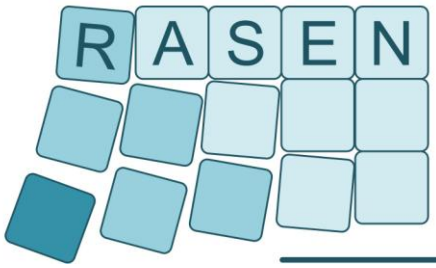

Compositional Risk Assessment and Security Testing of Networked Systems

Deliverable D4.3.1

Tools for compositional risk-based security testing

Documentation and Installation Guide

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:	WP4
Deliverable number:	D4.3.1
Nature of deliverable:	Prototype& Report
Dissemination level:	PU/PP/RE/CO
Internal version number:	1.0
Contractual delivery date:	2013-09-30
Actual delivery date:	2013-09-30
Responsible partner:	Smartesting

Contributors

Editor(s)	Fabien Peureux (SMA)
Contributor(s)	Julien Botella (SMA), Fabien Peureux (SMA), Martin Schneider (FOKUS), Fredrik Seehusen (SINTEF)
Quality assuror(s)	Frank Werner (SAG)

Version history

Version	Date	Description
0.1	13-06-27	TOC proposition
0.2	13-08-23	SMA contribution
0.3	13-08-30	SINTEF contribution
0.4	13-09-17	FOKUS contribution
0.5	13-09-23	Completed version for internal review
0.6	13-09-24	SAG review updates
1.0	13-09-27	Final updates – version ready to be delivered

Abstract

This report provides some basic information about the tools of the WP4 testing tool prototypes within deliverable D4.3.1 due at project month M12. The delivered tools are CORAS from SINTEF, CertifyIT from Smartesting, and RISKTest, Fuzzino library andBehavioral Fuzz Test Case Generator from Fraunhofer FOKUS.

Keywords

Security, security risk assessment, security testing, compositional test techniques, tool support, prototype

Executive Summary

The overall objective of RASEN WP4 is to develop techniques for how to use risk assessment as guidance and basis for security testing, and to develop an approach that supports a systematic aggregation of security testing results. The objective includes the development of a tool-based integrated process for guiding security testing deployment by means of reasonable risk coverage and probability metrics.

The overall objective of RASEN WP4 is to develop tools and techniques to support test-based and compositional security risk assessment. Deliverable D4.2.1 presents the WP4 techniques that were developed during the first year of the projects, while D5.3.1 presents the methodologies that the WP4 techniques should support. This report accompanies the D4.3.1 prototype deliverable, and gives an overview and introduction to the five tools of that deliverable.

The tools introduced in this deliverable are the following:

- The CORAS tool from SINTEF, supporting model-driven risk analysis.
- The CertifyIT tool from Smartesting for model-based security testing.
- The RISKTest tool from Fraunhofer FOKUS, which is a tool integration platform for risk-based security testing.
- The Fuzzino library from Fraunhofer FOKUS, which generates test data for fuzz testing.
- The Behavioral Fuzz Test Case Generator from Fraunhofer FOKUS.

Table of contents

1 Introduction

This D4.3.1 deliverable introduces documentation and installation guides of RASEN WP4 tool prototypes for deriving test cases from risk assessment results. The reader is referred to deliverable D4.2.1 for an overview of the WP4 research results after the first year of the RASEN project.

The prototypes presented in this report serve as a part of the RASEN tool-box for security risk assessment and security testing that is developed in the context of WP5. WP5 defines the common RASEN data meta-model that will be used for integrating the tools by facilitating the communication between them. An important part of the RASEN prototype development is therefore to provide support for exporting and importing data to and from the common RASEN data model.

The tools presented in the next two sections are CORAS, Smartesting CertifyIt, RISKTest, Fuzzino library and Behavioral Fuzz Test Case Generator. Section 2 gives a brief overview of the main characteristics of the tools, while Section 3 introduces the tools in some more details, by giving installations and user guidelines, and explaining the planned and current status for the development of tool features relevant in WP4. Finally, Section 4 provides a short conclusion.

2 Characteristics of the identified WP4 tools

This chapter introduces the identified RASEN tools to address vulnerability test case generation from risk assessment results. They form part of the RASEN tool-box for compositional risk assessment and security, which will be defined and delivered within the WP 5 of the project.

2.1 CORAS

General information	
Name	CORAS tool
Provider	SINTEF
Topic addressed	Model-based risk assessment
Description	The tool is based on Eclipse and the GMF/EMF framework, but it is distributed as stand-alone tool.
License	Eclipse Public License v1.0 (http://www.eclipse.org/legal/epl-v10.html)
Website	http://coras.sourceforge.net
Technical information	
Download site	http://coras.sourceforge.net/downloads.html
OS	Tested on Windows. A non-tested distribution is also available for Linux.
Technology environment	The tool needs Java.
Other dependencies	None.
Additional information	
Known issues/risks	None.
Additional useful information	None.

2.2 Smartesting CertifyIt

General information	
Name	Smartesting CertifyIt
Provider	Smartesting
Topic addressed	Model-Based Testing solution
Description	The tool is composed by a Rational Software Architect plug-in for modeling activities, and a stand-alone Java application for the Test Generation
License	Proprietary.
Website	http://www.smartesting.com
Technical information	
Download site	http://www.smartesting.com
OS	Win or Linux
Technology environment	Rational Software Architect v8.0.x and v8.5.x, EMF compliant, JAVA 1.6 and 1.7 compliant
Other dependencies	None.
Additional information	
Known issues/risks	None.
Additional useful information	None.

2.3 RiskTest

General information	
Name	RiskTest 0.1 as an extension to Fokus!MBT
Provider	Fraunhofer FOKUS
Topic addressed	Tool Integration Platform for Risk-Based Security Testing
Description	RiskTest is a tool integration platform for risk-based security testing. It consists to manage traces and gives services to evaluate / analyze these.
License	Proprietary.
Website	None.
Technical information	
Download site	None.
OS	Win, Linux and Mac
Technology environment	Java (1.6 and newer) 32-bit, Juno Eclipse Modeling
Other dependencies	Papyrus, TestingTech, ProR, CORAS, CReMa
Additional information	
Known issues/risks	None.
Additional useful information	None.

2.4 Fuzzino

General information	
Name	Fuzzino 0.2.4.2 (to be integrated with Fokus!MBT)
Provider	Fraunhofer FOKUS via GitHub
Topic addressed	Fuzz Test Data Generation
Description	Fuzzino is a library that provides generation of test data for fuzz testing.
License	Apache License 2.0
Website	https://github.com/fraunhoferfokus/Fuzzino
Technical information	
Download site	https://github.com/fraunhoferfokus/Fuzzino
OS	Win, Linux, Mac
Technology environment	EMF 2.7, Java 1.7
Other dependencies	None.
Additional information	
Known issues/risks	None.
Additional useful information	None.

2.5 Behavioral Fuzz Test Case Generator

General information	
Name	Behavioral Fuzz Test Case Generator0.1.15 (to be integrated with Fokus!MBT)
Provider	Fraunhofer FOKUS
Topic addressed	Model-Based Behavioral Fuzz Test Case Generation
Description	The tool is a test case generator that generates from UML sequence diagrams that represent valid interaction with the system under test.
License	Proprietary.
Website	None.
Technical information	
Download site	None.
OS	Win, Linux, Mac
Technology environment	Plug-in for Eclipse Indigo. It uses Eclipse EMF und Eclipse UML2 (Version 3.2.1).
Other dependencies	None.
Additional information	
Known issues/risks	None.
Additional useful information	None.

3 Descriptions of the identified WP4 tools

This Section gives a more detailed presentation of the five WP4 tools, and defines the guidelines to install and use them.

3.1 CORAS

3.1.1 Presentation

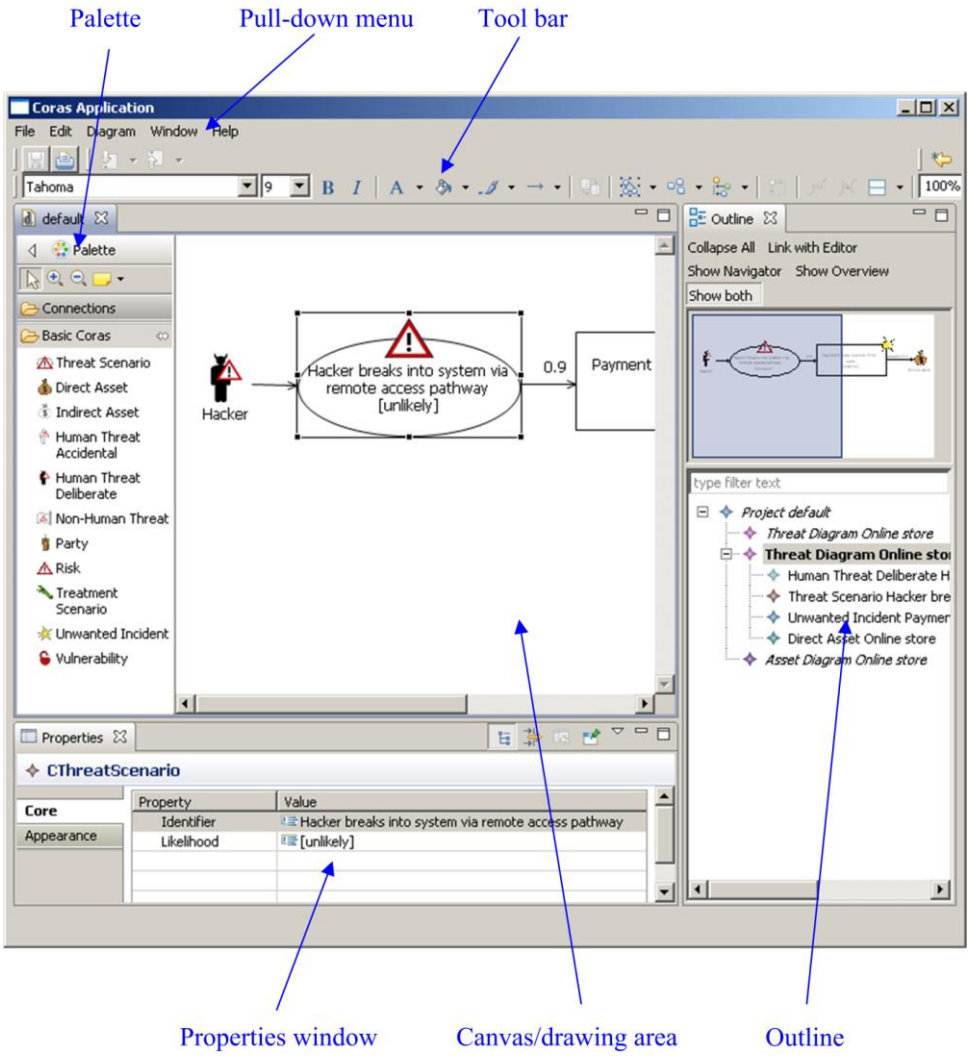

Figure 1 – Screenshot of the CORAS tool

The CORAS tool is an open source diagram editor that supports the CORAS risk analysis language. The CORAS language is a graphical language whose constructs correspond to notions that are relevant during a risk analysis, e.g. threats, vulnerabilities, risks, unwanted incidents, threat scenarios and assets. The CORAS tool is intended to be used intensively during workshops where information is gathered through structured brainstorming. The tool is also intended to be used to document a risk analysis and to present the risk analysis results.

The CORAS tool is designed to support on-the-fly modeling using all five kinds of basic CORAS diagrams, thus facilitating the entire CORAS risk analysis process. A screenshot of the CORAS diagram editor is given in Figure 1. As indicated in the figure, the editor has six main parts:

- A pull-down menu that offers standard functions such as open, save, copy, cut, paste, undo and print.
- A tool-bar that offers easy access to the standard functions of the pull-down menu.
- A palette that contains the model elements and relations for drawing the diagrams.
- A drawing area or canvas for drawing the diagrams.
- A properties window that lists the properties of a selected element, and that can be used to edit the values of the properties.
- An outline that presents a project and its diagrams as a tree.

Except for the pull-down menu and the tool bar, all parts of the tool can be closed or hidden.

In the tool, a project is a collection of diagrams, and each diagram must belong to a project. A project must therefore be created before any diagrams are created.

The outline contains a tree representation of the project. The diagrams of the project are listed at the first level, and under each diagram all the diagram elements are listed. When a new element is created in the drawing area, it is automatically added to the tree under the correct diagram.

The drawing area is the part of the tool where the diagrams are made by inserting, editing, annotating and deleting elements. This is also where likelihoods and consequences are inserted to diagrams as part of the risk estimation, and it is also where risk levels are inserted as part of the risk evaluation.

3.1.2 Installation Guidelines

- Download the zip-file containing the current version of the tool.
- Extract the zip-file to any folder of your choosing.
- Double click the file "Coras.exe" located in the eclipse folder of the distribution.

3.1.3 User Guide

Creating a new project

Before the CORAS tool can be properly used, a new project must be created. To create a new project:

- Select File -> New -> Coras Project in the File menu located near the top left corner of the window.
- Enter the desired file name and folder of the new project.
- Check “High Level CORAS”, “Dependent CORAS”, or “Legal CORAS” if you want to use these extensions of the basic CORAS language.
- Press “Finish” when done.

Creating a new diagram

The first time a new project is created, a new threat diagram called unnamed is automatically created within the project as shown in the outline view of the right hand side of the Figure 1. To create an additional diagram:

- Right-click the project entry in the tree outline located on the right hand side of the CORAS tool window.
- In the pull-down menu, select “new X diagram” (where X is the type of diagram that can be created e.g. threat, asset).
- The new diagram should appear in the tree outline. In order to open the new diagram, double click the diagram in the tree outline.

The difference between the diagrams is the kind of risk model elements that can be created within them. Note that the diagram type "Treatment" can contain (almost) all risk model elements; all other diagrams are subsets of this diagram (except the asset diagram).

Editing diagrams

Once a project and an appropriate diagram have been created, risk model elements can be added to the diagram by selection the appropriate element in the palette on the left hand side of the window, and left-clicking the diagram canvas. Relations can be created by selecting them from the pallet, or by holding the mouse over a source or target element until two small boxes appear on the border of the element. Holding down the left-mouse button on one of these small boxes will allow you to create a new relation.

3.1.4 Features within RASEN project

Planned features

The following extensions to the CORAS tool are currently planned within the RASEN project

- Add support for importing a predefined list of vulnerabilities, possibly based on CVE (Common Vulnerabilities and Exposures). The user will then be able to select these vulnerabilities when specifying the risk model. Furthermore, the vulnerabilities will be used as a means for associating appropriate test patterns to test procedures derived from the CORAS tool.
- Add support for importing and exporting CORAS models to models of the common RASEN data meta-model as a mean of communicating with the other tools in the RASEN toolbox.
- Add support for identifying, prioritizing and selection test procedures based on the risk model. This feature is broken down into the following features/tasks:
    - Implement an algorithm for calculating test procedure priority.
    - Implement support for specifying estimates needed for test procedure prioritization that is not part of the standard CORAS risk model. Specifically, estimates for *effort* and *uncertainty* need to be supported.
    - Implement support for specifying likelihood, consequence, and risk types. The calculation of the test procedure priority relies on the likelihood and consequence estimate in the risk model. However, for this calculation to be correct, the user needs to specify what kind of likelihood and consequence estimates are in the model, e.g. if the likelihood is probability, or a frequency, etc.

Current status

The first version of an export from the CORAS model to the current version of the RASEN data meta-model has been implemented. The function takes a CORAS project as input and produced an XML-file representing the RASEN data model. As an example, consider Figure 2 that shows a CORAS threat diagram. Exporting this diagram to the RASEN data model will produce the XML-file shown in Figure 3.

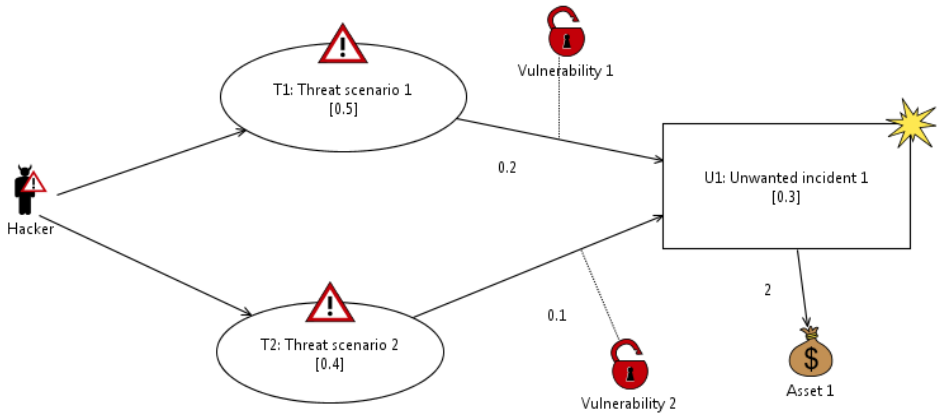

Figure 2 – Example of a CORAS diagram

```
<?xml version="1.0" encoding="ASCII"?>
<rasen.riskassessment:RiskModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:rasen.riskassessment="http:///rasen/riskassessment.ecore"
xmlns:rasen.securityriskassessment="http:///rasen/securityriskassessment.ecore">
<relations type="INITIATES" source="//@elements.4" target="//@elements.3"/>
<relations type="IMPACTS" source="//@elements.1" target="//@elements.0">
<properties xsi:type="rasen.riskassessment:Consequence" value="2.0" type="//@types.1"/>
</relations>
<relations type="INITIATES" source="//@elements.4" target="//@elements.2">
<properties xsi:type="rasen.riskassessment:Likelihood" value="0.5" type="//@types.0"/>
</relations>
<relations type="LEADS_TO" source="//@elements.2" target="//@elements.1">
<properties xsi:type="rasen.securityriskassessment:Vulnerability" identifier="V__vulnerability
1" name="vulnerability 1"/>
<properties xsi:type="rasen.riskassessment:Likelihood" value="0.2" type="//@types.0"/>
</relations>
<relations type="LEADS_TO" source="//@elements.3" target="//@elements.1">
<properties xsi:type="rasen.riskassessment:Likelihood" value="0.1" type="//@types.0"/>
<properties xsi:type="rasen.securityriskassessment:Vulnerability" identifier="V__vulnerability
2" name="vulnerability 2"/>
</relations>
<types name="PROBABILITY_MUTUAL_EXLUSIVE"/>
<types name="REAL"/>
<riskFunctions name="PRODUCT"/>
<elements xsi:type="rasen.securityriskassessment:Asset" identifier="A__asset 1" name="asset
1"/>
<elements xsi:type="rasen.securityriskassessment:UnwantedIncident" identifier="UI__u1"
name="unwanted incident 1">
<likelihood value="0.3" type="//@types.0"/>
</elements>
<elements xsi:type="rasen.securityriskassessment:ThreatScenario" identifier="TS__t1"
name="threat scenario 1">
<likelihood value="0.5" type="//@types.0"/>
</elements>
<elements xsi:type="rasen.securityriskassessment:ThreatScenario" identifier="TS__t2"
name="threat scenario 2">
<likelihood value="0.4" type="//@types.0"/>
</elements>
<elements xsi:type="rasen.securityriskassessment:Threat" identifier="T__hacker"
name="hacker"/>
</rasen.riskassessment:RiskModel>
```

Figure 3 – Example of a XML-file produced from a CORAS diagram

The algorithm for test procedure prioritization and selection (described in deliverable D4.2.1) has been implemented in the CORAS tool. A preliminary version of the graphical interface for invoking the test prioritization and selection algorithms has also been implemented. The user interface is outlined by the red rectangle in Figure 4. The two buttons "generate prioritization" and "generate selection" invoke the prioritization and selection algorithms, respectively, and displays the output in the text area at the bottom. The "Export" button is used to export the CORAS project into the generic RASEN data model.

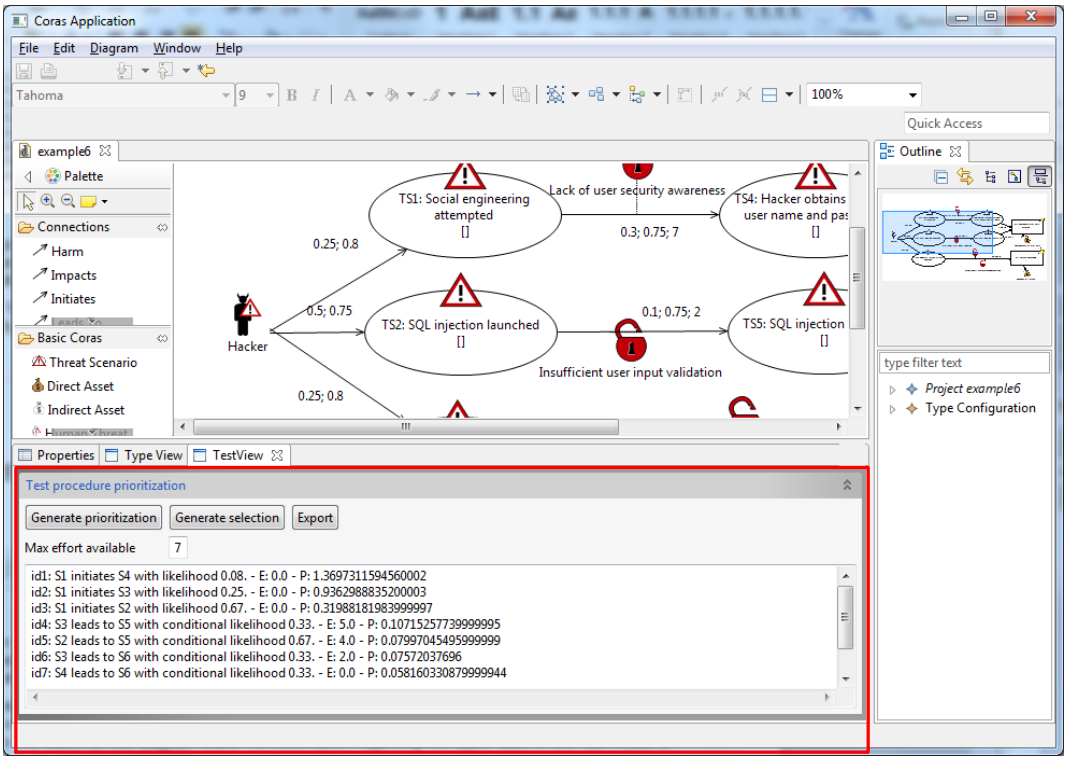

Figure 4 - GUI for test procedure prioritization and selection

One of the algorithms for the test procedure prioritization has been implemented in the CORAS tool. Currently, it can produce a comma separated list of prioritized test procedures. Some estimate values that the prioritization relies on (testability and uncertainty) need to be hard coded before the implementation can run. This also needs to be done for the type structure, i.e. the kind of likelihood values and consequence values that are in the CORAS model.

Work has started on making a graphical user interface that the user can use in order to specify the type structure of CORAS models. As outlined in Figure 5 by the red rectangle, a new tab has been added to the button of the diagram editor. All information regarding the types used in the risk model is intended to be specified in this tab.

In summary, what remains is to

- Develop import of vulnerabilities.
- Develop import from RASEN data model to the CORAS tool.
- Finish development of GUI-support for type specification, and test prioritization/selection.

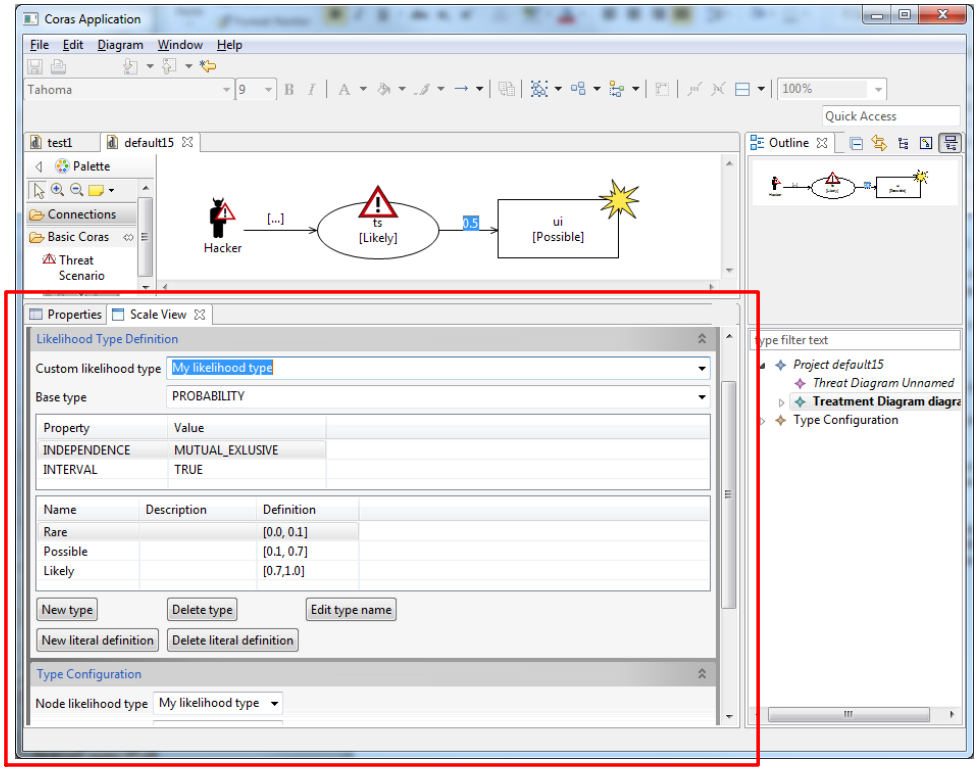

Figure 5 – GUI for specifying types

3.2 Smartesting CertifyIt

This section introduces CertifyIt developed and provided by Smartesting to generate security test cases from vulnerability risk assessment. It should be noted that a more detailed presentation of the tool including tutorial, and tips to use it efficiently, are provided with the tool (modeling guide, UML/OCL reference guide and software documentation).

3.2.1 Presentation

Smartesting CertifyIt is a tool suite that automatically generates test cases based on a model of system requirements. Manual test design is labor intensive and error prone; this manual work can be avoided for complex applications by modeling the key concepts (abstraction) and allowing Smartesting CertifyIt to automate your test design work. Since the model is more expressive and simpler than the system-under-test, it can more readily be reviewed for correctness and coherency, as well as be updated more easily. Smartesting CertifyIt supports UML/OCL models as the specification modeling language, and generates test cases to cover security test patterns used as test objectives. Finally, the generated test cases can be exported in UML sequence diagrams and can be fully integrated into the process promoted by the RASEN project.

Smartesting CertifyIt enables to:

1. Implement your testing strategy by importing security test patterns.
2. Generate your tests from several criteria, based on risk assessment, and manage traceability.
3. Publish abstract test cases for documentation or manual exploitation.
4. Publish test scripts into a testing environment for automated execution.
5. Easily manage the evolution of the specification: update your model and Smartesting CertifyIt will generate the new test cases.

3.2.2 Installation Guidelines

Smartesting test generation solution works with models edited by an Eclipse-based UML modeler. Smartesting provides a plug-in that checks whether the model fits the Smartesting CertifyIt restrictions on UML, and exports a model to be used by the Smartesting CertifyIt tool to generate the test cases. To use Smartesting test generation solution, both the Eclipse-based modeler plug-in for Papyrus and the Smartesting CertifyIt software are needed.

IBM RSA Modeler Plug-in installation

1. This plug-in must be installed with the Eclipse update site tool. The following steps describe the installation of a new release.
2. Start Papyrus Eclipse-based UML modeling workbench.
3. From the workbench menu, select Help->Software Updates->Find and Install...
4. A dialog opens. Select "Search for new features to install", and click "Next". Select "New Archived Site ...".
5. Specify the path to a .zip file available in the 'install' folder of the Smartesting CertifyIt installation directory. Once the path is selected, the following window should appear. You can use the 'Name' field in order to define your own Local Site name.
6. Click the "Finish" button of the "Install" window to launch the installation.
7. Check the Smartesting CertifyIt plug-in, and click « Next ».

   At this stage, you may have warnings indicating that some features cannot be installed (due to unavailable dependencies). In that case, check the “System Requirements and Supported Software”, and upgrade RSAmodeler if required.
8. Accept terms of the license agreement, and click « Next ».
9. Click the « Finish » button to install the plug-in.

If verification messages appear, just accept those in order to complete the installation. It should be noted that write permissions for the install location are needed. Afterwards, restart RSA modeling workbench. The Smartesting CertifyIt plug-in should be successfully installed at this point.

Smartesting CertifyIt software installation

To install Smartesting CertifyIt software, simply run the setup program and follow the installation instructions (this implies accepting the terms of the displayed license agreement).

Before using Smartesting CertifyIt, a license needs to be defined. Prerequisites are: Smartesting CertifyIt must be installed and the RSA Eclipse-based modeler plug-in must be installed. There are two types of licenses: floating license server or capacity-based license.

3.2.3 User Guide

A model is an abstraction of the reality to achieve an objective. The objective of Smartesting solution is to support the validation engineer in its testing effort. Hence, the activity of creating UML models must always be evaluated against a testing goal: “Will this provide the relevant tests required for my system?”. In this perspective, the validation engineer will make a number of choices while modeling its system:

- What are the boundaries of the system?
- What are the inputs and outputs relevant for testing the system?
- What is the amount of information needed to model the behavior of the system?

The objective of the validation engineer is not to model all the software system in its most accurate detail, but, most of the time, to model just enough and rightly focused system behavior to leverage Smartesting CertifyIt capabilities to generate relevant test cases.

This is why the use of Smartesting CertifyIt, and the use of models, must always be considered as part of a larger perspective on how the target system must be tested. As a result, a test-oriented model should not be considered as design model. It only describes the system as a black box and only models parts of the system relevant for Smartesting CertifyIt. Consequently some UML design concepts such as interfaces, ports, and others are not useful.

What are the required steps to specify a test-oriented model?

1. Select a functionality that needs to be tested (a requirement).
2. Define the available control points and observation points for this functionality.
3. Model the control and observation points on a class diagram.
   a. Defining the required level of abstraction.
   b. Using classes, operations and data types to model those points.
4. Model the expected system behavior of this functionality with OCL and state machines.
5. Model an initial state for the system as object instances and links.

3.2.4 Features within RASEN project

During the RASEN project, several Smartesting CertifyIt extensions have to be developed.

1. Test Purpose language extension, enabling sets creation using OCL code evaluated on the initial state of the system
2. Keyword creation to be used by Test Purposes. This mechanism enables to create generic Test Purposes, and to help for maintenance and reuse.
3. Capability to link a Test Purpose to a requirement identifier to ensure the traceability through the all test generation process.
4. Test Purpose catalogue import/export to reuse and apply Test Purposes on several systems under test.
5. Generation and Animation standalone Java API to enable fuzzed test sequences validation regarding the model.

Those features are presented in a more detailed way in the D4.2.1 RASEN deliverable.

3.3 RISKTest

3.3.1 Presentation

The RISKTest trace management platform is integrated in the IDE of the Eclipse workbench (see Figure 6) and runs with the modeling tool Eclipse in the version JUNO and INDIGO. Trace management capabilities are realized, i.e. the creation of trace links, the navigation of trace links and the evaluation trace links. The tools supported by RISKTest are: the risk modeling tool CORAS, the Eclipse UML modeling editor Papyrus, the requirement modeling tool ProR, based on the ReqIf model, and the test managing tool TTworkbench. Services embedded in RISKTest are the fuzz test generator and the vulnerability coverage analyzer.

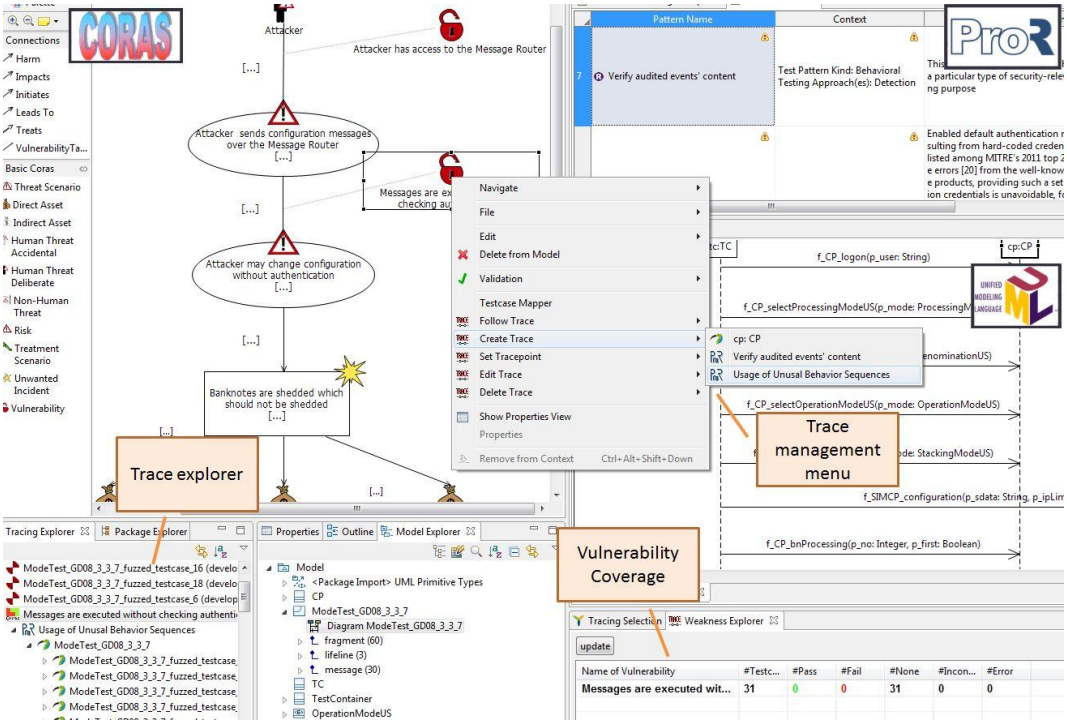

Figure 6 – RiskTest embedded in Eclipse

3.3.2 Installation Guidelines

1. Ensure that a Java 32-bit is installed.
2. Download the latest Juno Eclipse Modeling instance.

   (http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/junosr2)
3. The tools supported by RiskTest have to be installed in the eclipse before RiskTest will be installed itself. For this, install the tool Papyrus: Enter menu “Help” >> “Install Modeling Components” >> search for the Papyrus Tool (in incubation state) and install it.
4. Install the TTWorkbench tool (http://www.testingtech.com/support/downloads.php). Please contact TestingTech to get a license needed for usage.
5. Use the update site for RISKTest to install the other tools CORAS and ProR. This is needed because the official download sites do not provide the version supports by RISKTest anymore. To use the update site: “Help” >> “Install New Software…” >> “Add…” > enter the update site link in the “Archive” text field.
6. By usage of the update site the trace framework CReMa and all RISKTest components can also be installed.

3.3.3 User Guide

The RISKTest integrates the trace administration directly in the model development editor views. The user can use the trace management easily by selecting element(s) in the editor(s) and open the popup context menu. There, the user can select between creating new traces between selected elements, navigating to a traced element from a selected element, deleting a trace between the selected element and a traced element, and editing one of the traces of a selected element.

The fuzz test generator can be opened by selecting a UML-file with the .uml extension. Then select “DIAMONDS” >> “Fuzz Interactions” in the context menu.

The Vulnerability Coverage explorer is integrated in the eclipse framework as a view and can be opened by selecting the menu “Window” >> “Show View” >> “Other” >> “Tracing” >> “Weakness Explorer”.

3.3.4 Features within RASEN project

Current Status

- The trace management (the creation, deletion of traces) is directly integrated in each of the integrated tools. Additionally, traces are visualized in the trace explorer by a tree structure and can be used for navigation. A filter mechanism can be used to find traces.
- RISKTest allows bidirectional navigation between related elements. Trace source and target can be visualized directly within the integrated tools.
- Creating traces can be done manually or automatically, e.g. by model-based test generation services.
- The traces are defined on basis of a trace meta-model that distinguishes the individual elements that are allowed to be part of a trace and allow for distinguishing different trace types.
- The trace meta-model defines a service interface that allow for introducing services that query the trace model for information (e.g. services for coverage analysis or impact analysis).
- RISKTest is extensible. That means, it provides a well-defined interface to easily integrate other tools that are based on Java and Eclipse.

Planned Features

- Support for the exchange format for RASEN Data Integration Model specified in RASEN deliverable 5.4.1.
- Risk-based security test condition identification and prioritization: Prioritize test items and test conditions on basis of risk analysis and improved test pattern.
- Dashboard of security testing results based on risk assessment.
- Risk-based security test condition identification and prioritization: Prioritize test items and test conditions on basis of risk analysis and improved test pattern.

3.4 Fuzzino

3.4.1 Presentation

Today, data fuzzing is a widely accepted and performed technique for security testing. There are many different fuzzers available, some are commercial, and many are open-source. The main difference between the commercial and the open source fuzzers is that the latters are often developed are generic or for a specific protocol, while commercial fuzzers address a large number of different protocols. Hence, for testing different systems, several fuzzing tools are needed when considering open source tools. Additionally, Charles Miller came to the conclusion that the more fuzzers are used the better (see [1]) in order to find the most weaknesses. Following that, for performing fuzz testing a lot of fuzzing tools should be used. Furthermore, each tool has to be configured for the specific SUT requiring the tester to become acquainted with each tool. The fuzz test data generator Fuzzino solves this problem. It combines the core of several open source fuzzing tools – their test data generators – and makes them available for other tools by a unified interface.

3.4.2 Installation Guidelines

For compiling the Fuzzino sources, you need Eclipse EMF 2.7, more precisely the following JAR files:

- org.eclipse.emf.common, e.g. org.eclipse.emf.common_2.7.0.v20120127-1122.jar
- org.eclipse.emf.compare, e.g. org.eclipse.emf.compare_1.2.2.v20120214-0915.jar
- org.eclipse.emf.ecore, e.g. org.eclipse.emf.ecore_2.7.0.v20120127-1122.jar
- org.eclipse.emf.ecore.xmi, e.g. org.eclipse.emf.ecore.xmi_2.7.0.v20120127-1122.jar.

3.4.3 User Guide

In order to retrieve fuzzed values from Fuzzino, a request in the form of an XML document (herein after referred to as “request document”) has to be created. A request document may contain several type specific requests, namely string requests, number requests, structure requests, and collection requests. Each of these type specific requests has attributes to specify the data format of valid values.

Additionally to this data form specification, generators and operators (in combination with concrete valid values) can be specified that shall be used by Fuzzino for creating fuzzed values. If no generators or operators are specified, Fuzzino will use all generators and, if valid values are contained in a type specific request, all operators that match the data format description. To avoid this behavior, in the case of generators the tag `useNoGenerators` can be included in the request while no operators are used if no valid values are contained in a request.

Because most of the fuzzing generators and operators create a huge number of valid values, the number of fuzzed values has to be specified for each type specific request.

The usage of Fuzzino is illustrated in Figure 7 for the case of a request of the type string. The first message contains the above mentioned information.

Every request is answered by Fuzzino with a response containing the fuzzed values, grouped by the generators and by operators in combination with the corresponding valid value that created them. This is depicted by message 2 in Figure 7.

Because of the limited number of fuzzed values returned by Fuzzino, it is often necessary to retrieve further values from Fuzzino. For that purpose, each response has two additional attributes, `moreValues` that indicates if further values can be retrieved, and `id` that has to be used to retrieve further values from Fuzzino that differs from the already retrieved values (see message 2 in Figure 7). To do so, type specific request has to be complemented with this id obtained from the response of Fuzzino. The data format description as well as the generators, operators and valid values can be omitted because they are already known from the initial request (see messages 3 and 4 in Figure 7).

If the desired number of fuzzed values was retrieved from Fuzzino, a request should be closed by sending the corresponding tag `closeRequest` with the id from Fuzzino. When receiving such a tag, Fuzzino removes temporary files regarding the request to be closed. After closing a request, it is impossible to retrieve further values for this request (see messages 5 and 6 in Figure 7).

3.4.4 Features within RASEN project

- Integration with Fokus!MBT.
- Implementation of new fuzzing heuristics developed along the case studies.
- Capability to read UML models to extract data format specifications.

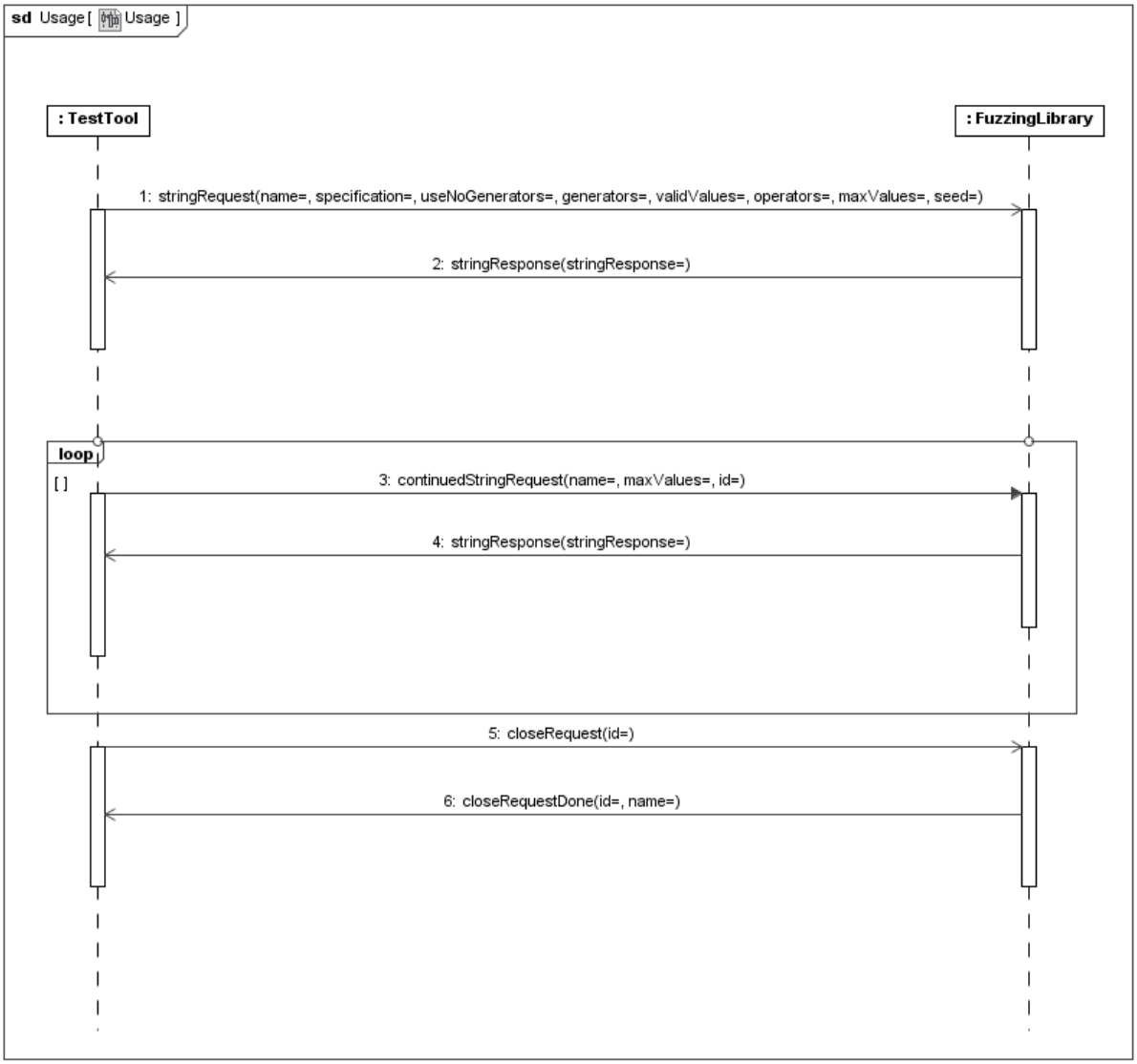

Figure 7 – Usage of Fuzzino by sending requests and receiving responses

3.5 Behavioral Fuzz Test Case Generator

3.5.1 Presentation

The behavioral fuzz test case generator implements behavioral fuzzing of UML sequence diagrams. It reads a UML model containing one or more interactions and generates behavioral fuzz test cases in form of UML Interactions. Additionally, it can generate TTCN-3 code on basis of a skeleton test case.

3.5.2 Installation Guidelines

The test case generator is provided as a plug-in for Eclipse Indigo. It requires the following plug-ins:

- EMF
- Eclipse UML2

3.5.3 User Guide

The test case generator is provided as an Eclipse plugin. In order to start test case generation, It can be directly accessed by right-clicking on a .uml file as shown in Figure 8.

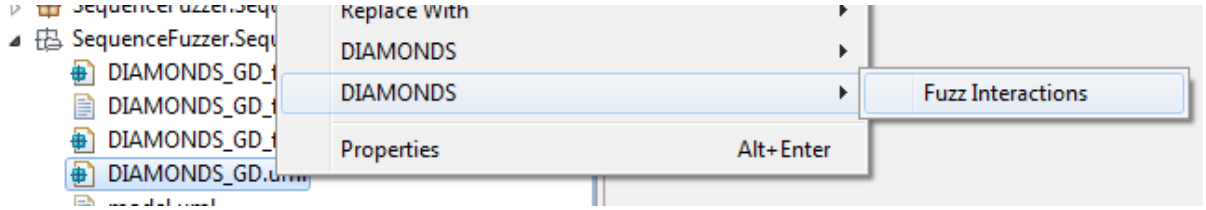

Figure 8 – Context menu of test case generator plug-in on a .uml file

After that, a configuration page is displayed (see Figure 9). Using this configuration page, the test case generator is configured. This configuration page allows to specify

- where the model that will contain the behavioral fuzz test cases shall be saved,
- whether to generate TTNC-3 code from the behavioral fuzz test cases and where to save it,
- which UML interaction within the selected model shall be fuzzed,
- which are lifelines that represent a test component and which represent the system under test,
- the behavioral fuzzing operators to be used,
- the maximum number of test cases to be generated, and
- how many fuzzing operators shall be used to generate a single test case.

3.5.4 Features within RASEN project

- Integration with Fokus!MBT.
- Implementation of additional fuzzing heuristics developed along the case studies.

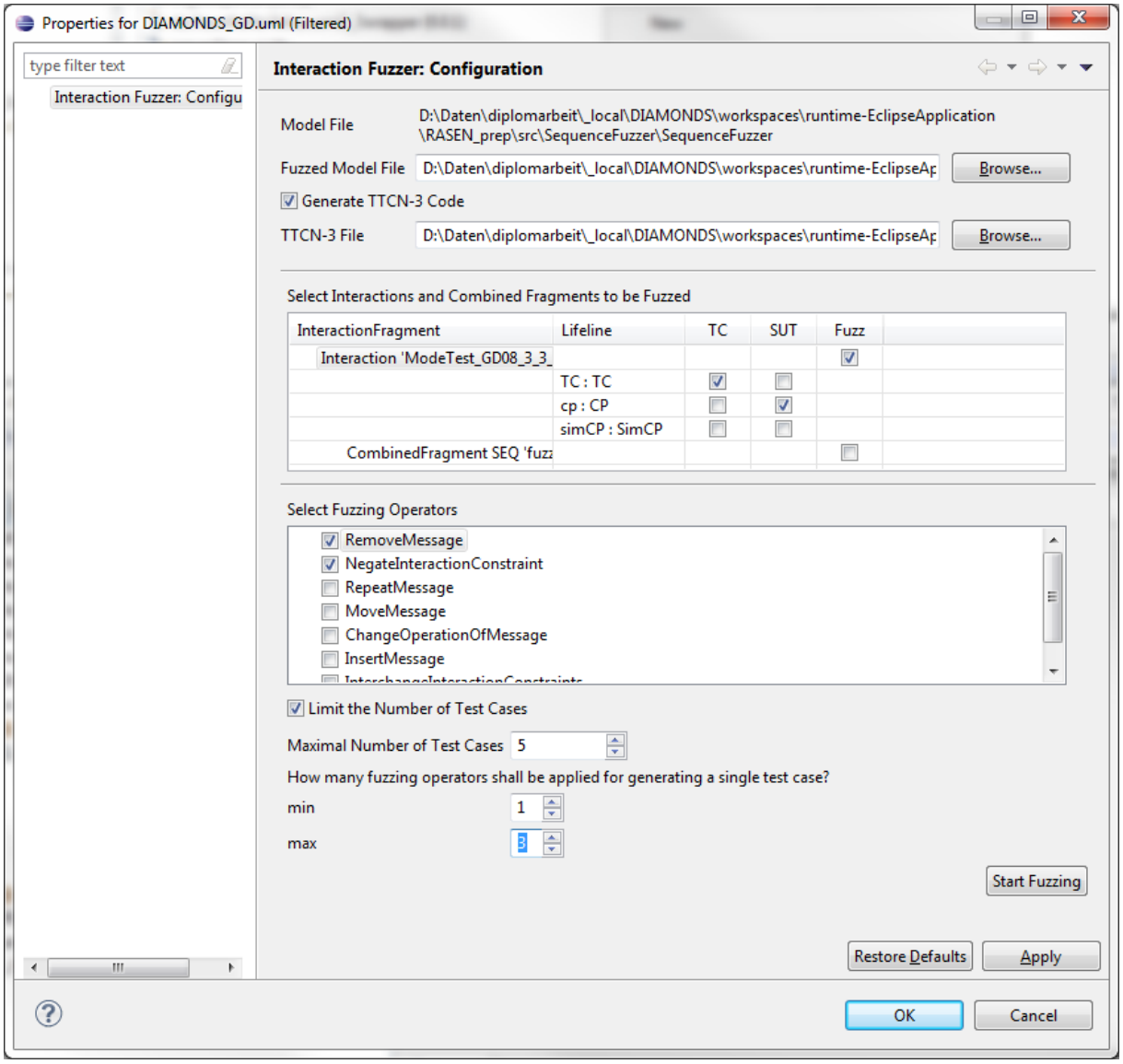

Figure 9 – Configuration page of behavioral fuzz test case generator

4 Conclusion

In this report, we have given an overview of the RASEN WP4 tools that are delivered as prototype deliverable within this deliverable D4.3.1. In context of WP4, these tools are developed to support compositional security test generation driven by risk coverage.

The tools presented in this deliverable are CORAS, Smartesting CertifyIt, RISKTest, Fuzzino library and Behavioral Fuzz Test Case Generator. For each tool, features, current development status and further development goals have been described. They will all be integrated to the RASEN tool suite, which is defined within WP5 in terms of methodologies and tool support.

References

[1] Takanen A., J. DeMott, and C. Miller: Fuzzing for software security testing and quality assurance, 1st ed. Norwood, MA, USA: Artech House, Inc.(2008)