing technologies provided by Smartesting and Fraunhofer.
Test result aggregation tool for supporting the aggregation of security test results into a format that allows us to verify the risk picture and to update the risk picture based on the results	CORAS risk monitor prototype (developed by SINTEF) which defines rules for updating the risk assessment at run-time	The CORAS risk monitor may be used as a starting point. However, the tool needs to be rewritten to align it with the RASEN approach. In particular, active testing (as opposed to passive testing/monitoring) must be taken into account. In addition, algorithms for verifying (as opposed to updating) the risk assessment picture need to be developed.

Table 1 – Conceptual overview on the RASEN tools

This document describes the initial tool integration requirements and the initial integration design for the RASEN risk assessment and security testing toolbox. The starting point for the definition of the integration requirements and the initial design is the anticipated data flow between the security risk assessment tools and the security testing tools. Section 2 introduces some relevant basics of tool integration and describes the RASEN approach to integrate the RASEN tools. Section 3 introduces generic data models that are used as a basis for describing the data flow between the RASEN tools.

The data models are intended to cover the main security risk assessment and testing tools that are used in RASEN. Based on these data models, the integration requirements are identified in Section 4. The integration requirements are outlined by means of so called integration use cases. These use cases describe the interaction between the RASEN tools and thus document the integration requirements by emphasizing the data flow between the tools for each use case. Moreover they can be used as evidence that the data models contain sufficient and relevant information needed to support the main RASEN approaches. Finally, Section 5 describes the initial design of the RASEN risk assessment and security testing toolbox. This section starts with a brief analysis of the existing RASEN tools and their integration capabilities. It finally outlines the initial design of the RASEN risk assessment and security testing toolbox. Section 6 concludes the results of the deliverable and provides an outlook on the next steps to be taken to integrate the RASEN tools.

2 Overview of the Integration Approach

The RASEN project focuses on security testing techniques for risk-based test case identification and derivation and on security risk assessment techniques that use aggregation of test case results to update the risk assessment. Because risk assessment, test design and test management are in generally (as well as in RASEN) supported by different tools, the integrative techniques from above require at least a weak integration of the RASEN tools to be sufficiently supported by tools. This section gives an overview on various aspects and technologies for tool integration and presents the tool integration approach favored by RASEN.

2.1 Tool Integration Aspects

Looking at the tool integration problem, we have to look at the characteristics of the individual tools used today. Typically, tools work on their own data structures, which are well suited to the task which needs to be performed by the tool. So the tool can only process data that is relevant for the tool. Tools can save and load their internal data to a file which may have a proprietary format. In such cases it is very difficult to make use of the tool specific data in a different context than the respective tools. So the question is how to transfer the data between tools. Tool integration is not limited to the question of data exchange. The various aspects of tool integration have been discussed in literature for a long time and a couple of tool integration solutions have been developed. Basically, based on work of Thomas and Nejmeh[19] and Wassermann [20] the aspects of tool integration can be summarized as follows:

- **Data integration:** Data integration is probably the most obvious aspect of tool integration. The interesting point here is to share data between different tools. This is crucial for software development tasks, because certain data sets are needed at different locations throughout the complete development process. If data cannot be shared between tools, they need to be replicated by manual work.
- **Control integration:** Control integration is about the availability of certain functionalities provided by a tool in the context of another tool. Control Integration may help in avoiding the implementation or deployment of similar or same functionality in various different tools or locations. Control Integration is not easy to achieve since tools might need a modular architecture, which reflects the service and consumer paradigm. So tools must provide their functionality via dedicated interfaces.
- **Presentation integration:** The objective of presentation integration is to give the user a homogeneous user experience, which means to provide a common look and feel. The benefit of presentation integration is the reduction of the learning and training phase for new tools. So users know already how the UI of the tool is working. There are a couple of frameworks and toolkits that contribute to achieve presentation integration (e.g. Swing, SWT, etc.).
- **Process integration:** This aspect of tool integration is focusing on how tools may interact in order to support a development process. So it is important to identify certain process steps and to figure out which tools can be used for which part of the process and how they have to interact. This includes also the input and output required to complete certain steps. In particular events usually play an important role for process integration.

There are in general two different architectural approaches for integrating tools. This could be either done in a tool coalition approach or in a tool federation approach, where tool coalition is based on point-to-point connection between tools and tool federations are based on a central integration platform. Depending on the context of the tool integration both approach have benefits and drawbacks. The major difference is that tool coalitions can be used easier in small and ad-hoc environments, where tool federations better fit to larger environments. Some tool federation platforms are presented in the next sections.

2.2 Tool Integration Platforms

Jazz [3] is a tool integration approach of IBM. It basically allows data integration and control integration. The data ownership concept of Jazz is such that every tool holds its own data. A centralized and externalized data repository is not part of the architectural design. Only via specific interfaces, subsets of the tool data are made public. Jazz has a particular approach to presentation integration as it allows the rendering of data located in remote tools with the user interface of that tool. This works for specific subsets of data. The communication of Jazz Integration Architecture is based on REST-full web services.

CDO [1] is an Eclipse project and hosted by the Eclipse Foundation. CDO is a data integration technology that is based on a database principle either with object-relational or object-oriented mappings. CDO is a common way to persist data, which is based on the Eclipse Modeling Framework – EMF. It is based on classical client-server architecture. CDO supports basically an online mode, which allows seeing changes done by a team member immediately in the development environments of the other team members. Every update made by clients is communicated to the server and then back to the other clients.

EMF Store [2] became recently an Eclipse project hosted at the Eclipse Foundation. It is another model repository for EMF-models in parallel to the CDO one. EMF store follows a very similar approach to ModelBus as it supports both the online and the offline model for collaboration. EMFStore emphasizes the merge process in Eclipse IDE based environments. Furthermore, it claims to have special support for migration of models. This means that it in particular support the evolution of meta-models. Similar to CDO in EMFStore the meta-models need special preparation before they can be used with EMF-store.

ModelBus [14] is a model-driven tool integration framework that allows building a seamlessly integrated tool environment for system engineering and test processes. It comprises a tool integration platform (run-time environment) as well as the tool integration development environment. ModelBus is based on SOA principles. It consists of a central bus-like communication infrastructure, a number of core services and a set of additional management tools. Depending on the usage scenario at hand, different development tools can be connected to the bus via tool adapters. Once a tool has been successfully plugged in, its functionality immediately becomes available to others as a service. Alternatively, it can make use of services already present on the ModelBus. The particular strength of ModelBus is the automation of individual development steps by using the orchestration facility. It helps to automatically trigger and execute sets of actions needed for running a development process.

CReMa [10] is a traceability platform for Eclipse-based tools. Traceability is a mechanism for relating artifacts that reside in different tools, and is vital for ensuring the completeness of the specification and for the system itself [11]. A system with trace relations can respond accordingly on specification changes and system restructurings. CReMa is being developed by the Itemis AG and has been extended in the DIAMONDS project by Fraunhofer FOKUS to meet the requirements of a Security Test Management Platform (STMP) for Risk-based Security Testing. CReMa and the DIAMONDS extensions allow for tracking the dependencies between risk assessment artifacts that have been defined in CORAS and test results that are provided by the Security Test Management Tool. The plugins to adapt to the tools are written in Java and designed as Eclipse plugins, which also reflects a component based architecture. The whole platform is based on the EMF concepts and integrates with the Eclipse principals for presentation and data integration.

2.3 Conceptual RASEN Tools

The project proposal outlines four conceptual tools to describe the RASEN approach and thus to define the environment for tool integration. A closer investigation of the subject matter and the tools from the partners has shown that a more precise differentiation of tools is necessary to profoundly describe the integration between tools. The following table introduces seven different conceptual tools that are relevant in the context of the RASEN approach.

Conceptual tool	Concrete tool
Security Risk Assessment Tool (SRAT) for supporting compositional risk assessment	CORAS risk assessment tool (developed by SINTEF)
Security Risk Management Tool (SRMT) for continuously managing the results from risk assessments.	ARIS Risk & Compliance Manager (developed by Software AG).
Security Testing Tool (STT) for specifying security test cases (in detail) and generating executable tests	CertifyIt for Security Testing (developed by Smartesting) Fokus!MBT (developed by Fraunhofer)
Security Test Management Tool (STMT) for managing the executable tests and their results.	CertifyIt for Security Testing (developed by Smartesting) Fokus!MBT (developed by Fraunhofer)
Attack Pattern and Test Pattern Database (PDB) for managing attack patterns and test patterns	None, might be later integrated in STT and SRAT
Test Derivation Tool (TDT) for supporting the derivation and prioritization of test cases based on the risk assessment	None, might be later integrated in the STT tools, i.e. CertifyIt for Security Testing or Fokus!MBT
Test Result Aggregation Tool (TRAT) for supporting the aggregation of security test results into a format that allows us to verify the risk picture and to update the risk picture based on the results	CORAS risk monitor prototype (developed by SINTEF), which defines rules for updating the risk assessment at run-time

Table 2 – Conceptual RASEN tools

2.4 The RASEN Tool Integration Approach

As a starting point, the RASEN project has four concrete tools that need to be integrated so that some of their data can be exchanged. With respect to the tool integration aspects outlined in Section 2.1 this refers to data integration and control integration. Other integration aspects like presentation integration or process integration are currently out of scope.

The tools that form the basis of the RASEN toolbox are the risk assessment tool CORAS, the risk management tool Aris and the testing tools Fokus!MBT and Smartesting CertifyIt for Security Testing (called “CertifyIt” hereafter). We refer to the data, which are stored by these tools as the CORAS model, the Aris model, the Fokus!MBT model, and the CertifyIt model, respectively.

As illustrated in Figure 1, our approach for integrating the RASEN tools is to:

- Define a generic model for the security risk assessment and risk management domain (SRAM model) and a generic security testing model that captures test specification and test management (ST model).
- Define the relationship between the generic SRAM model and the generic ST model, resulting in a generic risk assessment and testing model (SRAT model for short).
- Define adapters for the risk assessment tools for translating to and from the tool specific models to the generic SRAM and ST model and to define similar adapters for the testing tools.

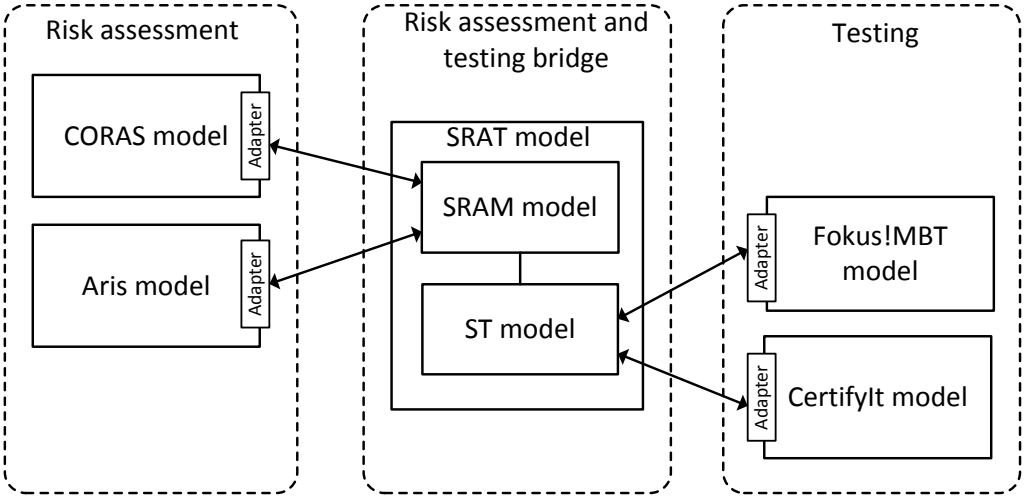

Figure 1 Overview of integration approach

As first step towards integration, the risk assessment tools and the testing tools should be able to handle subsets of the SRAT model. That is, the RASEN tools should be able to import and export data with respect to the terms and concepts of the SRAT model. In this deliverable, the foundations for such exports are defined.

3 Data Integration Models

In this section we define data models that serve as a basis for tool integration at the data level. The data models cover the following domains: testing, security testing, risk assessment, and security risk assessment. Put together, the models constitute the starting point for a generic model for security risk assessment and testing that will enable data exchange between the RASEN tools at the risk assessment level and the testing level.

To ensure that the data models can be used as a starting point for the conceptual model that will be developed later as part of the RASEN methodology, all data items have been defined (in English) based on relevant standards.

3.1 Generic Security Testing Model (ST Model)

The generic security testing model (ST model) defines the basic artifacts and data structures that are necessary to describe the data that are provided for data exchange by the RASEN security testing tools (STT and STMT). The data structures and artifacts are defined following established testing standards like IEEE 829[4], ISO/IEC/IEEE 29119[6], the ISTQB Glossary of testing terms [9], and the UML Testing Profile (UTP)[17]. While Section 3.1.1 describes the data structures for the basic testing terms and concepts, Section 3.1.2 describes their aggregation to test documents that are mainly inspired by IEEE 829. Section 3.1.3 finally describes refinements towards security testing.

3.1.1 Basic Testing Concepts

The data structures in this section describe the basic testing concepts and their relation to each other. The basic testing concepts are mainly inspired by the ISTQB Glossary of testing terms [9].

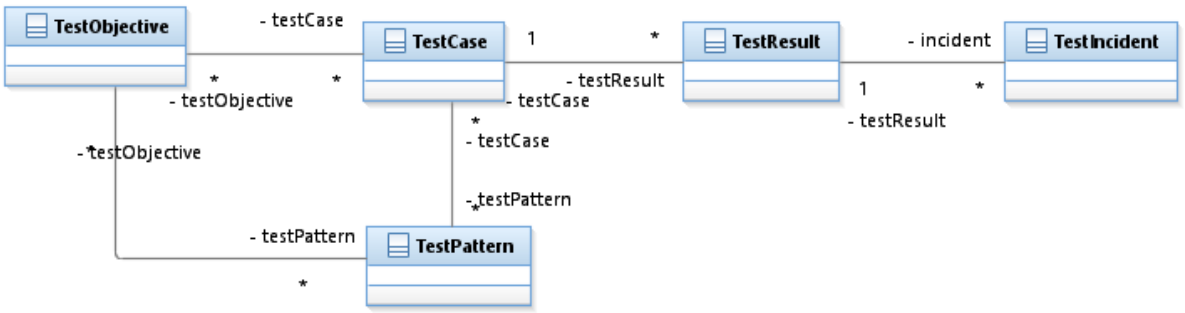

Figure 2 The basic testing concepts

- **Test Objective:** A (mainly textual) specification of what shall be tested (UTP [17]). Other standards refer to this concept as test requirement (ISTQB [17]) or test condition (ISO/IEC/IEEE 29119 [6]).
- **Test Case:** A set of input values, execution preconditions, expected results and execution postconditions, developed for a particular objective or test condition, such as to exercise a particular program path or to verify compliance with a specific requirement (ISTQB [17])
- **Test Result:** The consequence/outcome of the execution of a test. It includes outputs to screens, changes to data, reports and communication messages sent out (ISTQB [17]).
- **Test Incident:** Any event occurring during testing that requires investigation (ISTQB [17]).
- **Test Pattern:** An artefact that specifies a set of best practices to achieve dedicated test objectives in the context of a certain testing problem. Just as design patterns capture design knowledge into a reusable medium, test patterns capture testing knowledge into a reusable medium.

3.1.2 Aggregated Test Artifacts (Testing Documents)

The artefacts in this section describe aggregations of the basic testing concepts. These aggregations are used during a test process to transfer related data transferred from one process step to the next. The definition of the testing documents is taken from IEEE 829 [4] and ISO/IEC/IEEE 29119 [6].

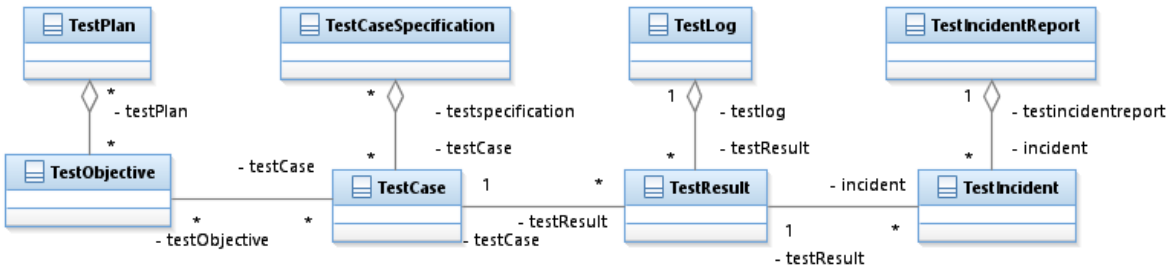

Figure 3 The main testing documents and their relation to the basic testing concepts

- **Test Plan**: A management planning document that shows: How the testing will be done, who will do it, what will be tested, how long it will take, what the test coverage will be, i.e. what quality level is required (IEEE 829 [4], ISO/IEC/IEEE 29119 [6]).
- **Test Case Specification**: Specifying the test data for use in running the test conditions identified in the Test Design Specification (IEEE 829 [4], ISO/IEC/IEEE 29119 [6]).
- **Test Log**: Recording which test cases were run, who ran them, in what order, and whether each test passed or failed (IEEE 829 [4], ISO/IEC/IEEE 29119 [6]).
- **Test Incident Report**: detailing, for any test that failed, the actual versus expected result and other information intended to throw light on why a test has failed. The report consists of all details of the incident such as actual and expected results, when it failed, and any supporting evidence that will help in its resolution. The report will also include, if possible, an assessment of the impact of an incident upon testing (IEEE 829 [4], ISO/IEC/IEEE 29119 [6]).

3.1.3 Security Testing Concepts

The data structures in this section describe the refinements that are necessary to adapt the basic testing concepts to the context of security testing.

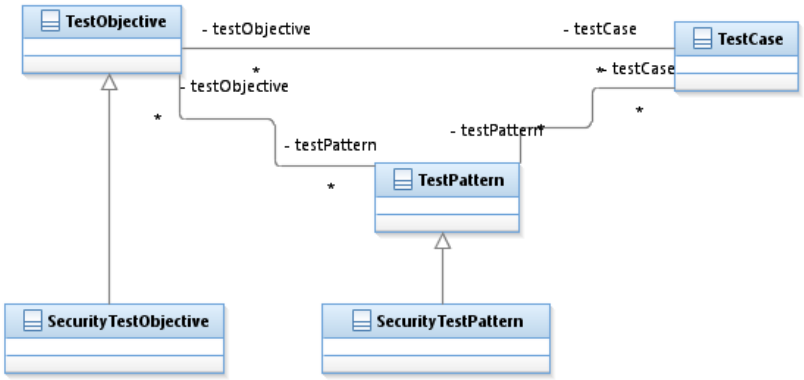

Figure 4 Basic security testing concepts

- **Security Test Pattern:** a software security test pattern is a recurring security problem, and the description of a test case that reveals that security problem, that is described such that the test case can be instantiated many times over, without ever doing it the same way twice. Just

as design patterns capture design knowledge into a reusable medium, software security test patterns capture security-testing knowledge into a reusable medium.

- **Security Test Objective:** a security test objective is a test objective specialized towards testing security properties like confidentiality, integrity or availability of a system.

3.2 Generic Model for the Risk Assessment and Risk Management (SRAM Model)

In this section we define a generic model for risk assessment and risk management. The definitions of the main terms are mainly based on the ISO31000 standard for risk management.

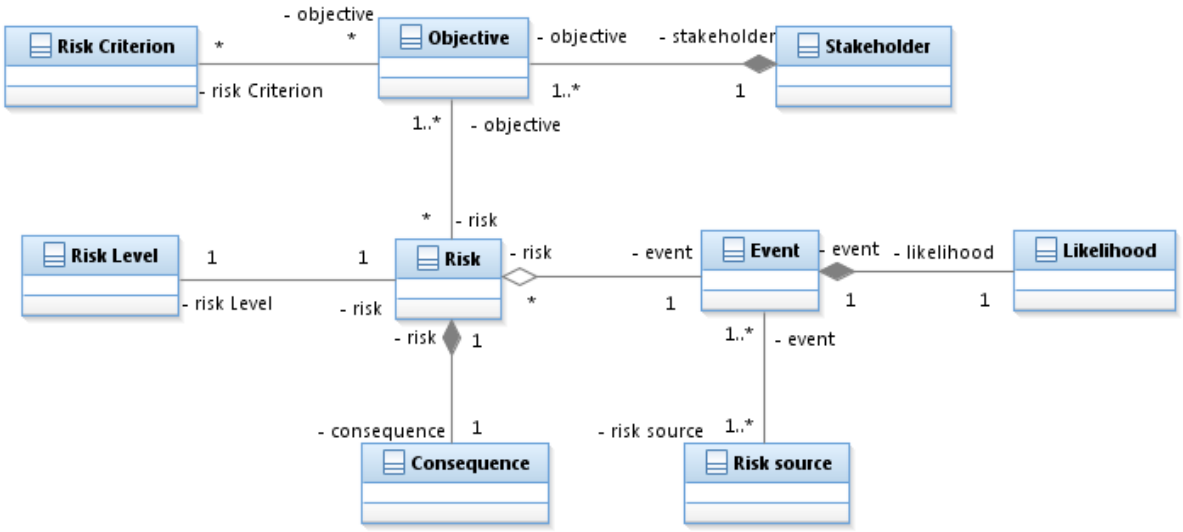

Figure 5 Generic model for risk assessment

- **Risk:** Risk is the combination of the consequences of an event with respect to an objective and the associated likelihood of occurrence (adapted from [7]).
- **Objective:** An objective is something the stakeholder is aiming towards or a strategic position it is working to attain (adapted from [18]).
- **Risk Source:** Risk source is an element, which, alone or in combination, has the intrinsic potential to give rise to risk [7].
- **Stakeholder:** Stakeholder is a person or organization that can affect, be affected by, or perceive themselves to be affected by a decision or activity [7].
- **Event:** Event is the occurrence or change of a particular set of circumstances [7].
- **Likelihood:** Likelihood is the chance of something happening [7].
- **Consequence:** Consequence is the outcome of an event affecting objectives [7].
- **Risk Criterion:** A risk criterion is the term of reference against which the significance of a risk is evaluated [7].

3.3 Security Risk Assessment Concepts

In this section, we refine the generic model for risk assessment into the domain of security. The definitions are mainly based on the ISO 27000 standard on information security.

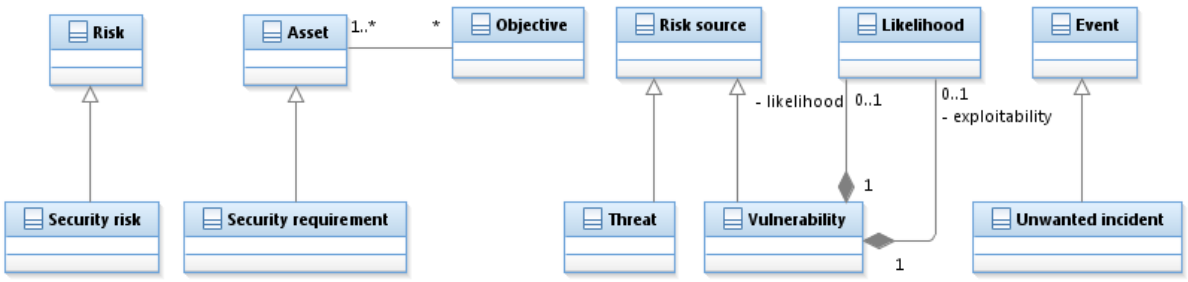

Figure 6 Generic model for security risk assessment

- **Asset:** Asset is anything that has value to the stakeholders (adopted from[8]).
- **Security Requirement:** Security requirement is a specification of the required security for the system (adopted from[18]).
- **Security Risk:** Security risk is a risk caused by a threat of exploiting a vulnerability and thereby violating a security requirement.
- **Unwanted Incident:** Unwanted incident is an event representing a security risk.
- **Threat:** Threat is potential cause of an unwanted incident [8].
- **Vulnerability:** Vulnerability is weakness of an asset or control that can be exploited by a threat [8].

3.4 Synthesis

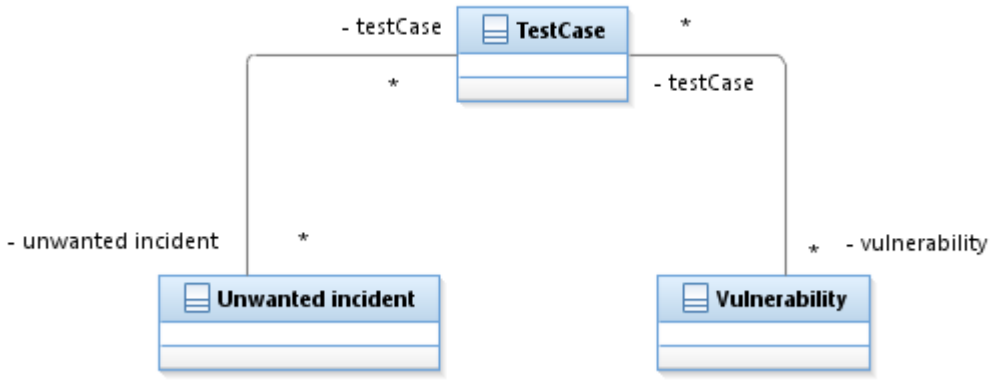

Figure 7 Generic model for security risk assessment and testing

In this section we relate the concepts from the testing domain to the concepts of the security risk assessment domain. In order to do this, it is not necessary to add any new concepts to the conceptual model, but associations have to be specified between the testing concepts and the security risk assessment concepts.

Currently, as shown in Figure 7, we see one main link between the testing and the security risk assessment domain going from *TestCase* (from the testing domain) to *Vulnerability* (in the security risk assessment domain). This is because the main purpose of security testing is to discover vulnerabilities. Other relationships, such as the risks that are being targeted by a test case can be derived by transitivity by following the associations of the vulnerability.

We have also added an association between *TestCase* and *Unwanted Incident*. This is because all potential vulnerabilities may not necessarily be known in advance before the tests are executed. In this case it will not be possible to link the testing model to the risk assessment model without introducing "dummy" vulnerabilities. In order to avoid this, we have therefore introduced a link between TestCase and Unwanted Incident. Note that if the potential vulnerabilities that are targeted by a test case *are* known, then the relationship to the unwanted incident can be deduced by transitivity.

4 Integration Requirements

The integration requirements are specified by first identifying the integral integration use cases that are addressed in the project. An integration use case describes the use of the RASEN tools by means of scenarios in which two or more tools are involved. The use case definition is based on the following use case template that provides a structure to unify the specification. Mandatory fields are the name of the use case, the actors that are involved, the precondition of the use case (i.e. the things that necessarily must have happened, so the use case scenario can be successfully started), the postcondition of the use case (i.e. the results after the scenario has been successfully applied) and the data that are exchanged between the tools.

Name	**The name of the integration use case**
ID	A unique identifier of the integration uses case. It starts with IUC_ and is expanded with a subsequent number (e.g. IUC_01)
Actors	The actors that are referred to in the scenario (supposed to be the users and the tools that need to be integrated)
Precondition	The precondition that need to be enabled when the scenario is initiated.
Postcondition	The postcondition that describes the result of the scenario.
Scenario	The scenario that describes the individual actions taken by the actors. 1. *Actor* :*Action* 2. *Actor* :*Action*
Data exchanged	The data that are exchanged during the integration use case

Table 3 – Integration use case template

4.1 Security Test Derivation Integration Requirements

Name	**Preparation for security testing based on risk assessment results**
ID	**IUC_01**
Actors	Risk Analyst (RA), Security Tester (ST), Security Risk Assessment Tool (SRAT), Security Testing Tool (STT)
Precondition	A risk assessment model with likelihood estimates exists in the **SRAT**
Postcondition	A prioritized list of test-scenarios exists in the **STT**

Scenario	1. **RA** loads the *risk assessment model* in the **SRAT** tool 2. **RA** annotates *the risk assessment model* in the **SRAT** tool with annotations (e.g. testability and uncertainty) that are used to guide the test-scenario prioritization. 3. **RA** starts the test scenario prioritization in the **SRAT** tool 4. **SRAT** automatically identifies potential test scenarios in the annotated risk assessment model and generates *a prioritized list of test scenarios*. 5. **RA** selects the test scenarios that should be tested and uses the **SRAT** to export these into a generic risk-assessment and testing model which contains the prioritization values. 6. **ST** imports the *generic risk assessment and testing model* (containing the prioritized test scenarios) into the **STT**. 7. **RA** constructs a risk assessment model and estimates likelihood values
Data exchanged	Data involved in the scenario: Risk assessment model, annotated risk assessment model, list of prioritized test scenarios, generic risk assessment and test model. Data exchanged: Generic risk assessment and testing model

Table 4 – Use case: Preparation for security testing based on risk assessment results

Name	**Test identification on basis of Attack Pattern + Test Pattern combination**
ID	**IUC_02**
Actors	Risk Analyst (RA), Security Tester (ST), Risk Assessment Tool (SRAT), Security Testing Tool (STT), Attack Pattern and Test Pattern Data Base (PDB)
Precondition	Risk assessment is done, test patterns are assigned to the elements of the risk assessment
Postcondition	Security test cases that are specific to the risks addressed in the risk assessment have been generated.
Scenario	1. **RA** loads the *risk assessment model* in the **SRAT** tool. 2. **RA** and **ST** identify test pattern in the **PDB** with respect to the *risk assessment model* in the **SRAT** tool. 3. **SRAT** calculates risk-based priority values for Vulnerabilities, Threat Scenarios or Treatments. 4. **STT** gets vulnerabilities, threat scenarios, mitigations and related test pattern from the **SRAT** and the **PDB** 5. **STT** generates a certain amount of security test cases with respect to the vulnerabilities, threat scenarios, mitigations, test pattern and the related risk values
Data exchanged	Vulnerabilities, Threat Scenarios and/or Treatments + related Test Pattern + Risk-based Priority Values

Table 5 – Use case: Test identification on basis of Attack Pattern + Test Pattern combination

Name	**Test prioritization for MBTG**
ID	**IUC_03**
Actors	Security Tester (ST), Risk Assessment Tool (SRAT), Security Testing Tool (STT)
Precondition	Risk assessment is done
Postcondition	Security test cases that are specific to the risks addressed in the risk assessment have been generated.
Scenario	1. **STT** gets vulnerabilities from the **SRAT** 2. **ST** assigns test purposes to the vulnerability in the **STT** 3. **STT** generates a certain amount of security test cases with respect to risk values
Data exchanged	Vulnerabilities + calculated Risk Values

Table 6 – Use case: Test prioritization for MBTG

4.2 Security Test Result Aggregation Integration Requirements

Name	**Calculating Vulnerability Coverage**
ID	**IUC_04**
Actors	Security Test Management Tool (STMT), Security Risk Assessment Tool (SRAT)
Precondition	The results of a test run exist in form of a *TestLog* in the ***STMT***
Postcondition	Vulnerabilities in the **SRAT** provide information about the number of failed and passed related security test cases.
Scenario	1. **STMT** gets the Test Specification from the **STT** 2. **STMT** exports the *TestLog* containing *TestResults* and related *Vulnerabilities* into a *generic risk assessment and testing model*. 3. **SRAT** imports the generic risk assessment and testing model and displays the vulnerabilities with an overview on related security test cases and their results
Data exchanged	Test Specification, Test Log, Test Result that relate to a Vulnerability, generic risk assessment and testing model

Table 7 – Use case: Calculating Vulnerability Coverage

Name	**Updating vulnerability information based on test results**
ID	**IUC_05**
Actors	Risk Analyst (RA), Security Tester (ST), Security Risk Assessment Tool (SRAT), Security Testing Tool (STT), Security Test Result Aggregation Tool (STRAT)
Precondition	The results of a test run exist in form of a *TestLog* in the **STT**
Postcondition	Measurements related to *Vulnerabilities* (likelihood of existence, exploitability) are updated in the *risk assessment model* based on the test results.
Scenario	1. **STMT** exports the *TestLog* containing *TestResults* and related *Vulnerabilities* into a *generic risk assessment and testing model*. 2. **STRAT** imports the *generic risk assessment and testing model* containing the *TestResults* and updates the vulnerability measures based on the test results. 3. **SRAT** imports the updated *generic risk assessment and testing model* with updated vulnerability measures. 4. **SRAT** updates the risk picture based on the new vulnerability measurements
Data exchanged	Vulnerabilities, vulnerability measures, Test Results, generic risk assessment and testing model

Table 8 – Use case: Updating vulnerability information based on test results

Name	**Getting information about new vulnerabilities based on test results**
ID	**IUC_06**
Actors	Security Risk Assessment Tool (SRAT), Security Testing Tool (STT)
Precondition	The results of a test run exist in form of a *TestLog* in the **STT**
Postcondition	The new *vulnerabilities* identified in the **STT** exist in the *risk assessment model* of the **SRAT**
Scenario	1. **STT** exports the test results to a generic risk assessment and testing model which contains the new vulnerabilities that are identified. 2. **SRAT** imports the generic risk assessment and testing model and converts it into a tool specific risk assessment model containing the new vulnerabilities that were identified. 3. The user of the **SRAT** places the new vulnerabilities at the right location in the tool specific risk assessment model.
Data exchanged	TestLog, Vulnerabilities, Generic risk assessment and testing model, Tool specific risk assessment model, (location of vulnerability in system).

Table 9 – Use case: Getting information about new vulnerabilities based on test results

4.3 Legal Risk Analysis & Compliance Management Integration Requirements

Name	**GRC in ARIS**
ID	**IUC_07**
Actors	Compliance Manager (CM), InfoWorld (IW), Risk Management Tool (SRMT), Risk Assessment Tool (SRAT)
Precondition	Decision to ensure compliance
Postcondition	Model in **SRMT** : • Business processes under analysis (e.g., InfoWorld's), • Legal requirements (e.g. data protection) • Identified compliance risks
Scenario	1. **CM** models **IW's** business process in **SRMT** 2. **CM** models legal requirements in **SRMT** 3. **CM** identifies and models compliance risks for **IW** in **SRMT**
Data exchanged	Risks from **SRMT** are used as input for analysis in **SRAT**

Table 10 – Use case: GRC in ARIS

Name	**Compliance risk assessment**
ID	**IUC_08**
Actors	Risk Management Tool (SRMT), Risk Assessment Tool (SRAT), Compliance Manager (CM)
Precondition	Models in **SRMT**: • Business processes under analysis (e.g., InfoWorld's), • Legal requirements (e.g. data protection) • Identified compliance risks
Postcondition	• Risk value is assigned • Risk evaluation is carried out • Risk controls are assigned to risk

Scenario	1. **CM** uses **SRAT** to assign risk values to identified risks 2. **CM** uses **SRAT** to evaluate risks 3. **CM** uses **SRAT** to assign risk controls to identified risks
Data exchanged	Risks from **SRMT** are used as input for analysis in **SRAT**

Table 11 – Use case: Compliance risk assessment

Name	**Compliance implementation**
ID	**IUC_09**
Actors	Risk Management Tool (SRMT), Risk Assessment Tool (SRAT), Compliance Manager (CM)
Precondition	• Business process, legal requirements and risks are modelled in SRMT • Risk value is assessed in **SRAT**
Postcondition	Output of risk assessment (accept or treat risks) used to inform compliance process in **SRMT**
Scenario	1. **CM** uses output from **SRAT** to model compliant business process in **SRMT**
Data exchanged	Risk value, Treatments

Table 12 – Use case: Compliance implementation

5 Security Tool Box Design

This section outlines the initial design of the RASEN risk assessment and security testing toolbox. It starts with the analysis of the technical integration capabilities of the RASEN tools and defines the requirements on the integration interfaces for the main conceptual tools that have been defined in Section 2.3. Finally this section outlines the next steps to be processed to finally obtain an integrated risk assessment and security testing toolbox.

5.1 Security Testing Tool Integration Capabilities

This section briefly describes the security testing tools developed and provided by the RASEN partners, and defines the interfaces that should be implemented during the project to integrate each test generation technology into a common framework. This goal will include extension and optimization of existing test generation tools (to specifically address risk-based vulnerability testing for large-scale systems), and interface adaptation to be able to link the tools to each other to propose a fully integrated framework covering the RASEN testing approach described in Figure 1.

Within the RASEN project, testing tool providers are Smartesting, with the tool *CertifyIt for Security Testing*, and Fraunhofer FOKUS, with the tool *Fokus!MBT*. These tools are respectively introduced in Sections 5.1.1 and 5.1.2. Finally, Section 5.1.3 gives a brief synthesis about the testing tool integration aspects.

5.1.1 CertifyIt

Smartesting CertifyIt for Security Testing is a commercial tool suite that automatically generates security test cases based on security test patterns and a behavioral/environmental model of the system under validation [13]. Figure 8 – Smartesting CertifyIt for Security Testing Process, introduces the various artifacts that are used to generate security test scripts from security test patterns.

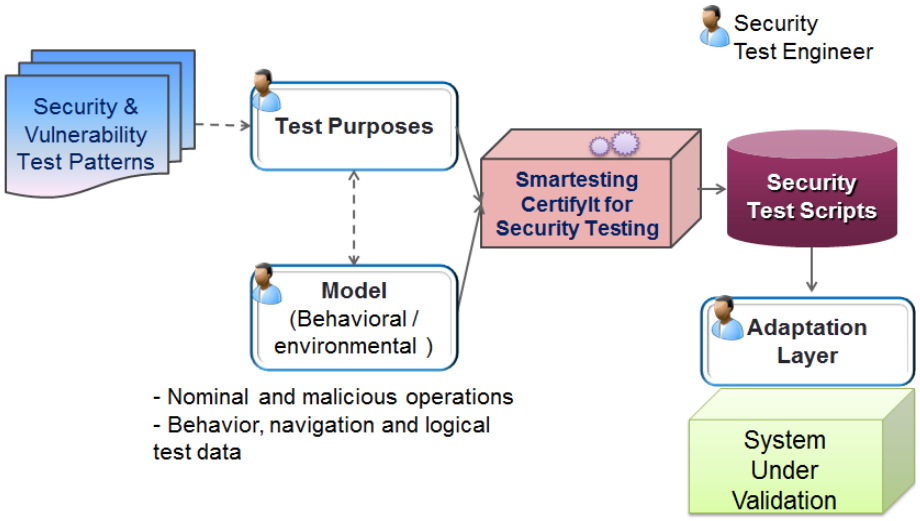

Figure 8 – Smartesting CertifyIt for Security Testing Process

Test purposes are a formal description of the test procedure directly interpreted by CertifyIt and composed with a test model (called model in the figure) to automatically generating security test scripts. The modeling activity produces a model based, on one hand, on the functional specifications of the application, and on the other hand on the test purposes which will be applied to it (keywords used in test purposes have to be modeled) - see [12] for more details.

Within the RASEN project, this initial approach will be used as a basis to perform Risk-Based Vulnerability Testing (RBVT), and will therefore be extended to:

1. take into account risk assessment to drive the test generation process;
2. support risk traceability throughout the testing process to provide relevant feedback to complete risk assessment;
3. support compositional security testing;
4. be integrated in the RASEN tool chain from Risk Analysis to Security Test results aggregation.

All these extensions will give rise to the RBVT approach, which is depicted in Figure 9. As shown in this picture, Smartesting CertifyIt for Security Testing Tool will be extended to specify security test cases and to generate the corresponding executable tests (STT), but Smartesting tooling will also embed the Test Derivation Tool (TDT) for supporting the derivation and prioritization of test cases based on the risk assessment, as well as the Security Test Management Tool (STMT) for managing the executable tests and their results.

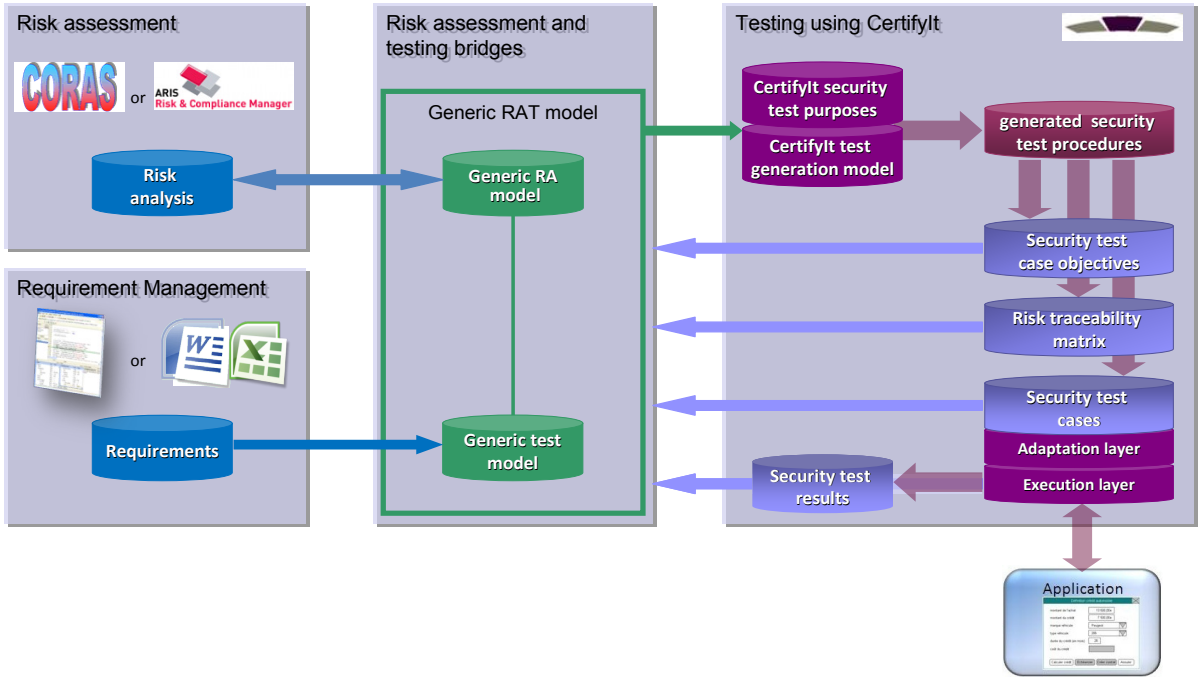

Figure 9: Smartesting RBVT global process

To implement this RASEN process based on Smartesting tooling, the following input data are required:

- The vulnerabilities to be targeted by the test cases in order to derive dedicated security test purposes that will define the strategy to be used by the test generation activity.
- The definition of security testing priorities from compositional risk assessment to be able to drive the test generation strategy.
- A generic test model including the behavioral relevant aspects of the application under test in order to derive a test generation model.
- An implementation of the application under test and the required access data (including logical test data and credentials for testing) to be able to execute the generated security test cases and to gather the test results.

To implement this RASEN process based on Smartesting tooling, the following output data can be proposed:

- The abstract test cases and corresponding executable test scripts, which include, for each generated test case, the test objectives (related to risk analysis) and the test structure (steps, input parameters, expected results).
- The execution results for each of the generated test cases that have been executed on the application under test, including verdict (Attack-Pass / Attack-fail / Functional-Pass / Functional-Fail / Inconclusive) and details related to the execution information.
- The risk traceability matrix for each of the generated test cases, with respect to risk analysis, as well as regarding the vulnerability repositories.

From a technical point of view, Smartesting provides an Eclipse-based modeler plug-in that checks that the model fits the Smartesting CertifyIt restrictions on UML/OCL, and exports a model in order to be used by the Smartesting CertifyIt tool to generate test cases. The test cases and all execution results are published into a proprietary XML file, which can be used (using dedicated Java API) to produce HTML test documentations (HTML publisher), test specification (UTP) or executable test scripts (Selenium, HTMLUnit, etc.). More generally, to acquire input data and to publish output data, RASEN development will make it possible to use structured input files (XML-based), Java libraries with dedicated API and Eclipse Plug-in with dedicated API to import and export artifacts from and to Smartesting tooling. These artifacts will provide sharing points with the other technologies of the RASEN framework.

5.1.2 Fokus!MBT

Fokus!MBT is a rich test modeling environment that simplifies the creation of the underlying test model by guiding the user through methodology-specific support. In RASEN it may serve as an implementation of the STT and STMT tools. In the project it will be extended so that it may additionally serve as TDT and TRAT tool. The Fokus!MBT is based on a service-oriented communication infrastructure of loosely coupled services, interoperating with each other in a distributed environment.

Fokus!MBT is based on UML and the UML Testing Profile (UTP). UTP represents the integrated data model used for data and information exchange among the services of the Fokus!MBT tool chain. This allows adapted services to interoperate with each other as well as it improves the communication flow among involved stakeholders. Fokus!MBT hides technical complexity of UML and all incorporated profiles from the test engineers, so that they can solely concentrate on their mission-relevant knowledge. The following modeling paradigms are incorporated in Fokus!MBT.

- **UML.** UML contributes elementary object-oriented concepts like packages, classes, properties and interfaces. It is also used to provide instances of the TestingMM with a package-oriented structure, like any MOF or UML model itself.
- **UTP.** UTP represents the core conceptual model of Fokus!MBT.
- **SysML.** The concise and intuitive possibility of expressing requirements inside UML models have been extracted from SysML. UML itself is not capable of modeling requirements natively.
- **Fokus!MBT profile.** Additional technical or conceptual concepts not provided by an official specification are incorporated into the proprietary Fokus!MBT profile.

Fokus!MBT is built on top of Eclipse Papyrus, an open source UML modeling environment, which, in turn, relies on the Eclipse Modeling Framework (EMF) and the Graphical Modeling Framework (GMF). In addition to diagrams, Fokus!MBT integrates an editor framework called *Core Editor*. The Core Editor is able to organize multiple so called *editor configurations*. An editor configuration consists of a number of editor pages. Each page visualizes certain task-relevant information encoded in the test model by using form-based widgets like trees, tables and lists. We call this kind of model authoring *form-based modeling*. It additionally adapts to the ModelBus framework [14] for control and process integration.

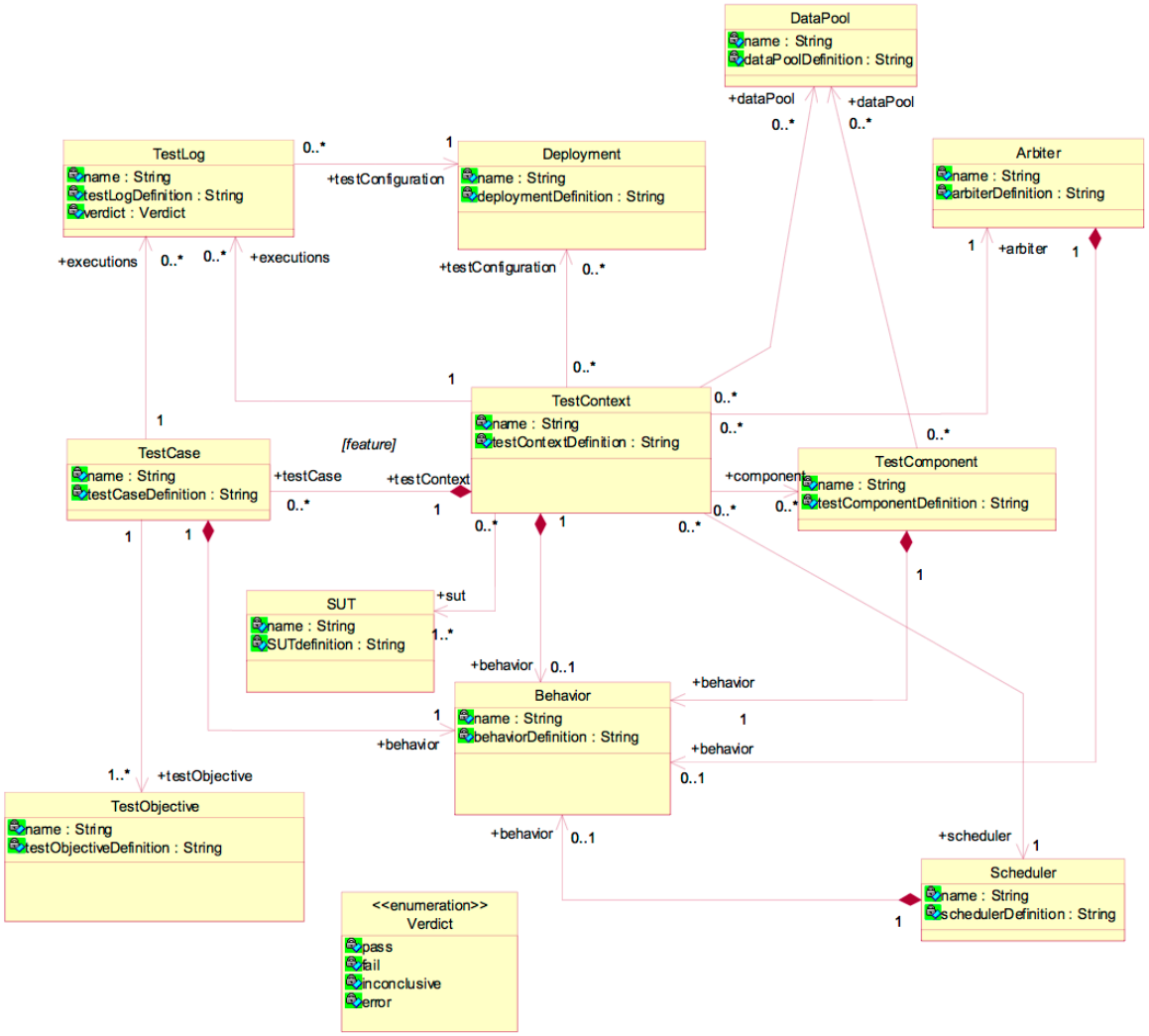

Figure 10: UTP meta-model

Fokus!MBT has no overall meta-model. It is based on standards like UML, UTP, SysML etc. The testing domain is at best described by the conceptual UTP meta-model that is shown in Figure 10: UTP meta-model. This meta-model will be the basis for the integration with the other tools of the RASEN project.

Fokus!MBT is neither a commercial nor an open source tool, but adheres to a proprietary licensing mechanism.

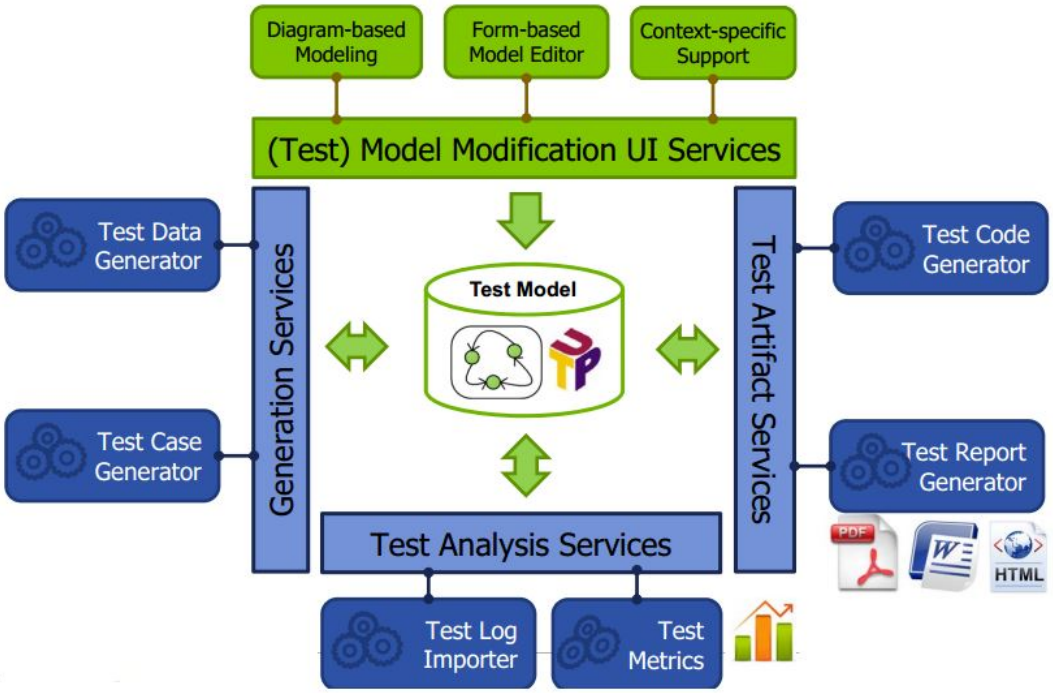

Figure 11: Fokus!MBT global services

Fokus!MBT 0.8 is based on Eclipse Indigo (3.7), Papyrus 0.8.2, UML 3.0.0 (i.e., UML 2.2 specification). The next version (0.9) will be based on Eclipse Kepler (4.3), Papyrus 0.10, UML 4.0.0 (i.e., UML 2.4.1 specification).

5.1.3 Synthesis

This section has presented the capabilities of the tools *CertifyIt for Security Testing* and *Fokus!MBT,* respectively provided by Smartesting and Fraunhofer FOKUS. It introduced the initial design of these tools and the planned extension to address RASEN objectives and integration into the RASEN risk assessment and security testing toolbox. This capability analysis has shown that the RASEN testing tooling is in line to be used in isolation, each tool offering its own services to generate risk-based security test cases and risk-related testing report, but also in collaboration. Indeed, the input and output format supported by the tools are (or will be) focused on UML2 standard and UML2 profiles such as UTP or SysML. These standards allow the tools to share the same testing artefacts, to exchange testing results and to interoperate to manage a test campaign. Moreover, from a technical point of view, the tools can be integrated into the Eclipse architecture, which makes it possible to produce a fully integrated testing framework to support the RASEN process.

5.2 Security risk assessment and management integration capabilities

This section describes the risk assessment tools that will be used as input to the RASEN project and that will be developed further within the project. Within the RASEN project, the main purpose of the RA tools will be to (1) provide a risk assessment model that can be used as a basis for test identification and prioritization, and (2) receive and update risk assessment model based on the test results. The two risk assessment tools of the RASEN project are the CORAS tool (described in Section 5.2.1) and the ARIS tool (described in Section 5.2.2).

5.2.1 CORAS Risk Assessment Tool

The CORAS risk assessment tool is intended to support documentation of risk assessment results. More specifically, the tool is a graphical diagram editor for specifying CORAS risk assessment diagrams.

The official version of the tool is freely available (http://coras.sourceforge.net/downloads.html) as a standalone tool based on Eclipse. The tool is open-source and released under the Eclipse Public License 1.0.

If needed, the CORAS tool can also be made available as an Eclipse plugin. In this case, the tool depends on the following plugins, which must be part of the Eclipse platform:

EMF 2.5, GMF 2.2, Epsilon Corse 0.8.9, Epsilon EMF/GMF Live Validation 0.8.9.

The CORAS meta-model

The risk assessment documents specified in the CORAS tool are stored in the form of a CORAS risk assessment model conforming to the CORAS meta-model. This meta-model is defined in Ecore, thereby enabling data integration on the basis of EMF.

A CORAS risk model consists of a set of diagrams which may be of different kinds. Each diagram describes a graph consisting of *elements* and *relations* between the elements. Both elements and relations may be of different kinds and they may have attributes such as likelihood values or consequence values.

The core CORAS meta-model is shown in Figure 12 - Figure 14. This meta-model will be the basis for the integration with the other tools of the RASEN project. However, it is expected that the meta-model may be changed during the course of the project as new requirements are identified.

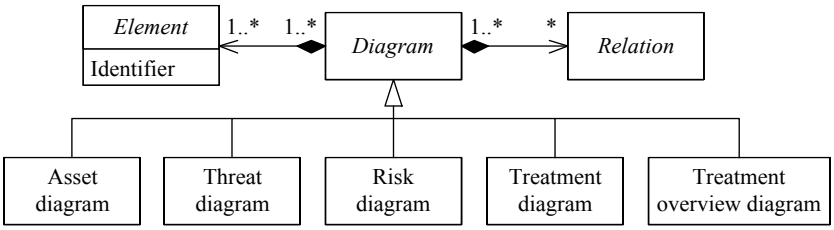

Figure 12 CORAS meta-model - diagrams

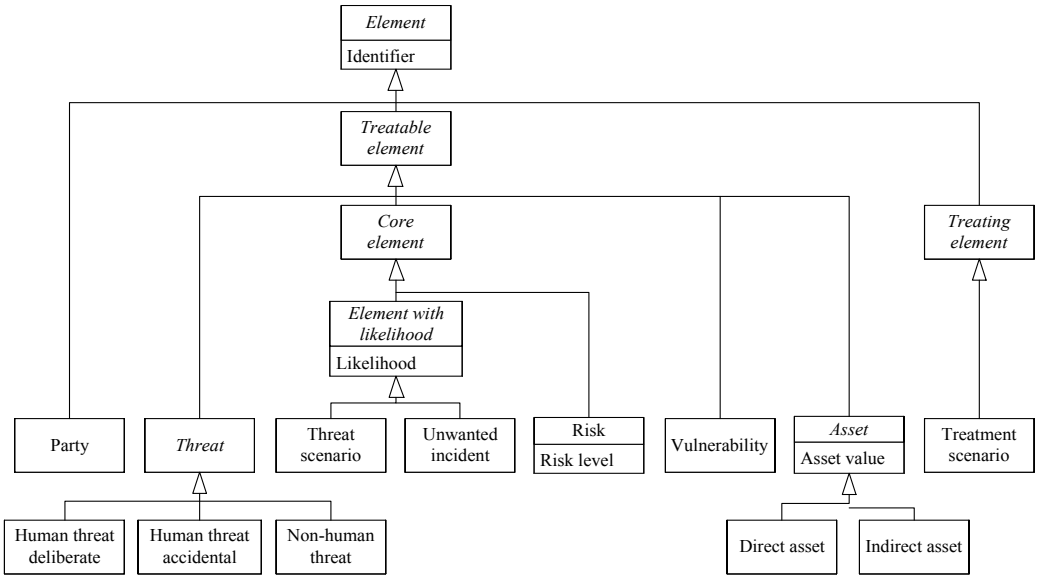

Figure 13 CORAS meta-model - elements

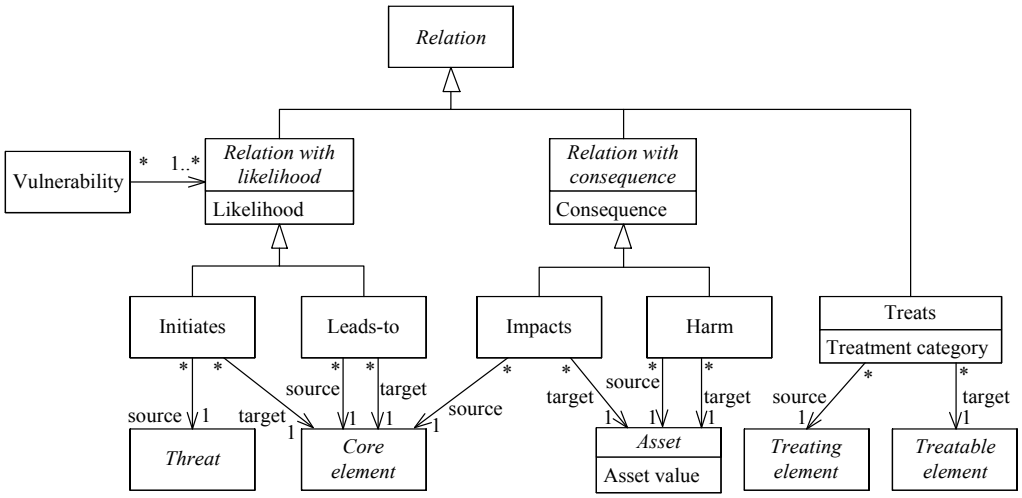

Figure 14 CORAS meta-model - relations

Interfaces

In the setting of risk-based testing, the CORAS tool must support test identification and test prioritization and store this information in a format that can be imported by testing tools. In the setting of test-based risk assessment, the CORAS tool must be able to import a generic risk assessment and testing model and convert this into a CORAS risk assessment model (described in the previous section). Hence, the two main interfaces of the tool are:

- **Export** the CORAS risk assessment model by converting into a *generic risk assessment and testing model* that contains the results of the risk-based test identification and test prioritization.
- **Import** a generic risk assessment and testing model and covert this into a CORAS risk assessment model.

The CORAS tool assumes that aggregation of the test results to the risk assessment model is performed before the generic risk assessment and testing model is imported into the CORAS tool.

5.2.2 ARIS Risk & Compliance Manager

The ARIS Risk & Compliance Manager (ARCM) is a role based workflow software system that supports the flexible implementation and efficient operation of an enterprise-wide compliance and risk management system. Elements like forms, control processes, risk documentation, reports and in particular those elements with a close relation to the modeling aspects are stored in the ARCM database. On the other hand elements like risk, diagrams, business models, and business control diagrams are located in the business process database of the ARIS Business Server (ABS).

The documentation of business processes and functions using models in ARIS brings a variety of advantages like consistency, reduction of complexity, reusability, potential for evaluation, integrity, etc. However, these advantages can only be fully utilized if methodological and functional rules and conventions for modeling in ARIS Business Architect (ABA) are adhered to.

In ARIS system data imports and exports are realized with the help of customer-specific report-scripts which map elements of the input/output to the internal object models. Hereby technically inputs from XML files like GRC-XML [16] can be used to be imported into the GRC modeling framework but technically any other input is feasible.

GRC-XML is a Risk and Control Taxonomy, combining a family of languages for Governance, Risk, and Compliance information sharing, integration, and communication. The framework can provide [16]:

- A basis for corporations to standardize on a common language of Risk and Control

- The ability to compare the results of risk and control initiatives between companies
- An integrated platform for corporation between various GRC systems

For the RASEN project we envision to use a combination of the CORAS tool (cf. Section 5.2.1) and the ARCM tool in the following style:

In the first step, the CORAS tool is used for an initial risk assessment according to the CORAS methodology. Once this is successfully done, the risks discovered along with risk estimates and other annotations are exported into a unified intermediate language (which could possibly be GRC-XML) and transferred into ABS, the central repository server. Here, this information is stored together with the specific risk (gained by the CORAS approach) and used as a perfect matching of the initial situation when the CORAS risk assessment took place.

5.2.3 Synthesis

Initially, our plan is to integrate risk assessment with the testing tools. This is to ensure that our integration approach is not specific to any particular risk assessment tool. However, we will also investigate the possibility of integrating the two risk assessment tools on the risk assessment side. This is because the tools have slightly different purposes. That is, the ARIS tool is more focused on risk management, while the CORAS tool is more focused on risk assessment. This means that it may be possible to use the CORAS tool as part of the risk management process supported by the ARIS tool.

5.3 Initial Integration Interface Definition

This section specifies the integration interfaces for the major conceptual tools that have been defined in Section 2.3. The interface definition is based on the integration use cases from Section 4 and denotes required and provided data for each of the conceptual tools. The data are specified in terms of the conceptual model from Section 3.

Conceptual Tool	**Security Risk Assessment Tool (SRAT)**
Concrete Tools	CORAS
Requires	**Generic testing model**[1] (see IUC_04), Test Log, Test Result that relates to a Vulnerability, Risk (see IUC_07, IUC_08), Updated generic risk assessment and testing model based on test results (see IUC_05, IUC_06)
Provides	**Generic risk assessment model** (see IUC_01), Vulnerabilities, Threat Scenarios, Treatments, Test Pattern + Risk-based Priority Values (see IUC_02, IUC_09), Vulnerabilities + Risk Values (see IUC_03, IUC_6)

Table 13 – Security Risk Assessment Tool (SRAT)

Conceptual Tool	**Security Risk Management Tool (SRMT)**
Concrete Tools	ARIS Risk Manager

[1] Please note that the data structures given in bold letters are covering all following structures for the given section.

Requires	Risk Values, Treatments (IUC_09)
Provides	Risks (see IUC_07, IUC_08)

Table 14 – Security Risk Management Tool (SRMT)

Conceptual Tool	**Security Testing Tool (STT)**
Concrete Tools	Smartesting CertifyIt (Section 5.1.1) Fokus!MBT (Section 5.1.2)
Requires	**Generic risk assessment model** (see IUC_01), Vulnerabilities, Threat Scenarios and/or Treatments + related test pattern + risk-based priority values (see IUC_02), calculated risk values (see IUC_03)
Provides	Test Specification (see IUC_04)

Table 15 – Security TestingTool (STT)

Conceptual Tool	**Security Test Management Tool (STMT)**
Concrete Tools	Smartesting CertifyIt (Section 5.1.1) Fokus!MBT (Section 5.1.2)
Requires	Test Specification (see IUC_04)
Provides	**Generic testing model** (see IUC_04), Generic testing model (see IUC_05), Test Log, Test Result that relate to a Vulnerability

Table 16 – Security Test Management Tool (STMT)

6 Conclusion and Outlook

This deliverable specifies the initial integration requirements and the initial integration design of the RASEN risk assessment and security testing tool box. The deliverable starts with a short overview on tool integration approaches and platforms. Afterwards the deliverable introduces the tools and artifacts that are seen from a conceptual point of view, necessary to address the main RASES use cases. Integration is approached from two directions. Generic data models, so called conceptual models, model the data and artifacts that are relevant for security testing and security risk assessment and security risk management. These models provide the basis for the definition of so called integration use cases. An integration use case describes a cross-tool data exchange scenario and is used to identify the data to be exchanged between the RASEN tools. The final section outlines the integration design. It starts with the description of the tools that are already available from the partners. Based on the integration use cases and the tool descriptions, finally a list of tool integration interfaces are specified. After having defined the initial requirements and the initial integration design, the integration task will proceed as follows.

1. Create deployable generic security testing (ST model) and security risk assessment and management model (SRAM model). These models will be designed in UML.
2. Create a deployable generic risk assessment and testing model (SRAT model) on basis of the ST model and the SRAM model.
3. Instantiate a common XML-based exchange format on basis of the SRAT model.
4. Develop export/import adapter for the relevant instantiations of the conceptual tools (concrete tools) with respect to the interfaces in Section 5.3.
5. Identify additional integration use cases and update the models if necessary.

On basis of the models RASEN will enable traceability between risk assessment artifacts and testing artifacts. Traceability has been especially used as a key concept in the domain of Safety. Traceability there is traditionally based on a Traceability Information Model (TIM) that allows understanding the relationships around various Safety Requirements. Such a model enables the realization of different types of traces [15].

- Structural traces: traces that are contained inside the models that have to be traced.
- Explicit traces: traces that are manually created and connect between models.
- Implied traces: transitive associations, indirect associations.

RASEN will integrate a major part of its tooling in the well-known open source tool integration platform eclipse (www.eclipse.org). Smartesting, SINTEF, and Fraunhofer are currently using this platform. This allows for deploying the project results into a large community of open source security and software development providers. Particular effort will be made to define open interfaces and data formats for communication between the various tools, thus enabling the integration of open source tools (developed by SINTEF and Fraunhofer), and commercial tools (developed by Software AG, Fraunhofer and Smartesting), and also enabling individual tools to be replaced in the future by tools that satisfy the interface definition.

References

[1] CDO project: Eclipse Connected Data Objects.www.eclipse.org/cdo/, as of date 15.02.2013

[2] EMF Store project: Model repository for EMF.www.eclipse.org/emf-store/, as of date 15.02.2013

[3] IBM: IBM Rational Jazz technology platform.www.ibm.com/software/rational/jazz/, , as of date 15.02.2013

[4] IEEE: IEEE Standard for Software and System Test Documentation (IEEE 829-2008), ISBN 978-0-7381-5747-4, 2008.

[5] IEEE: IEEE Standard Glossary of Software Engineering Terminology (IEEE 610.12-1990), ISBN 1-55937-067-7, 1990.

[6] IEEE: IEEE 29119 Software and system engineering - Software Testing Part 1: Concepts and definitions, 2012.

[7] International Standards Organization. ISO 31000:2009 (E), Risk management – Principles and guidelines, 2009.

[8] International Standards Organization. ISO 27000:2009 (E), Information technology – Security techniques – Information security management systems – Overview and vocabulary, 2009.

[9] ISTQB: ISTQB Glossary of testing terms version 2.2.http://www.istqb.org/downloads/finish/20/101.html, as of date 19.03.2013.

[10] Itemis: The CreMA tool.http://www.guersoy.net/knowledge/crema, as of date 15.02.2013

[11] V. Katta, T. Stålhane: A Conceptual Model of Traceability for Safety Systems, proceedings of Springer-Verlag

[12] F. Lebeau, B. Legeard, F. Peureux, A. Vernotte: Model-based vulnerability testing for web applications. SECTEST 2013, the Fourth International Workshop on Security Testing, affiliated with ICST 2013, Luxembourg, March 22, 2013.

[13] B.Legeard, F. Lebeau, F. Bouquet, J. Botella, J. F. Capuron, F. Schadle: Model-based testing of cryptographic components - lessons learned from experience. Proceeding of the IEEE Inter. Conf. on Software Testing, Verification and Validation. ICST 2013, Luxembourg, March 18-22, 2013.

[14] Modelbus team: Model-driven tool integration framework.www.modelbus.org, as of date 15.02.2013

[15] S. Nejati, M. Sabetzadeh, D. Falessi, L. Briand, T. Coq: A SysML-based approach to traceability management and design slicing in support of safety certification: Framework, Tool Support, and Case Studies. http://simula.no/publications/Simula.simula.193/simula_pdf_file, as of date 15.02.2013

[16] OCEG: GRC-XML Version 1.0.http://www.oceg.org/resource/grc-xml-version-10, as of date 15.02.2013

[17] OMG: UML testing profile version 1.1 (formal/2012-04-01). http://www.omg.org/spec/UTP/1.1, as of date 19.03.2013

[18] F. John Reh. Glossary of business management terms and abbreviations. http://management.about.com/cs/generalmanagement/g/objective.htm, as of date 19.04.2012.

[19] I. Thomas, B. Nejmeh: Definitions of tool integration for environments. Software IEEE, 1992

[20] A. Wassermann: Tool integration in software engineering environments, Software Engineering Environment. Lecture Notes in Computer Science, 1990