

	<p align="center">Initial Security Testing Tools</p> <p align="center">Deliverable ID: D3.WP3</p>	Page : 1 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

	Initial Security Testing Tools	
	Version: 1.1 Date : 29.6.2012 Pages : 80	
	Editor: Matti Mantere	
	Reviewers: IT, TL,itrust	
	To: DIAMONDS Consortium	
<p>The DIAMONDS Consortium consists of: Codenomicon, Conformiq, Dornier Consulting, Ericsson, Fraunhofer FOKUS, FSCOM, Gemalto, Get IT, Giesecke & Devrient, Grenoble INP,itrust, Metso, Montimage, Norse Solutions, SINTEF, Smartesting, Secure Business Applications, Testing Technologies, Thales, TU Graz, University Oulu, VTT</p>		
Status: <input type="checkbox"/> Draft <input type="checkbox"/> To be reviewed <input type="checkbox"/> Proposal <input checked="" type="checkbox"/> Final / Released	Confidentiality: <input checked="" type="checkbox"/> Public Intended for public use <input type="checkbox"/> Restricted Intended for DIAMONDS consortium only <input type="checkbox"/> Confidential Intended for individual partner only	

	<p align="center">Initial Security Testing Tools</p> <p align="center">Deliverable ID: D3.WP3</p>	Page : 2 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

Deliverable ID: D3.WP3

Title:

Initial Security Testing Tools

Summary / Contents:

This document describes the initial security testing tools in DIAMONDS project.

Contributors to the document:Wissam Mallouli (Montimage), Bachar Wehbi (Montimage), Edgardo Montes de Oca (Montimage), Michel Bourdellès (Thales), Stephan Pietsch (Testing Technologies), Stephan Schulz (Conformiq), Julien Botella and Bruno Legeard (Smartesting), Nikolay Tcholtchev (Franhofer FOKUS), Martin Schneider (Fraunhofer FOKUS), Ilkka Uusitalo (VTT), Matti Mantere (VTT), Markku Tyynelä (Metso), Christian Wieser (OUSPG), Felix Jakob (Dornier Consulting), Andreas Schulze (Dornier Consulting), Andrej Pietschker (Giesecke & Devrient GmbH), Fredrik Seehusen (SINTEF)




	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	<p>Page : 3 of 81</p> <hr/> <p>Version: 1.1 Date : 29.6.2012</p> <hr/> <p>Status : Final Confid : Public</p>
---	---	--

TABLE OF CONTENTS

1. Introduction.....	8
2. Case Study test execution examples	8
2.1 Test execution in the automotive case study	8
2.2 Data Fuzzing with TTCN-3 in the Finance and Banking case study	10
2.2.1 Introduction	10
2.2.2 TTCN-3 Core Language Extensions	11
2.2.3 TCI Extensions	12
2.3 DO.Atoms framework	13
2.3.1 Extracting Risk Information	13
2.4 Application Programming Interface for active security testing	17
3. Implementation of a Trace Management Platform (FOKUS)	18
3.1 Architecture of the CReMa Tool	18
3.2 Requirements for User Interface Extensions	19
3.3 Requirements for Additional Extensions	20
3.4 Architecture of the Trace Management System	21
4. Data Fuzzing Library (FOKUS)	23
4.1 Existing Open Source Fuzzing Tools	23
4.2 Use Cases	24
4.3 Requirements	25
4.4 Implementation Approach	25
4.4.1 Interfaces Provided by the Library	26
4.4.2 Architecture	27
4.4.3 Extending the Library with New Fuzzing Heuristics	28
5. Fuzz Test Generator for UML Sequence Diagrams (FOKUS)	30
5.1 Use Cases	30
5.2 Requirements	30
5.3 Implementation Approach: Architecture	30
5.3.1 Interfaces	31
5.3.2 Fuzzing Heuristics	31
5.3.3 Fuzzing Strategies	33
6. Security monitoring libraries (Montimage)	35
6.1 MMT-Extract library	35
6.1.1 Installing MMT-Extract	36
6.1.2 Using MMT-Extract in a development project	36
6.1.3 Extraction API description	36
6.2 MMT-Security library	40
6.2.1 Installing MMT-Security	40
6.2.2 Using MMT-Security in a development project	40
6.2.3 Security analysis API description	40
7. Model-based behavioral security testing tool development (Smartesting)	42
7.1 Overall Process	42
7.2 Sample Application	43
7.2.1 Functional and Security requirements	43
7.2.2 Identified risks	43
7.3 Test Generation Model	43
7.3.1 Creating a new UML Project	43
7.3.2 Creating the UML Model in the project	45
7.3.3 Creating a Class Diagram in the model	46
7.3.4 Creating an Instance Diagram in the model	53

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 4 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

7.3.5	Creating a Test Suite	56
7.3.6	Using Smartesting Simulator.....	57
7.3.7	Generating tests with Smartesting Certifylt.....	58
7.3.8	A step further: observations and traceability	60
7.4	Test Purpose language for Security Oriented Test Generation	64
7.4.1	Test Purpose Language Objectives	64
7.4.2	Test Purpose Language	64
7.4.3	Test Purpose Language Editor	66
7.4.4	Test Generation Strategy from Test Purpose	66
8.	Framework for ACTIVE security testing (FSCOM)	68
8.1	Introduction to the Security testing framework.....	68
8.2	Security testing	68
8.2.1	Candidate EUTs/IUTs	69
8.2.2	Test scenarios	69
8.2.3	Test bed architecture	69
8.3	Identification of abstract test method	70
8.4	Protocol description using ASN.1	71
8.5	Integration of the security framework with OMNeT++	71
8.6	Tools to convert Montimage security rules into ETSI TPLan language	72
	References	73
	Appendix	75
	Appendix A: Extraction code examples for MMT extraction library	75
	Appendix B: Security analysis example.....	77


	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 5 of 81
		Version: 1.1
		Date : 29.6.2012
		Status : Final Confid : Public

FIGURES

Figure 1: Excerpt of a fuzzing workflow in do.ATOMS	9
Figure 2: CORAS model for the threat scenario 'Uploading modified firmware'	14
Figure 3: CORAS legend of used elements.....	14
Figure 4: CORAS scenario within the system model.....	15
Figure 5: CORAS scenario modeled in the system model as a sequence diagram	16
Figure 6: Sequence diagram sorted under the CORAS use case in the system model hierarchy	16
Figure 7: The priority of a message is a combination of the introduced securityscore and the occurrence of a message within the system model.....	17
Figure 8: User Interface Extensions	19
Figure 9: Use case diagram for trace navigation and administration extensions	20
Figure 10 Use case diagram for TMP trace editor view extensions	21
Figure 11: Overview of the Trace Management System	21
Figure 12: Use Case Diagram for TMP services and TMP Core Features.....	22
Figure 13: Excerpt from an XML Request File.....	26
Figure 14: Excerpt from an XML Response File.....	27
Figure 15: Internal Architecture of the Data Fuzzing Library	28
Figure 16: The Class ComputableList and its Relationships	29
Figure 17: Illustration of the Architecture.....	31
Figure 18: Eclipse Command Framework Class AbstractOverridableCommand and its Subclass FuzzingOperator.....	33
Figure 19: The Classes Implementing Fuzzing Strategies	34
Figure 21: MMT-Extraction Library	36
Figure 22. MMT-Security Library	40
Figure 23: Smartesting process and tool for model-based security testing	42

TABLES

Table 1: Template example	12
Table 2: Chosen Fuzzing Heuristics from Selected Fuzzers	24


	<p align="center">Initial Security Testing Tools</p> <p align="center">Deliverable ID: D3.WP3</p>	Page : 6 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

HISTORY

Vers.	Date	Author	Description
0.1	2011/09/26	W. Mallouli	Document creation
1.0	2012/05/29	M. Mantere	Edited document based on reviews and revisions by reviewers and partners.
1.1	2012/06/29	M. Mantere	Slight changes, added list of contributors, fixed status and confidentiality classifications. Also added executive summary.

APPLICABLE DOCUMENT LIST


Ref.	Title, author, source, date, status	DIAMONDS ID
1		

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 7 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

EXECUTIVE SUMMARY

The DIAMONDS project focuses on model-based security testing of networked systems. Testing is the main method to reliably check that a software-based system meets its requirements with regard to functionality, security and performance. In this document we discuss the initial security testing tools in DIAMONDS.

Model- based Testing is the approach of deriving systematic tests for a system based on an abstract representation thereof called models. These models may describe the behaviour of the system, security constraints (for example access control), the security requirements, or information about possible security threats, faults or attacks. The state of the art in model-based testing tools was given in D1.WP3 [6]. In this document D3.WP3 we describe the initial model-based security testing tools by the different project partners. This document can be considered a progress report of each partner developing its testing tool.

	<p align="center">Initial Security Testing Tools</p> <p align="center">Deliverable ID: D3.WP3</p>	Page : 8 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

1. INTRODUCTION

The document comprises of the automotive and financial case studies related information concerning security testing tools and further information on security testing tools design. The initial part of the document, namely the Section 2 handles the case study examples, while the latter part of the document discusses other topics concerning design of security testing tools.

As originally planned in the FPP, the task 3.4 was supposed to be in charge of the definition and development of add-ons and/or new tools for executing security related tests. This was also related to the fact that the original plan for WP 3 was the development of a generic and universal test definition and execution framework that could have been applied to all case studies. This is not the case anymore, so during the plenary meeting in Vienna (Feb 2012) there was the common agreement that the result of this task will be changed as follows:

- there will be no development of new generic, generally accepted tools solely related to test execution
- existing test execution tools will be extended to be able to execute security related tests in the context of each case study
- each case study uses its own tools, there is no need for a generic universal test execution tool


2. CASE STUDY TEST EXECUTION EXAMPLES

In the following sections two examples of case study specific approaches to test execution are explained in more detail. The first example is related to the automotive case study provided by Dornier Consulting. In this case study the Bluetooth based connection between a car and a consumer mobile phone is examined using a risk based testing approach that also includes data as well as behaviour fuzzing. The second example is taken from the finance and banking case study provided by Giesecke & Devrient. Here a banknote processing machine gets analysed which involves risk based testing and model-based fuzzing methods.

2.1 TEST EXECUTION IN THE AUTOMOTIVE CASE STUDY

For test execution purposes Dornier Consulting uses its in-house testing framework do.ATOMS, which is already used for the model-based generation of test cases. In general, the execution of a fuzzing test case here is somewhat different compared to the execution of a normal test case. What differs most is the process of determining the test result. When a common functional test case is finished the SUT can be examined in some way (e.g. by inspecting the state of the GUI or analyzing the log files) in order to find out whether the test was successful or not. When it comes to the execution of fuzzing test cases the situation changes since the determination of the test result is often not that evident. For each fuzzing test case one has to find measurable criteria for deriving the test result from (e.g. the SUT could ignore a fuzzing attack completely and just continues to function normally or it faces a fatal crash).

In terms of do.ATOMS based test execution the workflow itself defines the control flow, so each activity is executed in the correct order and triggers a certain test step functionality (like validating or sending a CAN or Bluetooth message to the SUT, creating a log or a test report or triggering an image based validation of the touchscreen status of the head unit). That is also the case for fuzzing

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 9 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

test cases - the main difference is the fuzzing container which manipulates its child activity to inject data fuzzing. The particular manner in which this data fuzzing is applied depends on the concrete action performed by the child activity.

The following figure shows an excerpt of a fuzzing workflow that handles the Bluetooth pairing process (e.g. between the connectivity module of the car and the driver's mobile phone). Two different types of activities are shown. *RequestPair_I1*, *EnterPIN_I4*, *DiscoverServices_I5*, and *Unpair_I6* are triggering Bluetooth activities. Whereas the activities *VRM_SendSignals_I2* and *Con_Send_Signals_2_I3* are triggering Controller Area Network(CAN) messages. The data fuzzing container has been applied to *EnterPIN_I4* activity. The other activities that are sending Bluetooth requests or validating CAN signals remain untouched.

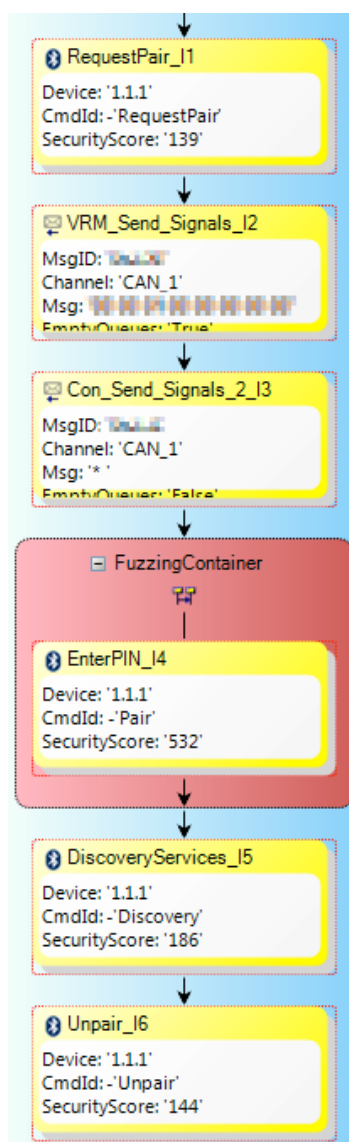



Figure 1: Excerpt of a fuzzing workflow in do.ATOMS

When speaking of data fuzzing, the creation of semi-valid input is of particular importance. In general, it depends on the specific protocol that is fuzzed how this input is generated. Taking the

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 10 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

example of sending a PIN to the connectivity module the fuzzing container investigates the underlying function that technically sends the PIN and inspects the corresponding parameters (in this case: the PIN of datatype “string”, consisting of at least a 4-digit number with values between 0 and 9). Depending on the datatype various fuzzing methods are applied by the fuzzing container to create a semi-valid version of the expected input data (like sending characters instead of digits or a PIN with an abnormal length, e.g. consisting of 1000 random digits). Additionally it can also be helpful to use invalid characters that have special meaning, for instance null characters that terminate C strings, or the percentage character that is used for formatted strings. While a dumb fuzzer would generate invalid input data for most of the messages, a model-based fuzzer has enough knowledge to fuzz only the parameters of a certain message, e.g. as in this example the message that carries the PIN. The fuzzing container itself can also be configured in various ways to control its runtime behavior. One example is that the container can be instructed to create a new fuzzed input value for each test run.

This fuzzing approach is realized by implementing the fuzzing library provided by Fraunhofer FOKUS. This library encapsulates all fuzzing related business logic and allows do.ATOMS the retrieval of fuzzed input data values by simply sending an XML request. This request contains a definition of the data type or structure that should be fuzzed and the fuzzing generators that should be applied to it. The fuzzing library will also be capable of supporting behavioural fuzzing which will be applied to the Dornier case study later on, too. Behaviour Fuzzing will affect the order in which the workflow activities get executed.

2.2 DATA FUZZING WITH TTCN-3 IN THE FINANCE AND BANKING CASE STUDY


2.2.1 Introduction

The Testing and Test Control Notation version 3 (TTCN-3) is a standardized test specification language created 10 years ago by the European Telecommunications Standards Institute (ETSI) that is becoming more and more popular. Having its root in the testing of telecommunications protocols, TTCN-3 spreads now throughout a large number of domains such as Mobile Telecommunications: 3G, 3GPP LTE, WiMAX, GSM, Internet protocols: SIP, IMS, IPv6, Intelligent transport systems (ITS), IOT – Internet Of Things (building automation, smart metering, etc), Automotive (AUTOSAR conformance and acceptance testing) and Smart Cards.

These domains have integrated TTCN-3 into their own methods and processes. Security Testing, a rather new domain for TTCN-3, provides a test method called Fuzzing. Fuzzing or Fuzz Testing is a testing technique that monitors a system for exceptional behavior (such as crashes, memory leaks) while stimulating it with random, invalid or unexpected input data.

In order to be able to apply this method with using TTCN-3, there was a need to extend the standardized language to support fuzz testing. Data Fuzzing with TTCN-3 is a combined effort of FOKUS Fraunhofer and Testing Technologies in the context of the DIAMONDS project. It is planned to present the work on at the upcoming TTCN-3 Users Conference 2012 in Bangalore as well as being part of the next DIAMONDS milestone M3.

FOKUS is working on a Data Fuzzer based on available open source fuzzing tools. Testing Technologies plans to integrate the FOKUS Data Fuzzer in their TTCN-3 test automation platform TTworkbench. For this reason the following extension of the TTCN-3 standard was discussed.

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 11 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

2.2.2 TTCN-3 Core Language Extensions

2.2.2.1 Package Conformance and Compatibility

The package presented in the present document is identified by the package tag:

"TTCN-3:2012 Fuzz Testing" to be used with modules complying with the present document.

2.2.2.2 The Fuzz Function

Generally, matching mechanisms are used to replace values of single template fields or to replace even the entire contents of a template. Matching mechanisms may also be used in-line. A new special construct called a `fuzz function` instance can be used like a normal matching mechanism "instead of values" to define the application of a fuzz operator working on a value or a list of values or templates. The definition of such a function is similar to the existing TTCN-3 concept of `external function` with the difference that the call is not processed immediately but is delayed until a specific value shall be selected via the fuzz operator. For fuzz testing, such function instances can only occur in value templates.

The `fuzz function` instance denotes a set of values from which a single value will be selected in the event of sending or invoking the `valueof()` operation on a template containing that instance. The `fuzz function` may declare formal parameters and must declare a return type. Since the execution time cannot be predicted, only formal `in` parameters are allowed (e.g. no `out` or `inout`). For sending purposes or when used with `valueof()`, `fuzz functions` must return a value.

Example:

```
fuzz function zf_UnicodeUtf8ThreeCharMutator(
    in template charstring param1) return charstring;

fuzz function zf_RandomSelect(
    in template integer param1) return integer;

template myType myData := {
    field1 := zf_UnicodeUtf8ThreeCharMutator(?),
    field2 := '12AB'O,
    field3 := zf_RandomSelect((1, 2, 3))
}
```

The `fuzz function` instance may also be used instead of an inline template.


Example:

```
myPort.send(zf_FiniteRandomNumbersMutator(?));
```

To get one concrete value instance out of a fuzzed template the `valueof()` operation can be used. At this time the fuzz function is called and the selected value is stored in the variable `myVar`.

Example:

```
var myType myVar := valueof(myData);
```

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 12 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

2.2.2.3 Seed

To allow repeatability of fuzzed test cases, an optional seed for the generation of random numbers used to determine random selection shall be used. There will be one seed per test component. Two predefined functions will be introduced in TTCN-3 to set the seed and to read the current seed value (which will progress each time a fuzz function instance is evaluated).

```
setseed(in float initialSeed) return float;
getseed() return float;
```

Without a previous initialization a random value will be used as initial seed.

2.2.3 TCI Extensions

The above declared fuzz function is implemented as a runtime extension and will be triggered by the TTCN-3 Test Control Interface (TCI) instead of (TRI), as external functions, in order to accelerate the generation by avoiding the encoding of the parameters and return values.


At the moment a template is sent or evaluated by use of `valueof()`, it has to be examined whether it contains fields that should be fuzzed. If so, `tcxFuzzySelect()` should be called by the Test Environment (TE) for each fuzz function instance contained by the template with the actual parameters used for each instance.

2.2.3.1 tcxFuzzySelect

Signature	Value <code>tcxFuzzySelect(in TcIValueList paramList, in TString fuzzOperator, in TFloat seed)</code>	
In Parameters	paramList	The template(s) to be used as parameters by the fuzz function.
	fuzzOperator	The ID of the fuzzing operator or generator to be used.
	seed	The seed to be used for any random decision generation during selection. It will be the value computed by the TE using <code>rnd(getseed())</code> which in turn will be set to the current seed value to be returned by <code>getseed()</code> .
Return Value	Returns the fuzzed value.	
Constraint	This operation shall be called whenever a template that contains fuzz function instances will be sent, or when the <code>valueof()</code> operation is called for it for each such function instance with the values given as actual parameters to that instance as paramList.	
Effect	For the given actual parameters a concrete value will be generated based on the optional fuzzing operator or generator and the given seed value.	

Table 1: Template example

To compute the concrete value based on the given template a randomized approach could be used using the given seed. Also external data fuzzers might be used to achieve better results.

	<p align="center">Initial Security Testing Tools</p> <p align="center">Deliverable ID: D3.WP3</p>	Page : 13 of 81
		Version: 1.1
		Date : 29.6.2012
		Status : Final Confid : Public

An integration of FOKUS Data Fuzzer is planned as the next step. This way using a lightweight Java interface a powerful fuzz engine is integrated. An EMF based interface to the FOKUS Data Fuzzer may provide fuzzing also for the test case generators.

2.3 DO.ATOMS FRAMEWORK

Before a more detailed overview over the case study's conceptual approach in terms of analysing of messages and applying fuzzing operations is given, the do.ATOMS framework should be introduced since this framework serves as the technological backbone of the automotive case study. The artificial term "do.ATOMS" stands for "Automated Testing of Model Based System Design". Following this approach the framework serves as a model based testing (MBT) environment for discreet interconnected embedded systems. Test cases are generated automatically out of a given system model. The system model is based on UML and specifies the structure as well as the behavior of the system under test (SUT). The system model gets processed by a so called model parser – depending on the UML tool used for creating the system model do.ATOMS provides a variety of model parser implementations (e.g. for Enterprise Architect or Rational Rhapsody models). The test cases can be transferred into a test case library and can be edited in an easy to use editor that shows the transformed test cases as workflows. During test case creation the system model parser also tries to parameterize the test cases so they are directly executable on the corresponding SUT test setup. The test case designer also handles test case management, the execution and the reporting as well. The test execution layer of do.ATOMS has a direct access to the interfaces provided by the SUT and follows a service-orientated approach.

2.3.1 Extracting Risk Information

2.3.1.1 CORAS Information

The approach of Dornier Consulting is based on a risk analysis made in CORAS as mentioned in Chapter 3.1. Each diagram in CORAS represents a threat scenario, which can be reached by using several different vulnerabilities. These vulnerabilities are applied with a value from 0.0 to 1.0, where 0.0 represents no protection and 1.0 represents full protection.

For each vulnerability an own connection from attacker to the threat scenario is defined. The connection is of type 'initiates' and is applied with a likelihood. According to the CORAS language specification the 'threat scenario' leads to an 'unwanted incident', which impacts an 'asset'. All connections are applied with a likelihood-value, just like the 'initiates' connection. In Figure 2 the threat scenario 'Uploading modified firmware' is shown.

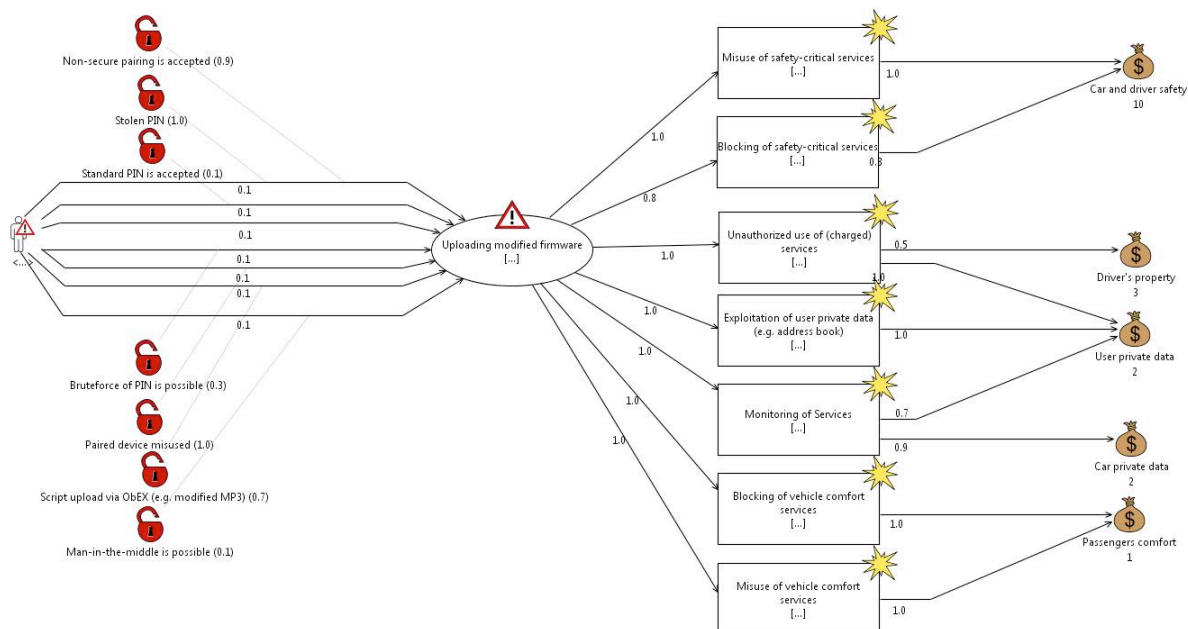


Figure 2: CORAS model for the threat scenario 'Uploading modified firmware'

For this approach the used terms have the following meaning:


- **Likelihood** The likelihood classifies the possibility that an event happens.
- **Vulnerability** By vulnerability a flaw in the system is meant. The value at the vulnerability represents the protection, which exists to prevent such an event. Therefore 1.0 means full protection and 0.0 means no protection at all.
- **Consequence** The consequence is applied to an asset. The value of the consequence represents the dimension of the consequence in respect to security. The higher the values are the stronger is the security impact.

The elements shown in Figure 3 are used in the CORAS diagram. The diagram usually begins with a **threat**, which has an 'initiate' connection to a **threat scenario**. From here a 'leads to' connects one or several **unwanted incidents**. The **unwanted incidents** have an 'impact' connection to the assets.



Figure 3: CORAS legend of used elements

In order to cover the complete consequence of a vulnerability, it is necessary to iterate over each path of the CORAS diagram. The following equation is used:

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 15 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

$$consequence = \sum_{i=1}^{Number\ of\ paths} (likelihood_{LeadsTo_i} \times likelihood_{Impacts_i} \times consequence_i)$$

Based on the values for the vulnerability, likelihood, and consequence a score can be calculated. For this approach the following equation is used:

$$securityscore = \frac{(likelihood \times consequence)^2}{vulnerability}$$

This is a simple approach to categorize the different vulnerabilities. In this approach the highest *securityscore* does not represent the most relevant message. But a high *securityscore* indicates a general relevant message in respect to security.

2.3.1.2 Modeling vulnerabilities

In order to generate test cases based on the system model, it is necessary to map the calculated *securityscore* to the system model. The most uncompromising way would be to model each path of CORAS to the system model. But this is not practical, since some threat scenarios are too complicated to model (e.g. exploits). Therefore the approach requires at least one scenario for each vulnerability to be modeled. These 'CORAS scenarios' describe the way how to reach certain vulnerabilities and are added in a separate package to the system model. With this modification the system model becomes a test model.

For this approach a use case was created for every vulnerability-case. These use cases were applied with a stereotype <<CORAS>> to identify these use cases. A selection of a typical Bluetooth set-up is shown in the Figure below.

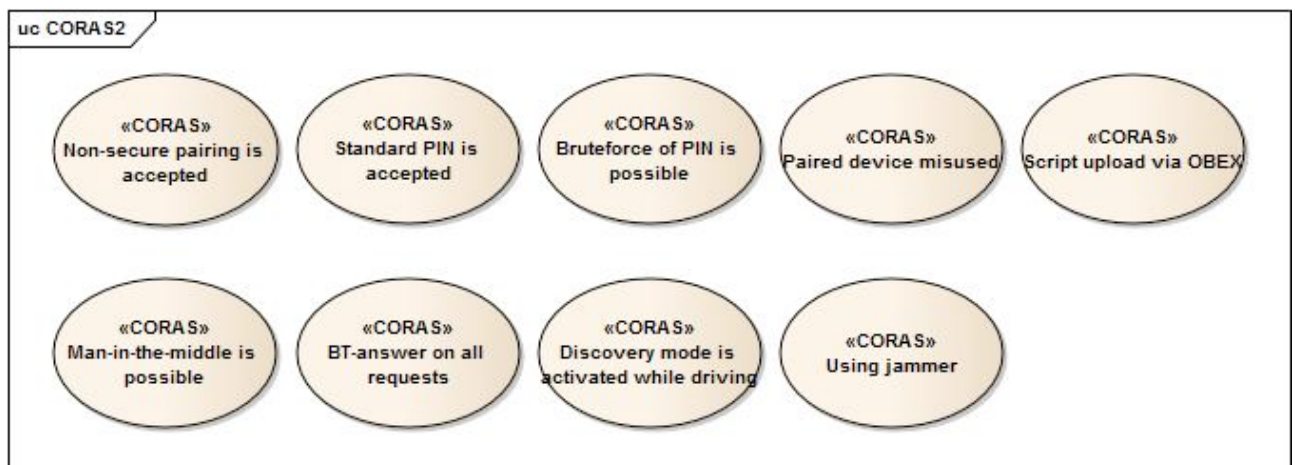


Figure 4: CORAS scenario within the system model

Since this case study is supported by a system model based on sequence diagrams, the vulnerability-cases are modeled as a sequence diagram as well. The following figure shows a sequence diagram for the vulnerability-case 'Standard PIN is accepted'. It is similar to a normal use case until the point, where the access is permitted to the intruder. By reaching the access of the system, the vulnerability is reached and the use case was successful. These use cases can be used for testing purpose. It will test the system under test for known flaws. In combination with a test pattern approach the security test of known flaws can get very efficiency.

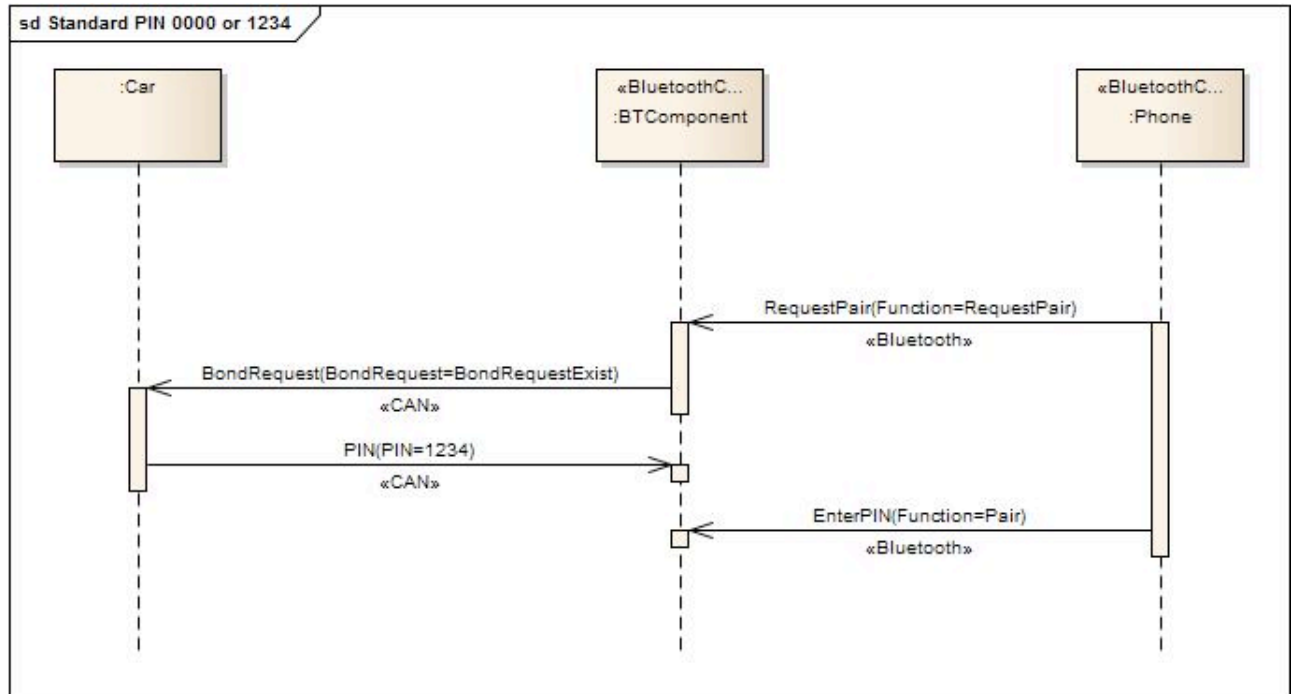


Figure 5: CORAS scenario modeled in the system model as a sequence diagram

In the following figures the hierarchy is shown. The CORAS use case acts like a package and contains the sequence diagram. In this case a use case could contain several sequence diagrams.

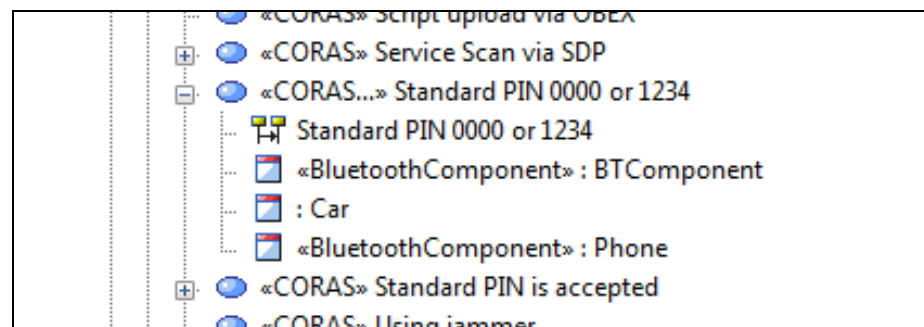


Figure 6: Sequence diagram sorted under the CORAS use case in the system model hierarchy

The messages used in the sequence diagram are identical with the messages used in the system model. This means that the messages even have the same internal identification number. Therefore a direct association from the messages used in the system model to the CORAS information is possible. With this approach messages of the global system model can be applied with a *securityscore*.

2.3.1.3 Prioritization of messages

For testing purpose the *securityscore* is used to calculate a priority of the different messages. Together with the occurrence of a single message in the system model a priority is calculated, which tells the testing

framework the relevant messages in respect to security (Figure 7). This priority influences the test case generation.

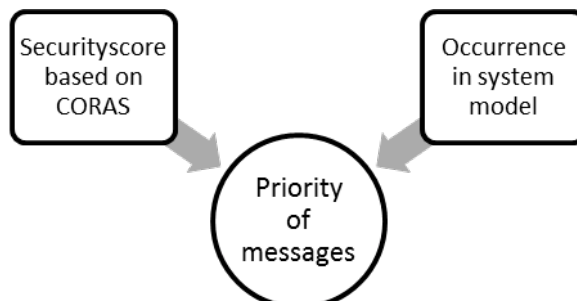


Figure 7: The priority of a message is a combination of the introduced securityscore and the occurrence of a message within the system model.

The presented approach allows mapping the CORAS risk analysis to the system model. The introduced equation for the *securityscore* is a simple way to establish a security ranking between the different vulnerabilities of the CORAS risk analysis. Based on the analysis the scenarios can be extracted to the system model. The containing messages can aggregate the *securityscores* of the CORAS scenarios, in which they are included. In combination to the occurrence of a message within the system model a stable priority order can be introduced in respect to security.

2.4 APPLICATION PROGRAMMING INTERFACE FOR ACTIVE SECURITY TESTING

As indicated in the D3.WP1 document, from the THALES case study on radio protocols application security testing, generic API has been defined in order to implement attacks.

The proposed API for active testing messages is the following:


Field		Type	#byte
Header 2bytes			
length		Length	1
Layer attacked	Specific attack	Layer_attack	1
Attack type		Typ_attack	1
Time of beginning of attack		Begin_attack	4
Padding set to 0		Padding11	4
PDU identifier	Message field assignment attack	ldf_pdu	1
PDU value		Val_pdu	1
Field value		Val_field	1
Début forcage		Begin_forc	4
Padding to 0		Padding2	4
TOTAL			24

Parameter : **Header**

Comment : Header which gives the API version.

Parameter : **Length**

Comment : Message length

	Initial Security Testing Tools Deliverable ID: D3.WP3	Page : 18 of 81 <hr/> Version: 1.1 Date : 29.6.2012 <hr/> Status : Final Confid : Public
---	--	--

<i>Parameter :</i>	Layer attacked	Layer_attack
<i>Comment :</i>	Define the application layer attacked	
<i>Parameter</i>	Attack type	Typ_attack
<i>Comments :</i>	Specify the attack type	
<i>Parameter</i>	Time of beginning of attack	Begin_attak
<i>Comments :</i>	Timing reference, as timing slot	
<i>Parameter</i>	Identifiant PDU	Idf_pdu
<i>Comments :</i>	Specify the affected PDU	
<i>Parameter</i>	PDU value	Val_pdu
<i>Comments :</i>	Specify the impacted field	
<i>Parameter</i>	Field value	Val_field
<i>Comments :</i>	Value of the impacted field	
<i>Parameter</i>	Beginning of forcing	Beg_forc
<i>Commentaires :</i>	Number of slots the field value is forced	

This message API is currently the one used for active testing specification on the THALES use case, and is sufficient to specify and validate the properties specified in D3.WP1. It will be able to be extended in the project following.

3. IMPLEMENTATION OF A TRACE MANAGEMENT PLATFORM (FOKUS)


The following sections describe the planned architecture of a traceability tool. One part of the tool is the core platform that handles the traces and the trace queries and will be called the Trace Management Platform. The other part is all supported traceable tools and form with the Trace Management Platform the Trace Management System. The architecture of the Trace management System and the implementation of a Trace Management Platform will be described in the following.

3.1 ARCHITECTURE OF THE CREMA TOOL

The Itemis AG is developing a tracing tool named CReMa [10] implemented as an Eclipse application. This CReMa tool is used as the starting point for the development of a trace management platform (TMP) [20] for the DIAMONDS project.

The CreMa tool is written in Java and designed in different Eclipse plugins which reflects also a sort of a component architecture. We extend and modify these plugins/components in order to adapt CReMa to the requirements of the DIAMONDS project. Below is a list of the main CReMa components that we had to extend and modify:

- **CReMa Core:** The core of the CReMa tool and the core for our TMP platform.
 - Two kinds of **trace model management** are supported: a local version to store the trace model in a local file saved in Ecore xml format, and a distributed version to use a ECDO (Connected Data Objects) [11] server to save the trace objects as CDO objects in case of a multi-user-scenario.
 - For the user a **preference table** is provided to set the kind of trace model management and define trace (connector) types and trace point roles.
 - In this plugin a **query language** based on JoSQL [12] is implemented. It will be used for filter options and arrangements of results from the tracing explorer view (see below, in **User Interface** part). For the DIAMONDS project, we want to define a query language to support

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 19 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

specific queries for elements in all supported models (see [20] , Chapter Traceability Concepts, Concept of a Query Language). The query language in this plugin is only used to handle queries of explicit created traces by the CReMa tool. However, this query language concept can be used as the basis for the TMP query language.

- In order to bind an Eclipse editor tool to the trace framework such that the traceability tool features can be used, an extension point (**tracepointprovider**) is defined. For this extension point a belonging interface (ITracePointProvider) has to be implemented.
- **Extensions to the CreMA Core:** These plugins implement the above extension point to enable all kinds of traceability functions. If an editor tool is required for the tracing tool, this extension point has to be used by implementing the above mentioned interface class. CreMA inherently supports the editor tools Papyrus [13] and ProR[14]. The Papyrus tool is an UML-editor to model all UML2 specified diagrams like class diagram, behaviour diagram and so on. In addition, it enables modelling based on SysML [16]. *ProR* is an editor tool to model requirements based on the ReqIF [15] standard by OMG. The DIAMONDS project brings in additionally the CORAS editor tool to model risks, vulnerabilities, threat scenarios and vulnerability treatments for a system. Furthermore, a test modelling tool will be supported.
- **Tracing Model:** The trace model is modelled with Ecore and used to generate Java classes.
- **User Interface:** The CreMA user interface is separated in two editor views:
 - The **tracing editor view** is for editing, creating and deleting traces. With a selection of an element in a supported editor you can set this element to a trace as one end point. With setting two end points the trace can be created. The trace end points can be assigned with role types and the trace can be assigned with a trace type according to the trace model presented in [20]. The list of available types for role type and trace type can be set by the user (over the corresponding Eclipse Preferences).
 - With the **tracing explorer view** it is possible to navigate through the whole trace model. The second feature is to create a filter to hide traces with specified trace type or trace end points.

3.2 REQUIREMENTS FOR USER INTERFACE EXTENSIONS

One part of our development is given by the user interface extensions for usability, i.e. navigating, creating and deleting traces directly in the belonging editor pane. The corresponding interface for the extension point in question has to be implemented to realize the features for the belonging editor tool as illustrated in Figure 9. The following features, illustrated in **Error! Reference source not found.**, were implemented in the scope of DIAMONDS:

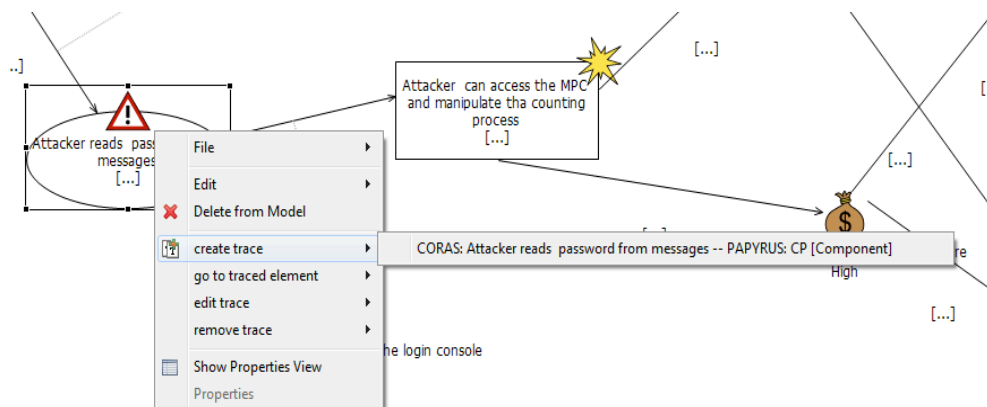



Figure 8: User Interface Extensions

- **Create trace** supports a user friendly creation of traces. First, the user selects two elements in the editor pane(s). In the popup option for creating a trace all relevant elements in the supported editors

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 20 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

are shown and can be selected. That way the user can choose from more than one element in more than one editor pane. With this feature the test manager can easily create trace relations between two models. Within the scope of future developments, we are going to improve this user interface such that only valid trace end points are shown for the selected element within the popup menu. In order to realize this, the preference table in the **CreMA Core Module** has to be modified.

- **Delete trace** shows the full list of all explicitly defined traces involving the selected element. The user can select a trace from this set and delete it.
- **Edit trace** loads the trace in the tracing editor view. The tracing editor view will also be shown in the actual Eclipse if not already shown.
- **Go to trace point** is the navigation option and shows the full list of all traced elements to/from the selected element. With selecting one of the shown elements, the related editor will be set in the active editor pane and the selected element will be focused.
- **Copy traced elements** is the extended version of the *Copy and Paste* option in the editor, in order to both - duplicate the elements in the model and create new traces from old traces. The one trace point of every copied trace is the copied element and the other is the original end point of the original trace.

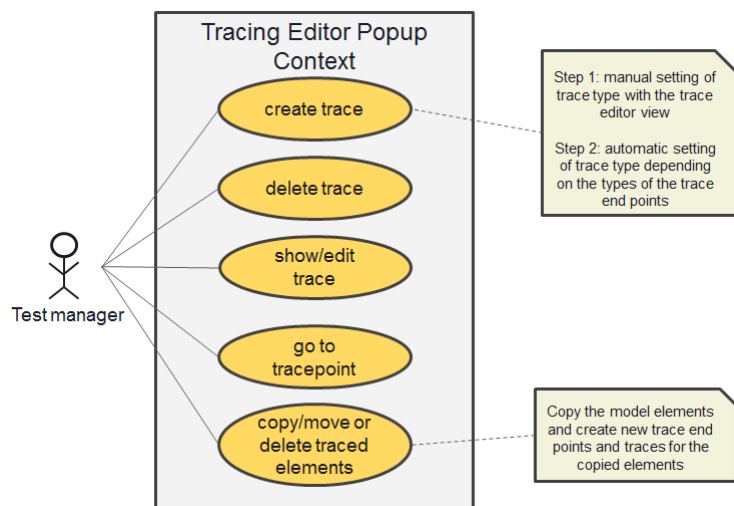


Figure 9: Use case diagram for trace navigation and administration extensions

3.3 REQUIREMENTS FOR ADDITIONAL EXTENSIONS

The user interface extension and other Trace Management Platform features require additional extensions in the Tracing Editor View and the tracing core. These extensions are shown in Figure 10:

- In the **preference table** the user defines trace types and trace end point roles. Over an extension, the user can allow traces by defining meta-traces on the base of valid trace end point types and assigned trace type name. Additionally, the preference table can be set with an automatic information model read-in mechanism.
- The test manager must be able to annotate hints, comments or additional information for a single trace. For this case the attribute **comment field** has to be added to the trace element in the trace model. Additionally the Tracing Editor View has to be extended with a text field to set a comment.

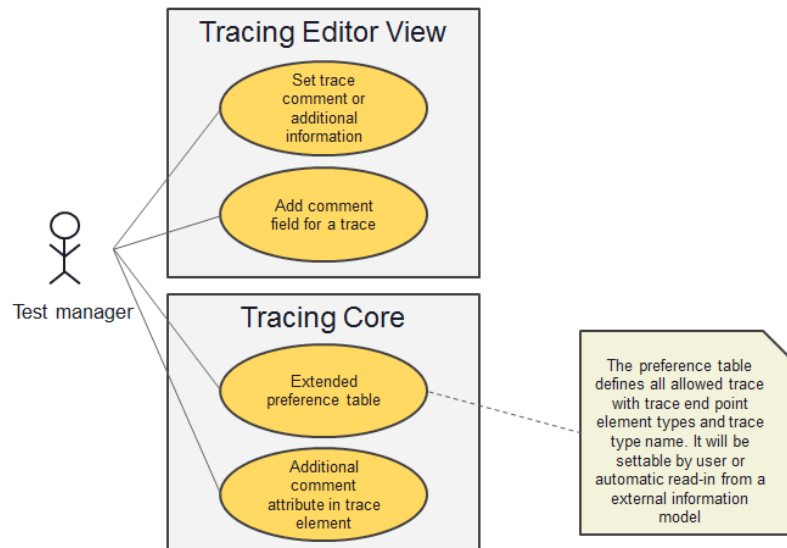


Figure 10 Use case diagram for TMP trace editor view extensions

3.4 ARCHITECTURE OF THE TRACE MANAGEMENT SYSTEM

The CreMA tool is the first step for the TMP to manage traces. The second step is to create adapters for model tool extensions as described in the motivation and definition in [20]: Traceability Methodology, Description of the Target Realization of the Trace Management Platform (TMP).

The planned architecture of the trace management system including the TMP, the domain adapters and model adapters is shown in Figure 11. The most important enhancement is that the editor tools will no longer be directly connected to the tracing system, but only indirectly. As described in [20], the individual tools are associated with their corresponding modeling concepts to certain domains. The Editor Tool Adapter serves as a gateway between the tool specific model and the generic domain model. This domain model is an artifact that interfaces over the Domain Adapter between TMP and the tool (over the Editor Tool Adapter). The editor tools are registered in the related domain adapter. Additionally, each domain adapter is registered in the TMP. Each domain adapter is implemented as Eclipse plugin and defines an extension point for an editor tool.

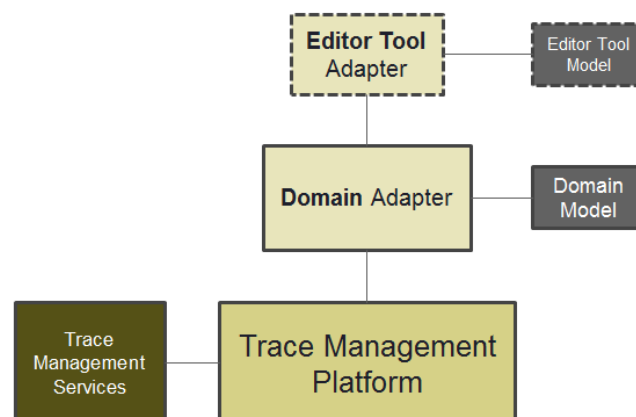



Figure 11: Overview of the Trace Management System

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 22 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

Additionally, Trace Management Services can be implemented and registered within the TMP. They can be complex services like “automatic test case generation” that analyzes all traced models to create tests based on the requirements involving information from the corresponding vulnerabilities and threat scenarios and the concerned system elements. Furthermore, simple services like fulfilling analysis type of requests coming from the test manager are possible, for example “Mark all unwanted incidents affected by test case <xyz>”.

Every new added domain needs only to be registered to the TMP extension point. The same is required for every new editor tool to be registered to the domain adapter extension point. Additionally, in case of adding new editor tools the services do not need to know anything about the involved tools. The queries for the service functions are just using the abstracted domain model to involve information from the specific tools and utilize it in their logic. Hence, there is no need for a service developer to understand the tool specific models. It is enough to rely on the concepts captured in the domain model. The translations between the tool specific model and the domain model are realized by adapters as illustrated in Figure 11.

It is important for the services that all model instances are globally available. In that context, given that the current platform acts based on a trigger, the selection of an element in an editor allows to immediately utilize an available trace and follow the relationships between the involved elements. In the case of a service for automatic test case generation, elements with specified types in specific model domains have to be collected and the implicit and explicit associations (as traces) have to be analyzed and exploited. The related model instances for the overall picture have to be globally available. Therefore, a registry mechanism to **register all related instances** is required. These enables additional use cases which are shown in Figure 12.

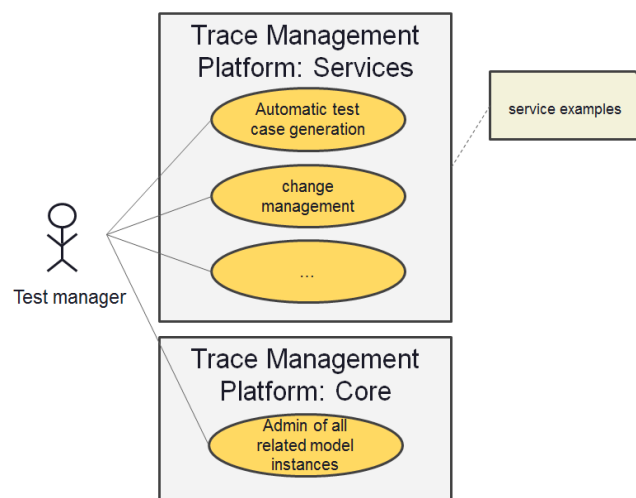


Figure 12: Use Case Diagram for TMP services and TMP Core Features

4. DATA FUZZING LIBRARY (FOKUS)

Today, data fuzzing is a widely accepted and performed technique for security testing. There are many different fuzzers available, some are commercial, many are open-source. The main difference between the commercial and the open source fuzzers is that the latter are often developed for a specific protocol, while the 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 ([17], [19] p. 242) 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 presented data fuzzing library solves this problem. It combines the core of several open source fuzzing tools – their test data generators – and makes them available for other tools over a single interface. Test data generators, that are fuzzing heuristics, consist of fuzzing generators and fuzzing operators. Fuzzing generators are producing values from a specification while fuzzing operators are modifying valid values.

4.1 EXISTING OPEN SOURCE FUZZING TOOLS

Because of the huge number of tools (see [18] for a list of tools available in 2005), a study of open source fuzzing tools was conducted. The study investigated fuzzing tools w.r.t the following aspects:

- **target of fuzzing attacks:** This could be a specific protocol, e.g. SOAP, or for instance Java classes or regular expressions.
- **last source code update:** This information is used to estimate whether the fuzzer is further developed.
- **category of fuzzer:** Random-based, block-based, generation- or mutation-based.
- **existence of a library interface:** If the fuzzer can be used as a library through an interface, its integration would be easier.
- **fuzzing heuristics:** What kind of fuzzing heuristics are implemented. That is the most interesting part because the fuzzing heuristics constitute the functional core of the library.
- **license:** Does the license of the fuzzer allow an integration within the planned library.

After evaluating ten open source fuzzing tools, three were selected for the initial implementation of the fuzzing library:

- **Peach** is an open source smart fuzzer that is under active development since 2006 and also commercially offered by Déjà vu Security. It can perform generation and mutation-based fuzz testing employing a data model and a state model. It has a rich set of generators and operators, e.g. for Unicode string, hostnames, variation and extreme numbers, i.e. going to the edge of an interval.
- **Sulley** is a block-based fuzzer for generation and mutation-based fuzzing. It is under development since 2008, and like Peach, it also has various fuzzing generators and operators. Additionally, Sulley uses many fuzz testing values from SPIKE, a generation-based fuzzing framework.
- **OWASP WebScarab** is part of Open Web Application Security Project, a non-profit organization that is focused on improving the security of application software. WebScarab is a tool for analysing traffic of web applications and provides a fuzzing plugin. That fuzzing plugin is able to generate values from regular expressions. This is interesting because it offers two possibilities: on one hand, the user can specify a regular expression for invalid values avoiding the manual specification of all the values. On the other hand, the syntax for valid values can be specified, and by modifying this, invalid values can be generated.

First, the data fuzzing library provides a uniform interface to access the fuzzing heuristics of the above mentioned tools. This interface and the library itself is open for further extensions. The chosen fuzzing heuristics are shown in Table 2.

Fuzzing Heuristic	Peach	Sulley	OWASP WebScarab
-------------------	-------	--------	-----------------

Fuzzing Heuristic	Peach	Sulley	OWASP WebScarab
Strings			
StringCaseMutator	O		
UnicodeStringsMutator	G		
UnicodeBomMutator	G		
UnicodeBadUtf8Mutator	G		
UnicodeUtf8ThreeCharMutator	G		
StringMutator	G		
PathMutator	G		
HostnameMutator	G		
FilenameMutator	G		
BadIpAddress	G		
BadNumbers	G		
BadDate	G		
BadTime	G		
Numbers			
FiniteRandomNumbersMutator	G		
String Repitition		O	
SQL Injection		G	
Command Injection		G	
Format String		G	
Delimiter	G		
RegExExpander			G/O
Numerical Edge Case Mutator	G	G	
Numerical Variance Mutator	O		
LongString		G	

G = Generator (generates values from specification)

O = Operator (generates values by modifying valid values)

Table 2: Chosen Fuzzing Heuristics from Selected Fuzzers


4.2 USE CASES

Two main use cases for the fuzzing library were identified:

- **Using fuzz testing in SUT specific test execution environments.** Most fuzzing tools integrate the definition of the data format, the test execution and test verdict arbitration. This requires the tester a) to connect the fuzzing tool to the SUT, which could mean a significant effort, especially for embedded systems, and b) to get familiar with the syntax of the data and to specify its format for each fuzzing tool.

The data fuzzing library faces these drawbacks by providing a unified interface for accessing fuzzed values without losing the power of fuzzing heuristics from different fuzzing tools. The test execution environment can request fuzzed values from the library and sent them to the system under test using its specific connection to the interfaces of the SUT.

- **Extending test tools with fuzz testing capabilities.** Fuzz testing is one test method beside others. Other test methods are already supported by test tools implemented in an existing test process. To integrate fuzz testing into an existing test process can lead to substantial effort because new tools have to be adapted and fitted to the whole test process. The fuzzing library enables to extend

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 25 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

existing test tools with fuzz testing capabilities acting as a library from which existing tools can request fuzz testing values.

4.3 REQUIREMENTS

In order to make a fuzzing library widely adoptable, the following requirements were figured out:


- **platform independence:** The library shall be available on many different platforms. At best on as many platforms as are used for existing test tools and test execution environments.
- **independent from programming languages:** As for platforms, the library shall be accessible from as many programming languages as possible. Hence, the library should not use a language specific interface but should be open for many programming language.
- **largely independent of type representation:** It should be possible to describe and fuzz a wide range of different data types without the limitation of the programming languages used for implementing the library.
- **efficient to use:** The user shall have the possibility to specify which fuzzing heuristics shall be used. This allows adjusting the library to fit specific requirements, e.g. only generate fuzz testing values based on Unicode. Because fuzzing generates a large set of values, the user must have the possibility to request a certain number of fuzz testing values.
- **transparent:** The fuzzing library shall tell its user which fuzzing heuristics were used, so more values only from a certain fuzzing heuristic can be requested. This is useful if a fuzz testing value generated by a certain fuzzing heuristic revealed a weakness in the system under test.
- **repeatability:** Fuzz testing is often a random-driven approach meaning that values are generated in partial randomly. The library shall support the repeatability of such randomly generated values, e.g. for regression testing.
- **extensibility:** Time is going further, and new fuzzing tools with new fuzzing heuristics will come. The library shall be extensible to meet these upcoming developments.

4.4 IMPLEMENTATION APPROACH

In order to meet the requirement for platform independency, the fuzzing library is developed on top of the Java Platform. Java is supported on many operating systems and seems to be appropriate to achieve the goal of availability on various platforms.

To preserve platform independence as achieved within Java and to minimize dependencies, the fuzzing operators taken from the fuzzing tools (see section 4.1) are reimplemented in Java. This brings benefits for the performance of the library since no integration of Python code (for Peach and Sulley) is required. The Java integration layer for Python, Jython [24], typically requires some significant time for initialization which might hamper the performance of the data fuzzing library. In addition, the realization of the fuzzing heuristics in pure Java improves the maintainability because only one programming language is used for the implementation of the whole library. The reimplementations of the fuzzing operators seems also to be appropriate because the core functionality of a fuzzing operator is generally quite simple and normally the fuzzing heuristics are hard-wired within the enclosing framework. Additionally, for the Jython integration of Python code into Java, there is additional overhead because a Java interface must be written for each Python class, and that Java interface in turn must be referenced in the Python class.

Fuzzing is often a random-driven approach. Fuzzed values are generated under the influence of randomness without following a certain set of deterministic rules. This impedes regression testing because every time fuzzed values are generated they differ from the previously generated ones. To enable regression testing, the fuzzing library returns a seed that can be used for later requests in order to retrieve the same values. Thus, the requirement for repeatability is fulfilled.

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 26 of 81
		Version: 1.1
		Date : 29.6.2012
		Status : Final Confid : Public

4.4.1 Interfaces Provided by the Library

The requirement of language independency has a big impact on the interface provided by the library. It must be accessible from different programming languages. Additionally, the interface must be able to handle many different representation formats. Hence, XML seems to be an appropriate choice for accessing the library and receiving its output.

In order to receive fuzzed values from the fuzzing library, a request must be submitted to the library. Such a request contains the relevant information of a type that shall be fuzzed, e.g. valid lengths and null termination for a string, as shown in Figure 13. Additional information are the number of values to be retrieved (attribute maxValues) as well as a name acting as a user-defined identifier (attribute name) that can be used for referring this type.

The following types are supported:

- **Strings:** Different kinds of strings, including filenames, hostnames, SQL query parameters.
- **Numbers:** Integers and floats, signed or unsigned with different kinds of precisions.
- **Collections:** Lists and sets. The type of each element is specified by referring one of these four types (strings, numbers, collections, or data structures) using the value of the name attribute.
- **Data structures:** Enables the specification of records with several fields where the type of each field is specified by referring one of these four types (strings, numbers, collections, or data structures) using the value of the name attribute.


```
<string name="SimpleStringRequest" maxValues="10">
  <specification type="String" minLength="1" maxLength="5" nullTerminated="true"
    encoding="UTF8" />
  <generator>BadStrings</generator>
  ...
  <validValues>
    <value>ABC</value>
    ...
    <operator>StringCase</operator>
    ...
  </validValues>
</string>
```

Figure 13: Excerpt from an XML Request File

Along with the specification of the data type, it is possible to specify which fuzzing heuristics shall be used and which valid values shall be fuzzed. This is of particular interest if a specific kind of invalid input data is needed, e.g. based on Unicode strings. This allows it to efficiently use the fuzzing library to get certain fuzzed values.

The response of the fuzzing library is also an XML file whereof an excerpt is shown in **Figure 14**. The response contains the fuzzed values according to the basic request type (in this case string) given by the request. Moreover, there are two new attributes for the tag string. moreValues denotes if further values than the enclosed can be retrieved from the library. Finally, id stands for a UUID that is necessary to retrieve further values after receiving the first response from the library.

The fuzzed values are complemented by information how the fuzzed values were generated by the library. They are grouped by the employed fuzzing generators – for fuzzed values that are generated along the type specification. Moreover, the employed fuzzing operators, and the valid values they were applied to. This makes the generation of fuzzed values transparent to the user of the library, and further request of the

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 27 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

fuzzed values generated by specific fuzzing operators if a previous value revealed some abnormal behaviour of the SUT.

```
<string name="SimpleStringRequest" id="ca53abee-0719-43da-a70d-96d61931fb08"
  moreValues="true">
  <generatorBased>
    <generator name="BadStrings">
      <fuzzedValue>+]s}9$# *Y</fuzzedValue>
      <fuzzedValue>0$2)v3D^U1_{X7x,Us\\</fuzzedValue>
      ...
    </generator>
    ...
  </generatorBased>
  <operatorBased>
    <operator name="StringCaseOperator" basedOn="ABC">
      <fuzzedValue>abc</fuzzedValue>
      <fuzzedValue>aBc</fuzzedValue>
      ...
    </operator>
    ...
  </operatorBased>
</string>
```

Figure 14: Excerpt from an XML Response File

The format of the request file as well as the format of the library's response file is specified using an XML schema. The parser and serializer for the XML are generated from those XML schemata using the Eclipse Modelling Framework (EMF).

4.4.2 Architecture

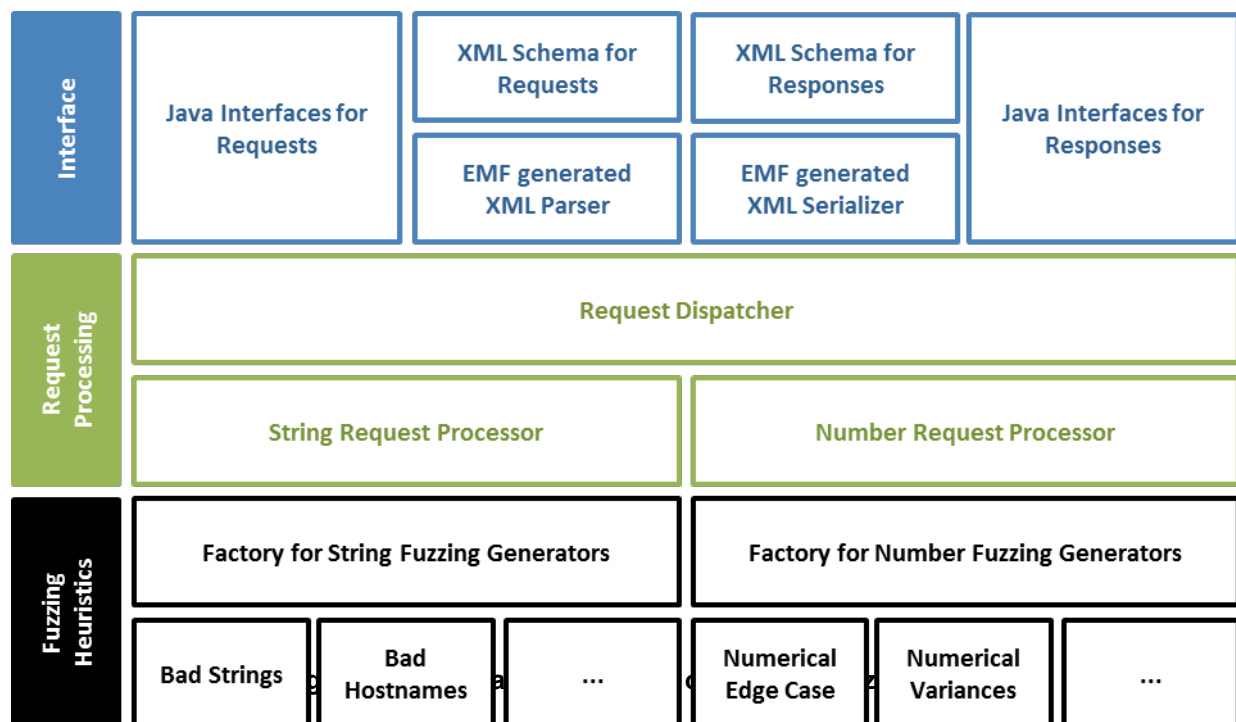
Figure 15 depicts the architecture of the fuzzing. It consists of 3 layers:

- **Interface:** The library can be accessed using two interfaces independent from each other. The language independent way is through XML for creating a request and getting the response. As described above, XML schemata were defined for determining the syntax of requests and responses. These schemata were used to generate the proper XML parser for request files and the proper XML serializer for response files. However, the XML schemata were used to define the general structure of XML requests on a high level of abstraction, in order to keep the schemata manageable, and allow the extension of the fuzzing library without the need for changing the XML schema when adding new fuzzing heuristics. The second way to access the library is by directly using the Java interfaces thereby saving the time for parsing the XML request and response. For the direct access via Java there are interfaces for data objects according to the information that can be submitted via XML requests and responses.

Whichever way was selected by the user to access the library, the information about the requests is delivered to the request processing layer. After the requests are processed, the interface receives the results from the request processing layer, creates the XML response (employing the EMF generated XML serializer) and delivers the response from the library to the user.

- **Fuzzing Heuristics:** This layer aggregates the various fuzzing generators and operators reimplemented from the selected fuzzing tools (Peach, Sulley, and OWASP WebScarab). They are grouped by their types (string, number, collection, data structure). For each type, there is a separate factory for generators and operators that creates instances of them according to the specification given by the request. Thus, only suitable fuzzing heuristics are generating fuzzed values.

- Request Processing:** This layer acts as a broker between the interface and the fuzzing heuristics. A request dispatcher receives a bunch of requests from the interface and passes them to type specific request processors, e.g. to a string request processor. Each type specific request processor handles one request by employing the fuzzing heuristics that match the type specification of the request. The results of the fuzzing heuristics are then returned for building the response to the request. That is, the request dispatcher collects all responses from the different, type specific request processors, and gives the aggregation of all responses to the interface layer.



4.4.3 Extending the Library with New Fuzzing Heuristics

As described above, there are two kinds of fuzzing heuristics: **fuzzing generators** that generate fuzzed values based on a specification, and **fuzzing operators** that modify existing, generally valid values. Both kinds of heuristics are supported by the library. Fuzzing generators simply constitute large lists of certain values that are known to have the capability to expose implementation weaknesses. Because of their size, they require a lot of memory at runtime. To overcome this memory issue, two strategies are realized: First, the fuzzed values of the generators are static members of the corresponding classes, in order to avoid duplication during instantiation of a generator class. Secondly, the values are not used as is but generated at request time. Mostly, there is an underlying pattern for the fuzzed values, e.g. many values of the generator Bad String consist of certain characters that are repeated a power of 2 times. Hence, such a value could be very long. In that context, the memory consumption could be limited by generating the fuzzed value when requested. This is not only valid for fuzzing generators, but also for operators, whereas the length of the valid values, fuzzing operators are applied to, is unknown and could be much larger. For that reason, as shown in **Figure 16** an interface called `ComputableList` is implemented by an abstract class `ComputableListImpl` that is derived from the `AbstractSequentialList` being part of the Java runtime environment. The abstract class `ComputableListImpl` is the base class of all fuzzing heuristics. Its iterator (`ComputingIterator`) does not iterate a list of values but calls the method `computeElement(int)` in order to obtain a value. Therefore, a value can be calculated when requested avoiding generating all values when the fuzzing generator or operator is instantiated.

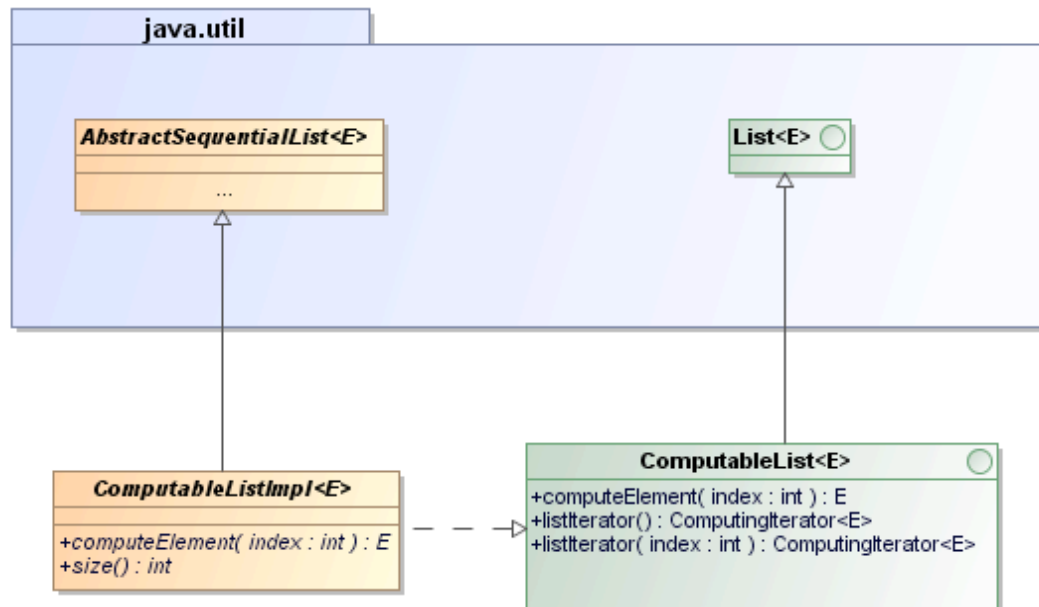



Figure 16: The Class `ComputableList` and its Relationships

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 30 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

5. FUZZ TEST GENERATOR FOR UML SEQUENCE DIAGRAMS (FOKUS)

Behavioural fuzzing of UML sequence diagrams is a new approach for creating fuzz test cases. Thereby messages are rearranged and constraints that affect the control flow within a sequence diagram are modified. Because of the novelty of the approach, there is a need for a tool to evaluate the effectiveness of the approach. The Fuzz Test Generator for UML Sequence Diagrams is a tool that implements the approach presented in [20].

5.1 USE CASES

The main use case for such a test case generator is to be integrated in an existing testing tool in order to extend it with behavioural fuzzing capabilities. That combines the advantages of both, the test tool and the tester's experience in using it, and the effectiveness of a new test technique, which may improve the overall test process.

5.2 REQUIREMENTS

An important goal for a test case generator is that it should be widely adoptable. The following requirements are resulting from that goal:


- **Support for an open, full featured interchange format for UML models** that is the OMG standard XML Metadata Interchange (XMI). That standard is supported by many modelling tools and frameworks [23].
- **Configuration of the test case generation.** The test tool should have the ability to steer the process of test case generation to achieve certain goals defined by the tester, e.g. to fuzz only certain sequence diagrams or certain parts of sequence diagrams.
- **Transparency of the test case generation.** Sequence diagrams can be confusing, since containing many messages and control structures can make it difficult to identify the differences in comparison to the original, unfuzzed sequence diagram. The test case generator should denote which parts of a sequence diagrams were affected by the fuzzing heuristics in order to easily identify what kind of invalid behaviour revealed a weakness.

5.3 IMPLEMENTATION APPROACH: ARCHITECTURE

In this section, the architecture of the fuzz test case generator is presented by a description of the process of test case generation.

Figure 17 depicts the overall architecture of the test case generator. It consists of 4 components:

- The interface of the test case generator is twofold: It consists of the **interface for the model** and the **interface for configurations**.
- The **fuzzing heuristics** that constitute the core of the test case generator.
- The **fuzzing strategies** controlling how fuzzing heuristics are applied to the system model in order to generate behavioural fuzzing test cases.

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 31 of 81
		Version: 1.1
		Date : 29.6.2012
		Status : Final Confid : Public

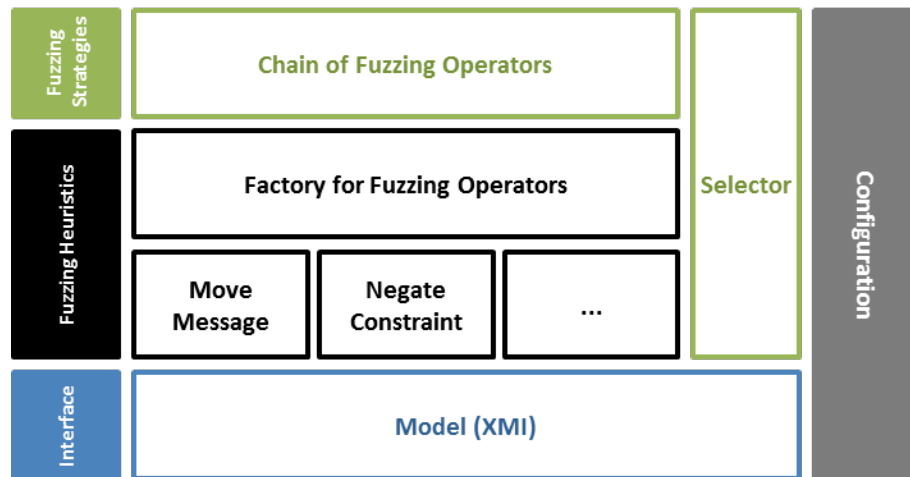


Figure 17: Illustration of the Architecture

5.3.1 Interfaces

The test case generator for UML sequence diagrams has two interfaces:

On one hand, there is an **interface for the model** (blue box in


Figure 17) whose sequence diagrams, or rather interactions, shall be used for generating test cases employing behavioural fuzzing. These are the interactions from the system model, which can be enriched with information what to fuzz by using combined fragments to identify these parts. This model is represented by an XMI file following the UML mapping to XMI developed by the Eclipse UML2 project [22]. The Eclipse UML2 project provides an implementation of the UML 2.x metamodel on top of the Eclipse Modelling Framework (EMF) and is supported by a wide range of tools [23] including the Eclipse Papyrus project, an editor for EMF based models with full support for UML2. On the other hand there is an **interface for the configuration** (grey box in

Figure 17) of the test case generator. This configuration is represented by a Java properties file containing key value pairs. In this configuration file, the user of the test case generator defines

- where to find the system model based on which test cases should be generated,
- the destination where to save the fuzzed model,
- the lifelines that represent the SUT and those which are under control of the test component,
- the fuzzing heuristics to be used,
- how many fuzzing heuristics shall be applied to one sequence diagrams, and
- how many test cases shall be generated at most.

5.3.2 Fuzzing Heuristics

As mentioned above, the test case generator uses the data format and implementation of the Eclipse UML2 project for parsing UML models represented by the XMI data format, as well as to programmatically access the model for the purpose of test case generation. Eclipse UML2 is based on Eclipse EMF that can also be used for generating code for tools that operate on EMF models. In order to enable that, EMF provides a set of interface and classes for editing EMF models, the Eclipse Command Framework. This framework forms the base of the fuzzing heuristics. In particular, the class *AbstractOverridableCommand* is a class that is used as a base class for all fuzzing heuristics that are the fuzzing operators. Because fuzzing operators modify the model, EMF commands and their belonging classes seem to be the best starting point for

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 32 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

implementing it. On one hand, fuzzing operators are actually commands similar to those from model editors with the difference that they are not applied by the user to the model but by a fuzzing strategy in order to generate fuzz test cases.

However, there are some differences and constraints:

- Similar to editor commands, fuzzing operators are used to modify a model in a distinct manner in order to build up a sequence diagram representing an invalid use of the system under test.
- They are applied on elements of the model, or rather on elements of interactions (where sequence diagrams constitute a view of an interaction). The elements that are modified are selected from the system model, but have to remain untouched because these and the enclosing interaction are the base for all test cases. Hence, the other fuzzing operators need the unmodified interaction representing a valid sequence of behaviour. Thus, the elements a fuzzing operator is applied to are originally found in the system model's interaction, but the fuzzing operator is actually applied to a copy of the element and its enclosing interaction. Therefore, the fuzzing operator must have the ability to distinguish the interaction and its copy. The model elements are taken from the system model's interaction, but the fuzzing operator is applied to model elements of its copy.
- This dichotomy between the system model's interaction and the copy of this interaction is tightened by the fact that an interaction is not necessarily self-contained because it can reference other interactions or it can be referenced by other interactions. Therefore, it is not sufficient for a fuzzing operator to consider the interaction where the model elements it shall be applied to is enclosed in, but also the referenced interactions as well as the referring interactions. This concerns for instance the fuzzing operator Move Message, because the position where a message can be moved to can be the enclosing interaction, the interactions referring the enclosing interaction, or the interactions that are referenced by the enclosing interaction.
- In contrast to editor commands that are directly executed by the user, fuzzing operators do not need to be undone.

EMF commands has a basic set of methods that are also useful for the fuzzing operators: methods for requesting if a command can be executed (`doCanExecute` in **Figure 18**), for preparing its execution (`prepare` in **Figure 18**) and for executing it (`doExecute` in **Figure 18**). On the other hand, they can be composed by using the EMF class `CommandStack` to execute a bunch of commands. Although providing a good basis for the fuzzing operators, an abstract subclass `FuzzingOperator` was created from `AbstractOverridableCommand` to extend it with methods needed in the context of behavioural fuzzing. These methods are the following:

- `addElement()` and `addSelection()` that are used to determine the model elements the fuzzing operator is applied to.
- `translateSelectionToInteraction()` and `setInteractionToBeFuzzed()` are methods that are used when a fuzzing operator shall be applied to its selected model elements. In that case, the counterparts of the selected model elements from the unmodified interaction are identified, which is the purpose of the method `translateSelectionToInteraction()`.
- `isApplicableToElement()` is used in order to identify if a fuzzing operator can be applied to a certain model element.
- `createWithAllParameters()` is necessary in order to obtain all possible expressions of a fuzzing operator. For instance, in case of the fuzzing operator Move Message the methods returns for one selected message a set of fuzzing operators where each fuzzing operator moves the message to a certain position within the enclosing interaction. The set of all fuzzing operators returned by the methods covers all possible positions where a message can be moved to.

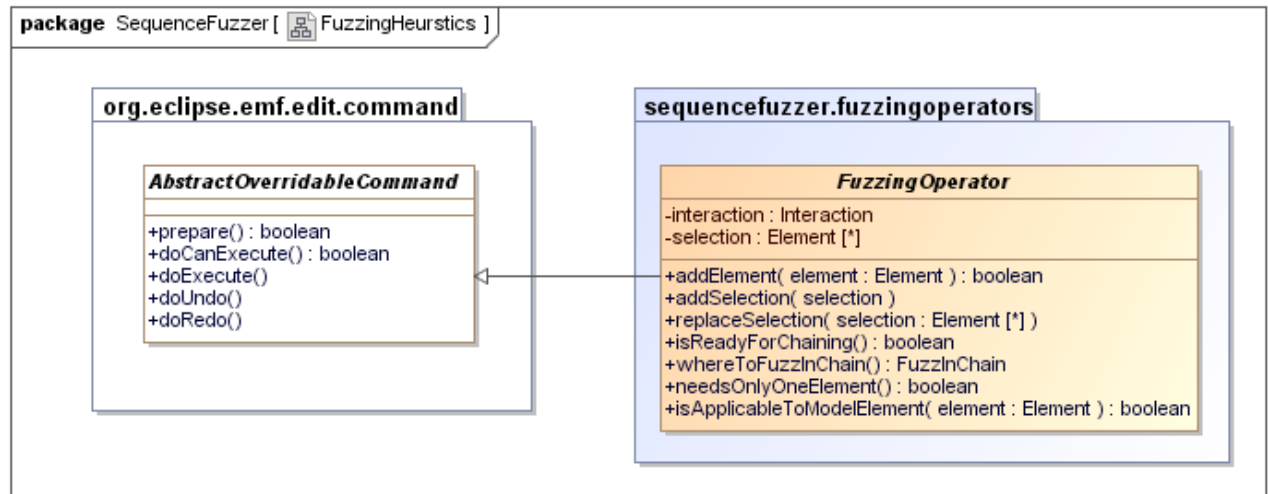


Figure 18: Eclipse Command Framework Class AbstractOverridableCommand and its Subclass FuzzingOperator


In order to create a behavioural fuzz test case, one or more fuzzing operators are applied to one interaction. When doing so, each fuzzing operator adds a comment to the fuzzed interaction that describes which kind of modification was performed and which model elements were affected. Finally, fuzzing strategies are needed for the task of selection appropriate combinations of fuzzing operators.

5.3.3 Fuzzing Strategies

The main task of fuzzing strategies is to determine how test cases are generated. This step is twofold: In the first step, the interactions and the enclosing model elements that shall be fuzzed have to be determined. For this purpose, there are two classes:

- The class *Selector* searches the interactions of a model for their model elements. It acts in concert with the fuzzing operators and asks each fuzzing operator whether a found model element, e.g. a message, can be fuzzed by that fuzzing operator. For each fuzzing operator, the model elements, the fuzzing operator can be applied to, are stored in a map. In order to traverse an interaction, the selector performs a depth-first search.
- The class *SmartSelector* is a subclass of the class *Selector* and extends the search for model elements by interactions referenced through an *InteractionUse* UML model element. It also respect which messages are relevant and which are not within a *ConsiderIgnoreFragments* combined fragment. Additionally, it ignores all model elements within the combined fragment *Negative* because they already constitute an invalid sequence.

In the second step, the fuzzing operators and its selected model elements are combined in order to build up the test cases. The combination of fuzzing operators is currently complete meaning that all combinatorial possibilities of fuzzing operators with a certain length are generated. One combination of fuzzing operators is represented by an instance of the class *ChainOfFuzzingOperators*. It stores a list of fuzzing operators and applies them to an interaction by creating a copy and “translating” the selection of the fuzzing operators to the copy of the interaction. Finally, it consecutively executes each fuzzing operator resulting in one test case per chain.

	<p>Initial Security Testing Tools</p> <p>Deliverable ID: D3.WP3</p>	Page : 34 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

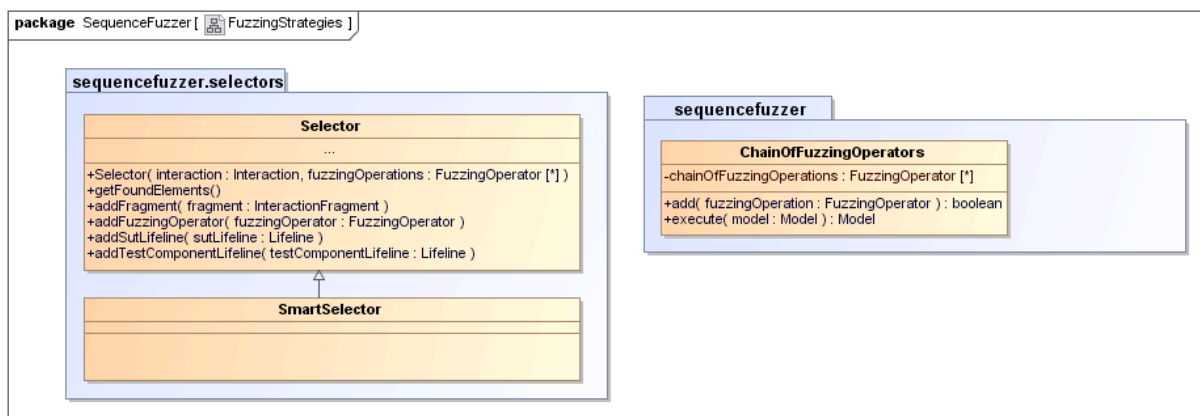



Figure 19: The Classes Implementing Fuzzing Strategies

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 35 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

6. SECURITY MONITORING LIBRARIES (MONTIMAGE)

Montimage monitoring tool (MMT) is an event-based monitoring solution that allows analysing network traffic according to a set of security properties referred to as MMT-Security properties. The main objective of these properties is to formally specify security goals and/or attack behaviours related to the application or protocol that is being monitored.

MMT can be delivered and installed as (1) a standalone tool that allows the analysis of live or pre-recorded structured data or as (2) a set of two libraries for integration into third party probes.

1. **MMT-Extract Library:** is a C library designed to extract data attributes from network packets, server logs and from structured events in general, in order to make them available for analysis. For this purpose, inspection techniques are used. MMT-Extract has a plugin architecture for the addition of new protocols and the parsing of proprietary structured data. In the rest of this document the terms: “packet”, “message” and “event” are used interchangeably.
2. **MMT-Security Library:** is also a C library designed to analyse the validity/satisfaction of a set of MMT-Security properties on a live or pre-recorded trace. This library uses MMT-Extract library to extract relevant attributes from the analysed events and makes it possible to deduce a verdict according to each MMT-Security property. The verdict will be: respected or violated if MMT-Security property denotes a security rule that the application or protocol being monitored has to respect; or, detected or not if the MMT-Security property denotes an attack or a vulnerability that the application or protocol has to avoid.

Providing MMT tool as a set of two libraries is very useful for performing online passive testing as in the context of the Thales case study related to radio protocols [25] as well as in the other case studies.

6.1 MMT-EXTRACT LIBRARY

MMT-Extract is the core packet processing module. It is a C library that analyses network traffic using Deep Packet/Flow Inspection (DPI/DFI) techniques in order to extract hundreds of network and application based events, including: protocols field values, network and application QoS parameters and KPIs. MMT-Extract has a plugin architecture for the addition of new protocols and a public API for integration with third party probes.

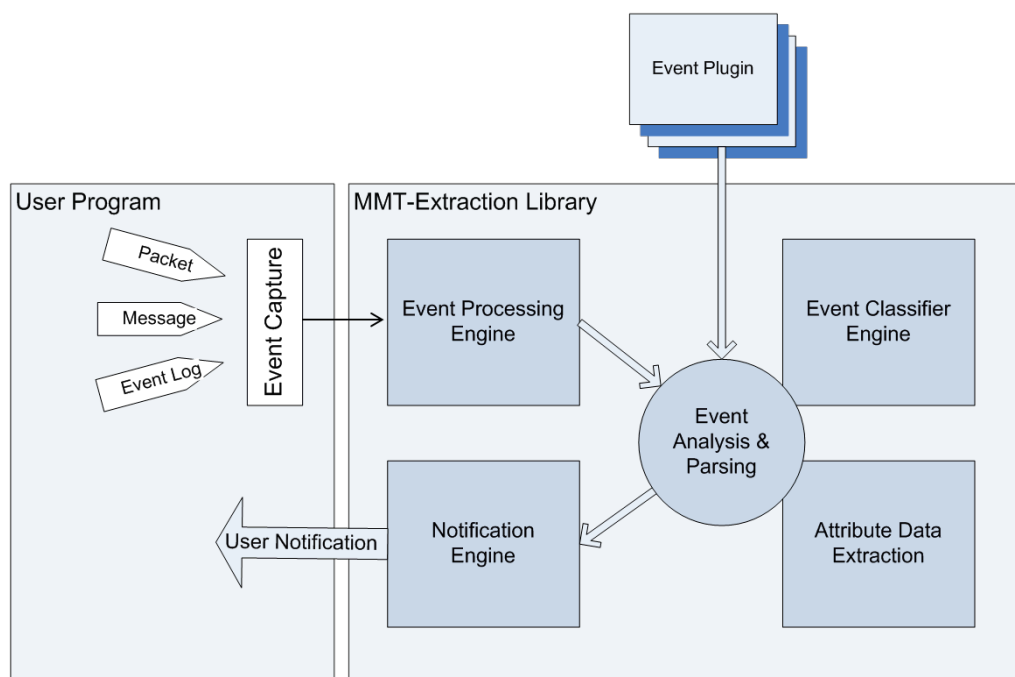


Figure 20: MMT-Extraction Library

6.1.1 Installing MMT-Extract

In order to install MMT-Extract, download and uncompress the MMT-Extract package. To be able to do this please send a request to contact@montimage.com. The package contains the extraction library (.dll for Windows, .so for Linux) and a folder with the required header files specifying the library's API.

6.1.2 Using MMT-Extract in a development project

In order to use the MMT-Extract library in your development project, you must perform the following:

- Include the header file "mmt_core.h": This file is the only one you need to include.
- Create a folder called "plugins" in the directory where the executable is located. If you have any MMT-Extract plugins, you MUST copy them into this "plugins" folder.


6.1.3 Extraction API description

The MMT-Extract API is specified in the header files provided with the download package. In the following we will shortly describe the content of the different header files (for further details, please refer to the documentation included in the package):

- mmt_core.h: this is where the core extraction API functions are defined.
- data_defs.h: this is where the data related API functions are defined. In addition, the different data structures that might be used in an integration project are defined here.
- types_defs.h: this is where the new data types are defined.
- extraction_lib.h: this file contains the generic extraction functions.
- plugin_defs.h: this file contains the definitions that can be used for the plugin development.

6.1.3.1 Initialization

In order to use the MMT-Extract library, you MUST initialize it; this is done using the following function signature:

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 37 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

```
int init_extraction();
```

The initialization returns a positive value on success and zero on failure. It is good practice to always check the return value of “init_extract”.

```
int close_extraction();
```

This function will close the extraction and free any previously allocated memory.

6.1.3.2 Message processing

MMT-Extract can process network packets in the de-facto “pcap” format, raw messages, log messages, etc.

The following function hands a message/packet to the core engine of MMT-Extract:

```
int packet_process(struct pkthdr *header, u_char * packet);
```

This function should be called for every packet/message/event to process. The **header** parameter of the function is a pointer to the meta-data of the message that include the message arrival time, the message length, etc. The **packet** parameter of the function is a pointer to the message data. The “packet_process” function will return a positive value on successful processing. Zero is returned if an internal error is encountered, although this should not happen.

```
void setDataLinkType(int dltype);
```

This function sets the link type to indicate the nature of the lower layer protocol. By default, the library is configured to process network packets in the “pcap” format with Ethernet as the link layer.

6.1.3.3 Registering extraction attributes

The MMT-Extract library allows registering attributes for extraction. This allows limiting the extraction overhead by only recuperating the data that is needed. Attributes are fields in network packets, specific data in event logs, etc.

```
void register_extraction_attribute(protocol_id, attribute_id);
void register_extraction_attribute_by_name(proto_name, attribute_name);
```

Either one of these two functions allows registering an attribute for extraction. An attribute is identified by the protocol and attribute id or names. If the registration succeeds, a positive value will be returned; zero is returned on failure.


```
int is_registered_attribute(protocol_id, attribute_id);
```

Allows verifying if an attribute, identified by its protocol and attribute identifiers, is already registered. It will return a positive value if the attribute is found registered.

```
int unregister_extraction_attribute(protocol_id, attribute_id);
int unregister_extraction_attribute_by_name(proto_name, attribute_name);
```

Either one of these two functions allow unregistering an already registered attribute. If the unregistration succeeds, a positive value will be returned.

```
void * get_attribute_extracted_data(protocol_id, attribute_id);
void * get_attribute_extracted_data_by_name(char * protocol_name,
                                           char * attribute_name)
```

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 38 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

Either one of these functions will return a pointer to the data corresponding to the provided attribute if it was detected in the last provided event, or NULL otherwise.

6.1.3.4 Registering callbacks

The MMT-Extract library allows registering callback functions. A registered callback function will be called following the extraction of attributes whenever a packet/message/event log/etc. is processed.

```
int register_packet_handler(int packet_handler_id,
                           generic_packet_handler_callback function,
                           u_char * args);
```

This function allows registering a packet handler that is a callback that will be called each time a packet is received. If needed, the user can provide a pointer to an argument that will be passed to the callback function when it is called. The callback function is associated with an identifier that should be unique, i.e., two callback functions cannot have the same identifier. This function will return a positive value upon success.

To verify that a packet handler callback function is registered with a given identifier, you can use the following function:

```
int is_registered_packet_handler(int packet_handler_id);
```

This function returns a positive value if a registered callback function is found for the provided identifier.

In order to unregister an already registered packet handler you can use:

```
int unregister_packet_handler(int packet_handler_id);
```

In addition to packet handlers, it is possible to register a callback function to be called when an attribute is detected. This can be done with the following function:

```
int register_attribute_handler(long protocol_id, long attribute_id,
                              attribute_handler_function handler_fct,
                              void * handler_condition, void * user_args);
```

This function allows registering an attribute handler that is a callback that will be called when the attribute defined by the given protocol and attribute ids. If needed, the user can provide a pointer to an argument that will be passed to the callback function when it is called. In the current version the parameter defined by handler_condition is not implemented and should be set to NULL. This function will return a positive value upon success.


```
int register_attribute_handler_by_name(char * protocol_name,
                                       char * attribute_name,
                                       attribute_handler_function handler_fct,
                                       void * handler_condition, void * user_args);
```

This function registers an attribute handlers given by its protocol and attribute names.

To verify that an attribute handler is registered, you can use the following function:

```
int has_registered_attribute_handler(long protocol_id, long attribute_id);
```

This function returns a positive value if a handler is already registered with the attribute defined by its protocol and attribute identifiers.

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 39 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

In order to unregister an already registered attribute handler you can use:

```
int unregister_attribute_handler(long protocol_id, long attribute_id);
int unregister_attribute_handler_by_name(char * protocol_name,
                                         char * attribute_name);
```

6.1.3.5 Data functions

In addition to the core functions presented so far, MMT-Extract provides a number of utility functions to assist the user of the library.

```
const char * get_protocol_name_by_id(long protocol_id);
```

Returns the protocol name, given its numeric identifier; NULL is returned if the given identifier does not correspond to any configured protocol.

```
long get_protocol_id_by_name(const char *protocol_name);
```

Returns the identifier of the protocol, given its name; "0" is returned if the given name does not correspond to any configured protocol.

```
int is_protocol_attribute(int protocol_id, int attribute_id);
```

Indicates if the attribute exists for the given protocol and attribute ids.

```
const char * get_attribute_name_by_protocol_and_attribute_ids(long protocol_id,
                                                             long attribute_id);
```

Returns the name of the attribute corresponding to the given protocol and attribute identifiers; NULL is returned if there is no attribute corresponding to the given identifiers.

```
long get_attribute_id_by_protocol_and_attribute_names(const char *protocol_name,
                                                     const char* attribute_name);
```

Returns the identifier of the attribute corresponding to the given protocol and attribute names; "0" is returned if there is no attribute corresponding to the given names.

```
long get_attribute_id_by_protocol_id_and_attribute_name(long protocol_id,
                                                       const char *attribute_name);
```

Returns the identifier of the attribute corresponding to the given protocol id and attribute name; "0" is returned if there is no attribute corresponding to the given parameters.

```
long get_attribute_data_type(long protocol_id, long attribute_id);
```

Returns the identifier of the data type of the attribute corresponding to the given protocol and attribute identifiers; "0" is returned if there is no attribute corresponding to the given parameters.

```
int get_data_size_by_proto_and_field_ids( int protocol_id, int attribute_id);
```


Returns the data size of the attribute corresponding to the given protocol and attribute identifiers; "0" is returned if there is no attribute corresponding to the given parameters.

```
int get_data_size_by_data_type(int data_type);
```

Returns the data size of the given data type; "0" is returned if the data type is unknown.

```
int get_field_position_by_protocol_and_field_ids(int protocol_id,
                                                 int attribute_id);
```

Returns the position in the message of the attribute corresponding to the given protocol and attribute identifiers; for attributes where the position depends on the content of the message, "POSITION_NOT_KNOWN" (value -1) will be returned. The position is defined as the byte offset from the beginning of the packet.

	<p align="center">Initial Security Testing Tools</p> <p align="center">Deliverable ID: D3.WP3</p>	Page : 40 of 81
		Version: 1.1
		Date : 29.6.2012
		Status : Final Confid : Public

Several extraction example codes are provided in Appendix A.

6.2 MMT-SECURITY LIBRARY

MMT-Security library is a security analysis engine based on MMT-Security properties. MMT-Security analyses and correlates network and application events to detect operational and security incidents. For each occurrence of a security property, MMT-Security allows detecting whether a security property was respected or violated; or, an abnormal behaviour (e.g., attack) was detected or not.

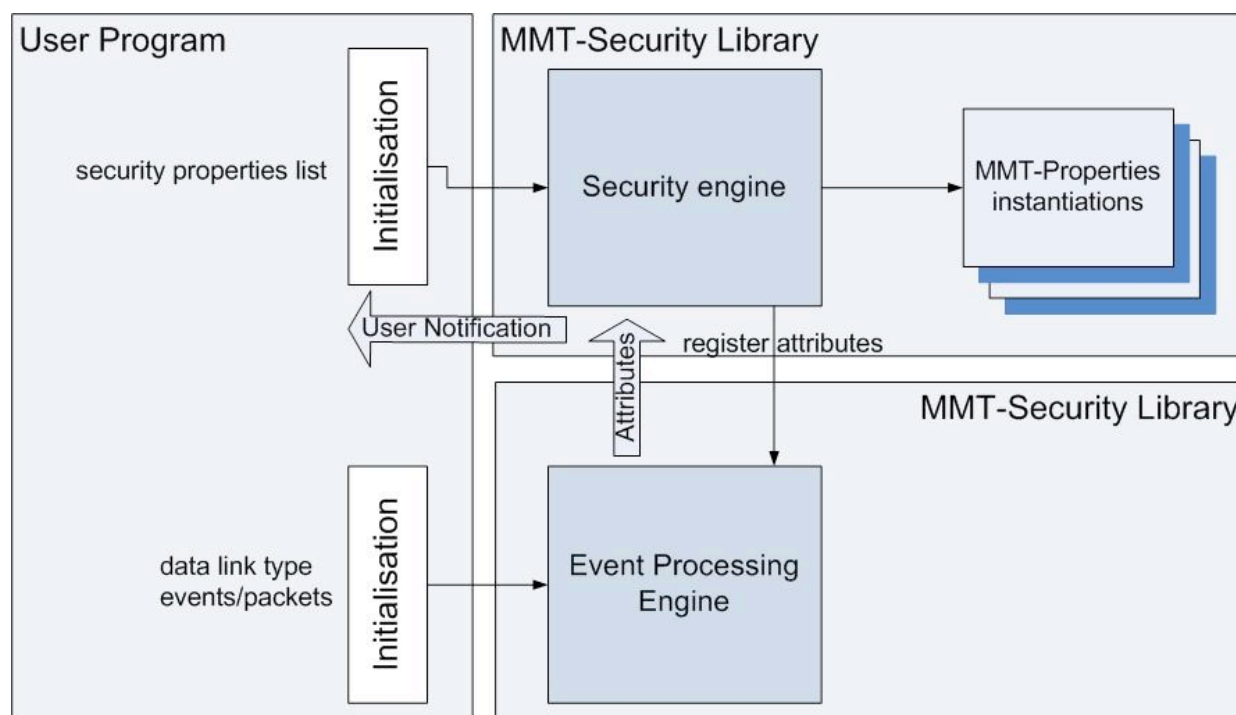


Figure 21. MMT-Security Library

6.2.1 Installing MMT-Security

You only need to download and uncompress the MMT-Security package (send a request to contact@montimage.com). The package contains the extraction library (.dll for Windows, .so for Linux) and a folder with the required header files specifying the library API.

6.2.2 Using MMT-Security in a development project


In order to use the MMT-Security library in your development, no specific action is needed.

6.2.3 Security analysis API description

6.2.3.1 Initialization

In order to use the MMT-Security library, you MUST initialize it; this is done using the following command:

```
void init_sec_lib(char * property_file,
    short option_satisfied, short option_not_satisfied,
    result_callback todo_when_property_is_satisfied_or_not);
```


	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 41 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

The initialisation function defines the XML properties file path, options and a pointer to the callback function that will be executed by the MMT-Extraction library each time an event occurs (e.g., a packet is captured).. The following arguments are used:

- Argument 1: properties file.
- Argument 2: if 1 then results will contain details on properties satisfied.
- Argument 3: if 1 then results will contain details on properties not satisfied.
- Argument 4: name of function that will be executed each time a property is satisfied or not.

In the future, more functions are planned to manage security properties (on-the-fly adding and removal of security properties) and to get the detailed results.

A security analysis example code is provided in Appendix B.

7. MODEL-BASED BEHAVIORAL SECURITY TESTING TOOL DEVELOPMENT (SMARTESTING)

7.1 OVERALL PROCESS

In deliverables D2.WP2 and D2.WP3, we have presented the concepts of automated generation of security tests based on behavioural models and test purposes. In this deliverable, we detail this approach in terms of process, actors & roles and used artefacts. In D3.WP2, a first method named *Smartesting Model-Based Security Testing from behavioral models and security-oriented Test Purposes* is presented.

Model-based security testing from behavioral models and test purposes is an extension of functional model-based testing (MBT):

- The model for test generation captures the expected behavior of the system under test (SUT). This model is dedicated for automated generation of security tests, and generally formalizes the security functions of the SUT but also the possible stimuli of an attacker as well as the expected answer of the SUT.
- The test purposes are test selection criteria that define the way to generate tests from the test generation model.

The main difference with “classical” functional MBT (see[26] for a detailed introduction of functional MBT) is firstly that the behavioral model may represent stimulations that are not defined in the specification of the SUT. For example, if a security test engineer wants to generate SQL injection test, he or she would represent SQL injection operation inside the test generation model. Secondly, the model-based security testing differs in the way test cases are selected from the behavioral models. In functional MBT, the way is often based on a structural coverage of the model, mixing the coverage of expected behavior and logical test data. The tests are in general “positive”, aiming to test all the nominal cases and few (or a several) errors cases. In security testing, the goal is really to systematically trying to break (or to bypass) the security functions of the SUT. This search for breaking software security barrier requires to systematically trying possible breakers in a large set of application contexts. The test purpose language goal is to support such test selection criteria.

The following figure presents the Smartesting process and tool for model-based security testing.

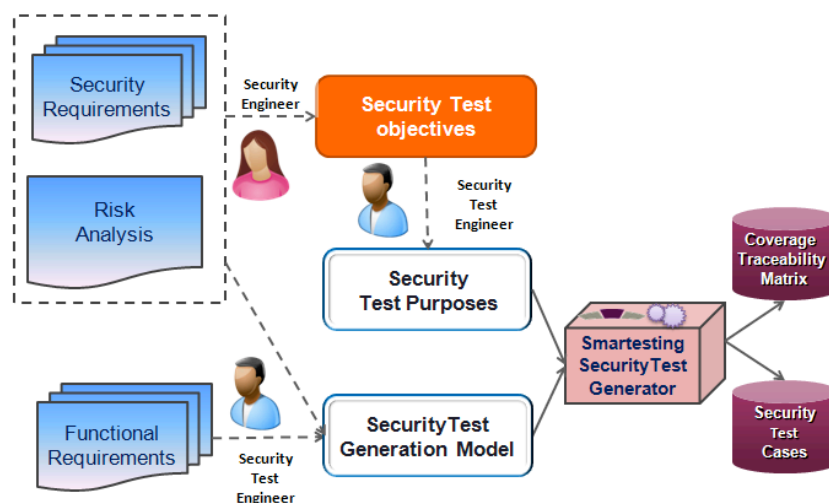



Figure 22: Smartesting process and tool for model-based security testing

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 43 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

The next pages will describe the functional and security requirements of a sample application with some identified risks. They will be used in the tool description as an example to illustrate the manipulations to be done to generate the test cases.

7.2 SAMPLE APPLICATION

In order to help understanding the following concepts, we define here a sample application that will have to be tested.

Example: Authentication management part of a simple web application for buying products.

7.2.1 Functional and Security requirements

The user authentication is done thanks to a login/password couple. We would like to test the correct behavior of the “login” and “logout” operations. Those actions should respect the following functional requirements:

- loginKo1: the login/password couple must match those of a known user, otherwise a “authentication failed” message should be displayed.
- loginKo2: the login or password field cannot be empty, otherwise a “empty required field” message should be displayed.
- loginOk: the user is well authenticated on the application, a “welcome” message should be displayed.
- logout: only an authenticated user can do this action. A “bye” message should be displayed.

The user can add some products to his cart, remove some products from it, consult his cart. All those operations can only be done while the user is connected. We would like to test the “add product to basket”, “remove product from basket” and “view basket” operations.

- addProductKo: the user is not yet connected, the product is not added to the cart. A “Please log first” message should be displayed.
- addProductOk: the product is added to the cart
- removeProduct: only an authenticated user can do that from the his “cart” page.
- displayCartKo: the user is not yet connected. A “Please log first” message should be displayed.
- displayCartOk: the “cart” page is displayed.
- backToShop: only available on the “cart” page. The shop page should be displayed.

No user is connected at the application initial state. At any time, no more than one user can be connected.

7.2.2 Identified risks


A user should not be able to view other user's cart. After logout, the cart should not be accessible, and no operation (adding or removing product) should be done on it.

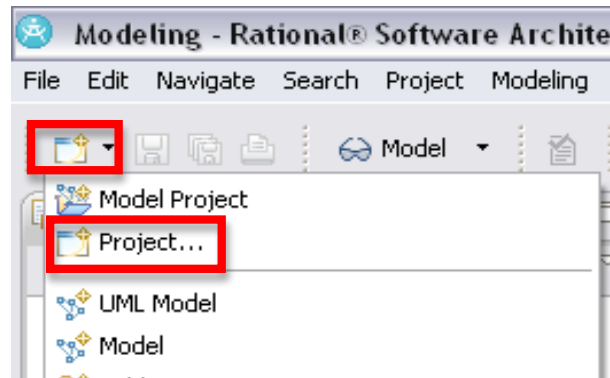
7.3 TEST GENERATION MODEL

As described before, one of the entry points of the solutions consist in creating the test generation model.

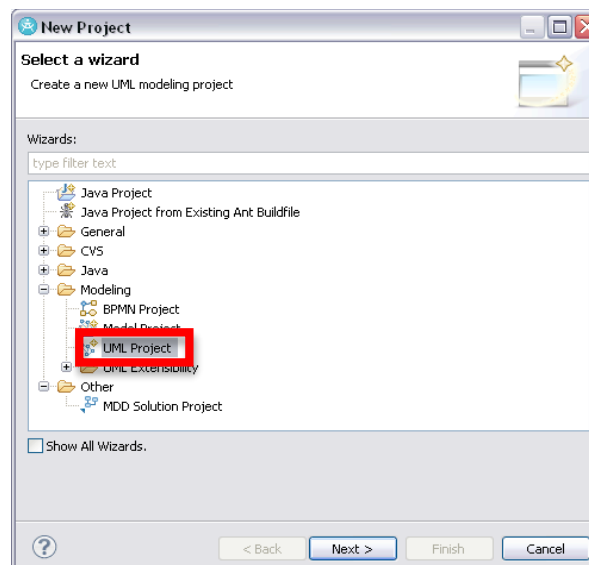
7.3.1 Creating a new UML Project

To create a project, launch IBM Rational Software Architect and select the *new*→*Project...* menu

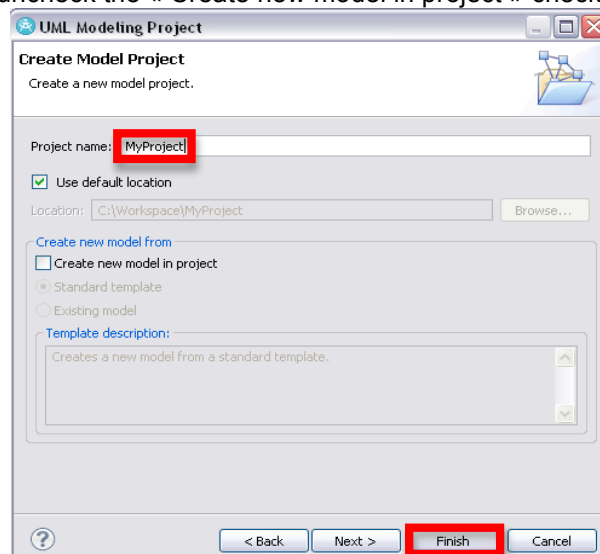
	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 44 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public




In the *New Project* window, select « UML Project » and click on *Next*.



Give a name to the project, uncheck the « Create new model in project » checkbox and click on *Finish*.

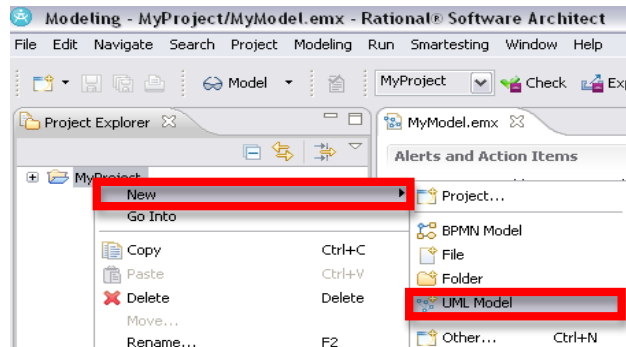


	<p align="center">Initial Security Testing Tools</p> <p align="center">Deliverable ID: D3.WP3</p>	Page : 45 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

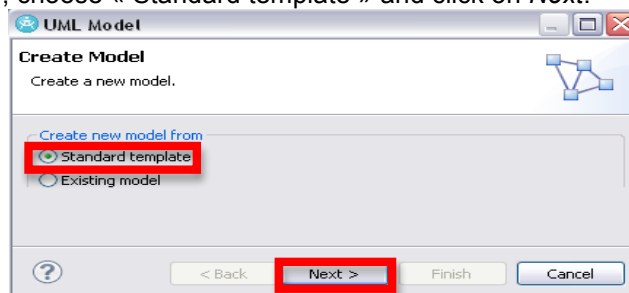
The project is now created.

7.3.2 Creating the UML Model in the project

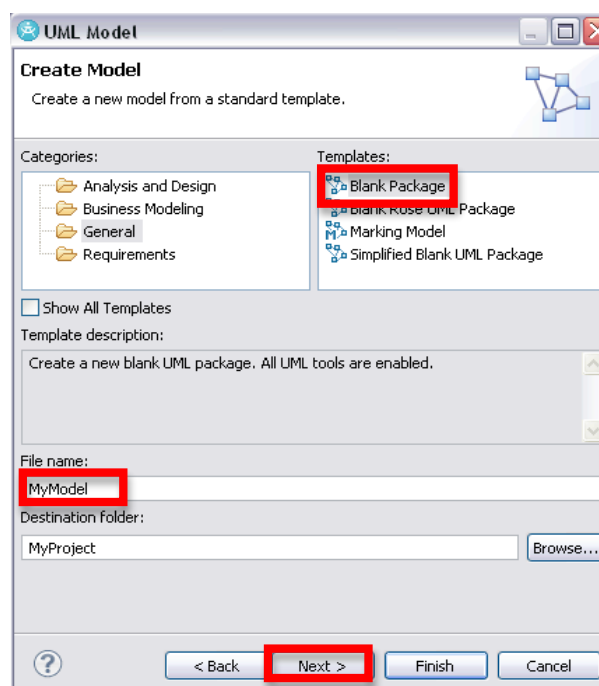
Right click on your project and select *New→UML Model*.




On the *UML Model* window, choose « Standard template » and click on *Next*.

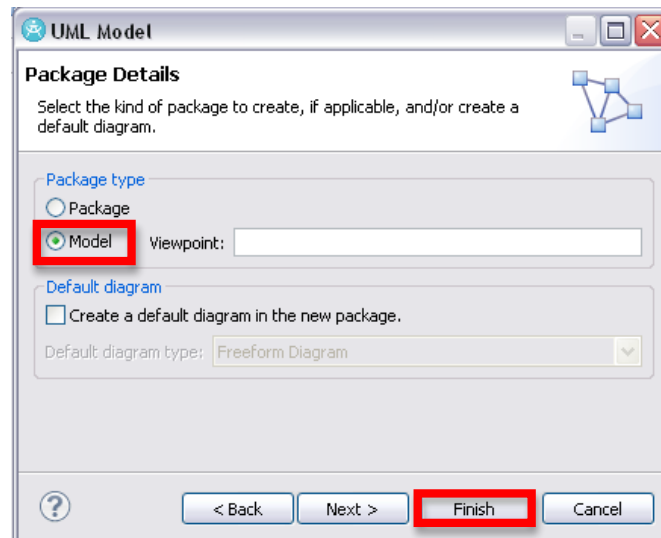


Then choose *Blank Package*, give a name to your model and click on *Next*.



	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 46 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

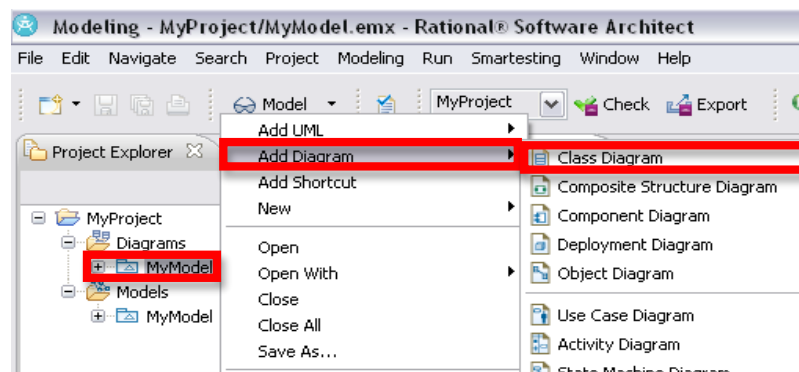
Select the *Model* radio button, and then click on *Finish*.



Your model is now created. You can now add diagrams to your model.

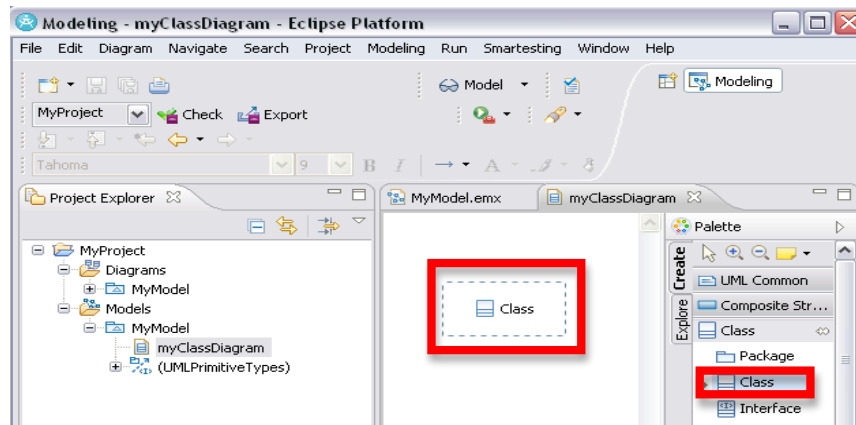
7.3.3 Creating a Class Diagram in the model

Right click on your model, then choose *Add Diagram* → *Class Diagram*. Choose a name for your class diagram.

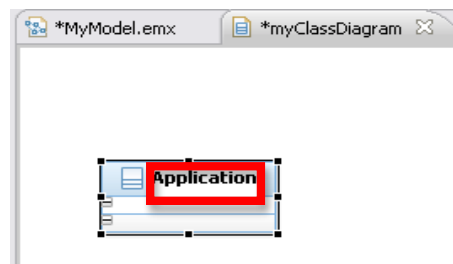


7.3.3.1 Class creation

Select the *Class* shape in the Palette, then click anywhere in the diagram to add the class.



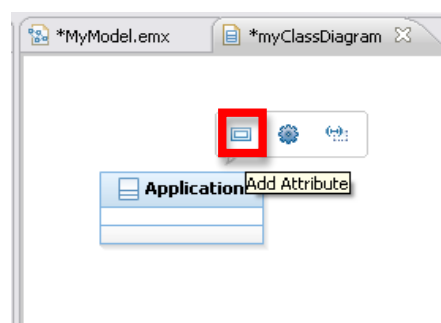
You can now choose a name for the class.




A first class is now created, you will be able to add attributes to this class.

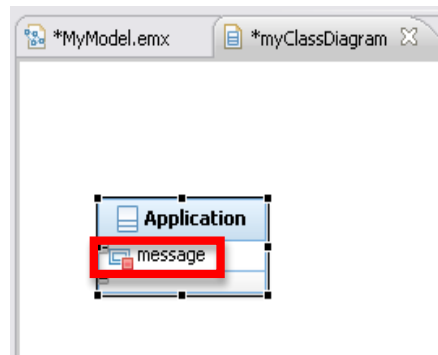
7.3.3.2 Attribute creation

Over the cursor over your class, a contextual action bar will enable you to add an attribute by clicking on the *Add Attribute* icon.



You can specify a name for the attribute.

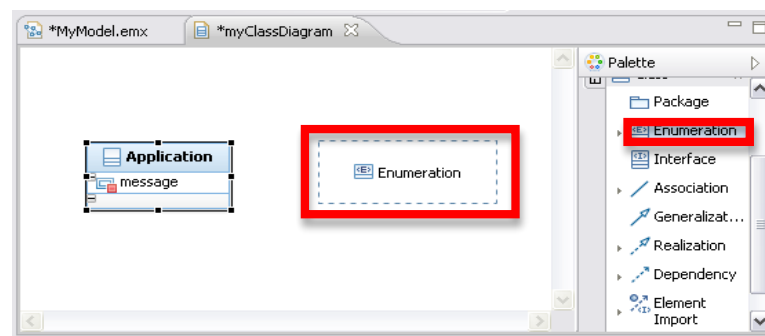
	<p align="center">Initial Security Testing Tools</p> <p align="center">Deliverable ID: D3.WP3</p>	Page : 48 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public



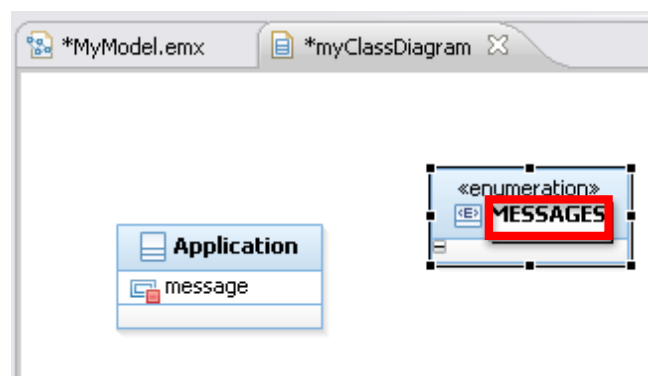
In order to give a type the the attribute, you can create an Enumeration that will represent a new type, containing the different possible values for the attribute.

7.3.3.3 Enumeration creation

Select the *Enumeration* shape in the Palette (after unfolding the class shape), then click anywhere in the diagram to add the enumeration.




You can now choose a name for the enumeration.

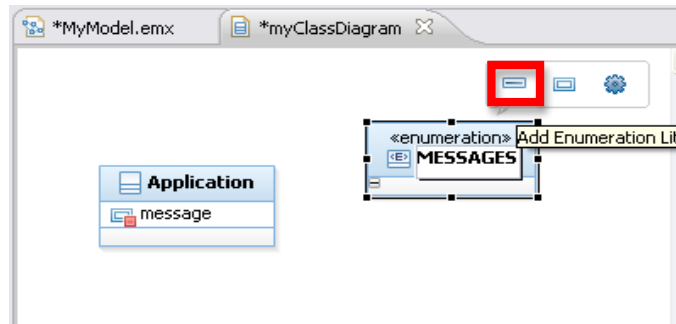


The enumeration will contain the different possible values for the represented type.

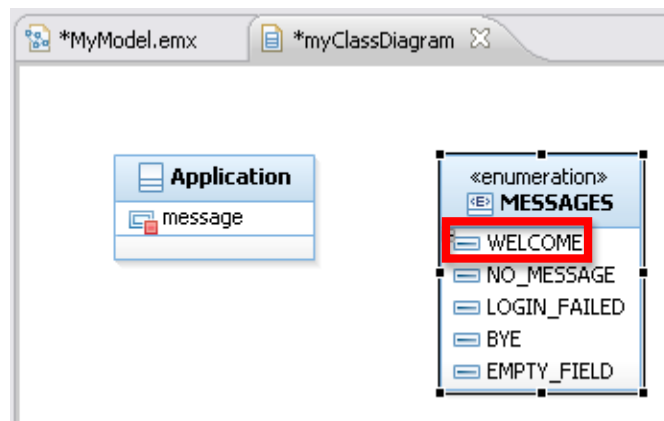
7.3.3.4 Enumeration literal creation

Like for the class attributes, over the cursor over your enumeration, a contextual action bar will enable you to add a literal by clicking on the *Add Enumeration Literal* icon.

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 49 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public



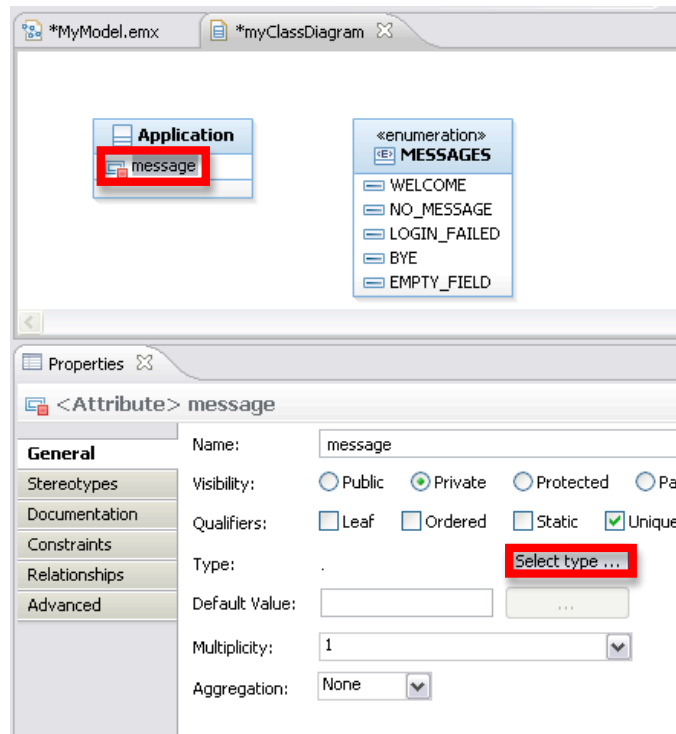
You can create an attribute for each of the values of the represented type.



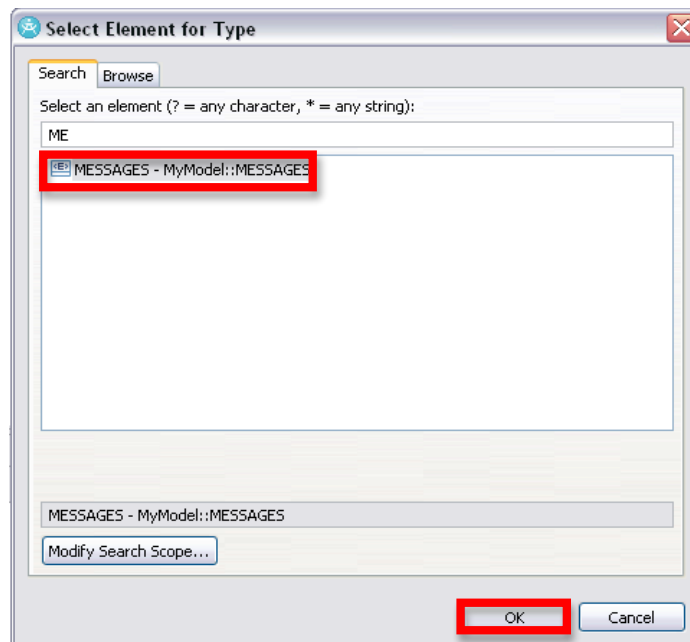
Your enumeration is now created, and can be used to specify an attribute type.

7.3.3.5 Attribute type specification

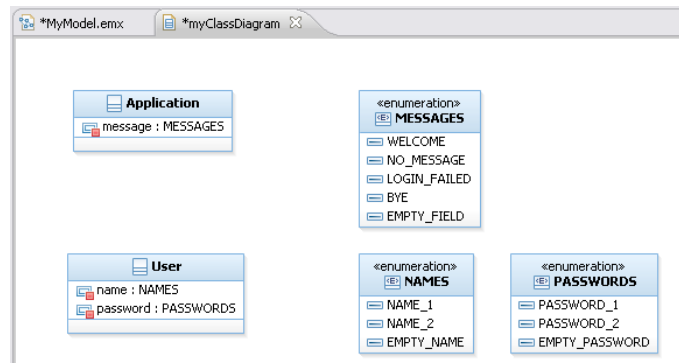
Select the attribute and go to the *General* part of the *Properties* tab. You can click on *Select type...* to choose the attribute type.



In the *Select Element for Type* window, you can search the Type you have created, and select it. Validate by clicking *Ok*.



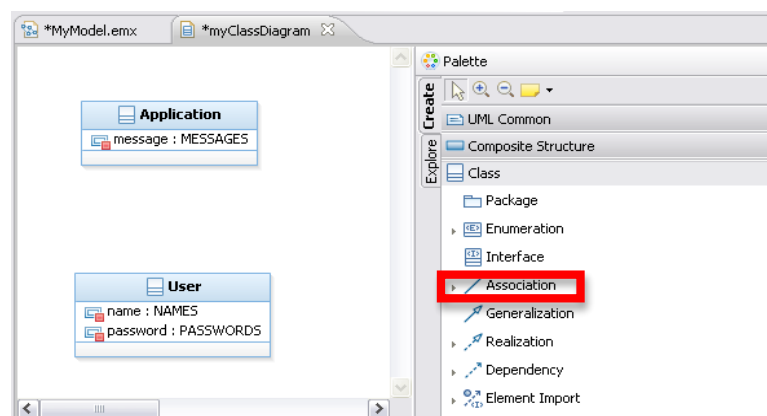
You have now created a first class, with and attribute. The attribute type has also been defined. You can continue by doing the same for a « user » who has a « name » and a « password ».



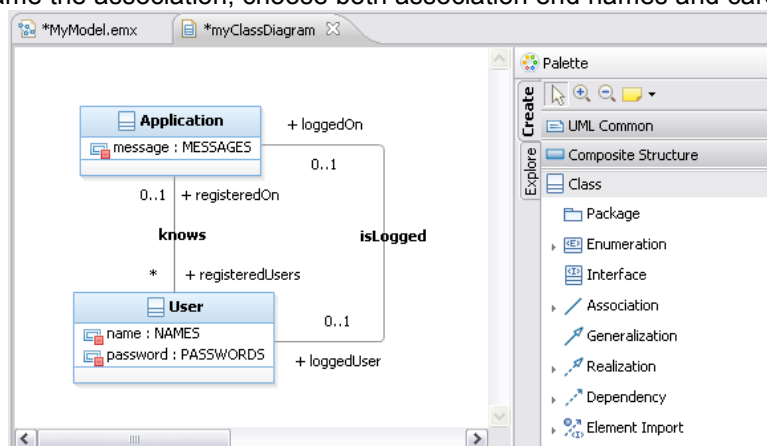
You can now define the relations between the classes by creating association links.

7.3.3.6 Association Link creation

To create an association link in the class diagram, select the *Association* shape in the Palette, then drag from one class to the other.



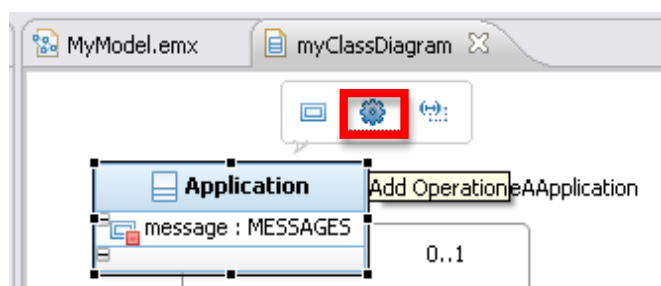
You will be able to name the association, choose both association end names and cardinalities.



The model structure is now defined. To represent the expected behaviors of the application, operations must be created.

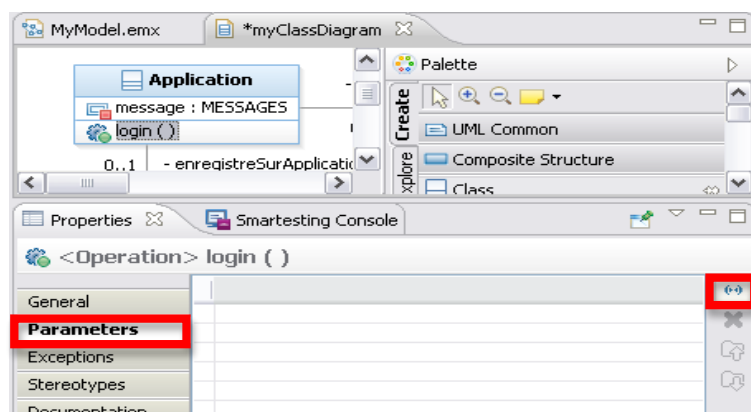
7.3.3.7 Operation creation

Over the cursor over your class, a contextual action bar will enable you to add an operation by clicking on the *Add Operation* icon. Name the operation as appropriated.



7.3.3.8 Operation parameters

After selecting the operation, the *parameters* part of the *Properties* pan enables to add parameters to the operation.




The parameter direction (In, Out, Return), name and type can then be chosen.

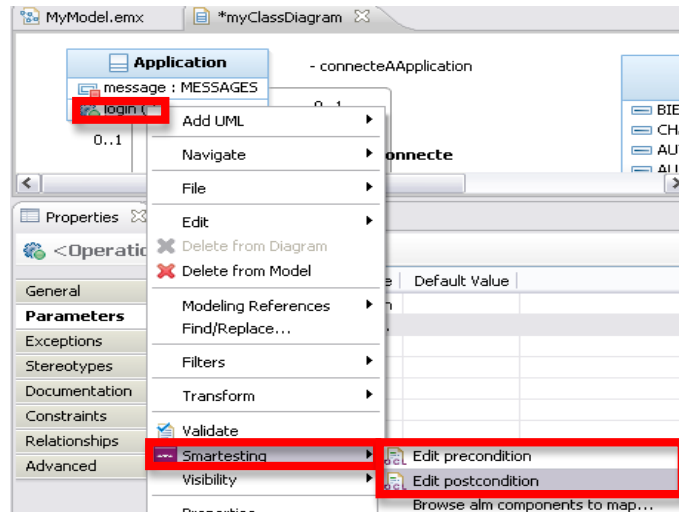
<Operation> login ()					
General	Direction	Name	Default Value	Type	
Parameters	In	IN_login		NOMS	
	In	IN_motDePasse		MOTS_DE_PASSE	

The expected behaviors of the operation can now be added to it with pre/post conditions.

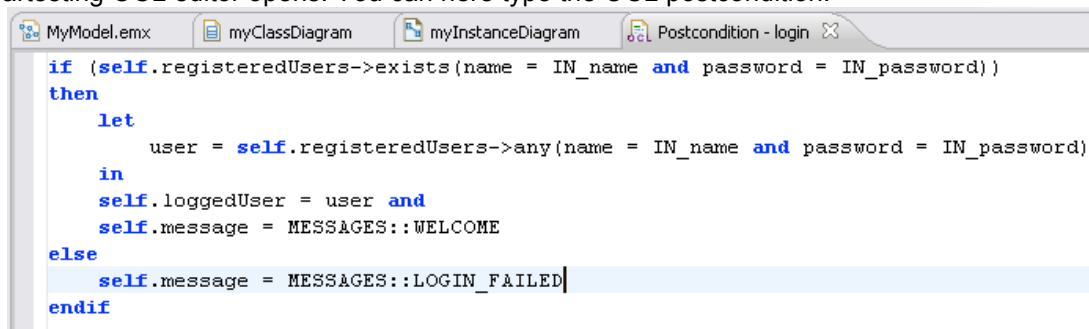
7.3.3.9 Operation Pre/Post conditions

Pre/Post conditions are defined with OCL code. To create a postcondition right click on the operation, then select *Smartesting* → *Edit postcondition*.

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 53 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public




The Smartesting OCL editor opens. You can here type the OCL postcondition.

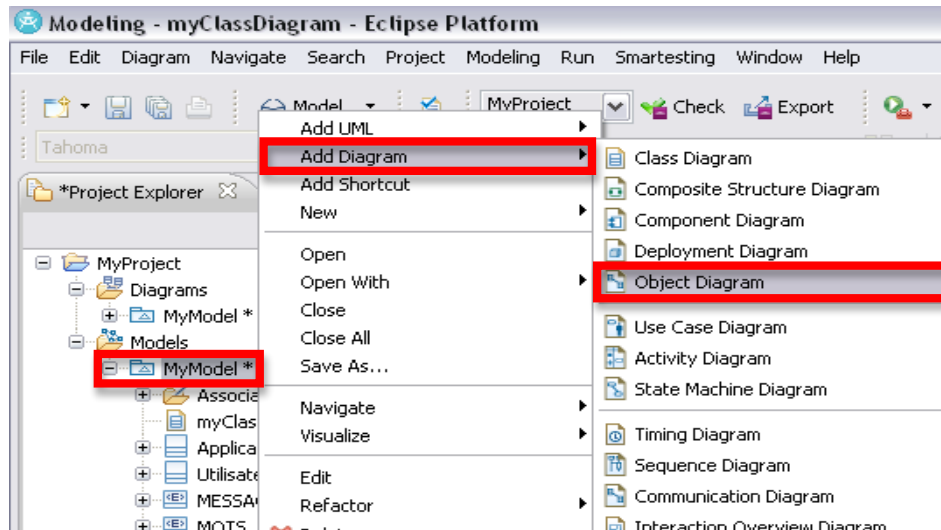


You have now defined the different classes of objects that can be manipulated by the application, with their relationships, and the expected behaviors of the application. Now to define the initial state of the test suite, an object diagram must be created.

7.3.4 Creating an Instance Diagram in the model

To create an object diagram, right click on the model, select *Add Diagram*→*Object Diagram*. You can name the object diagram as you wish.

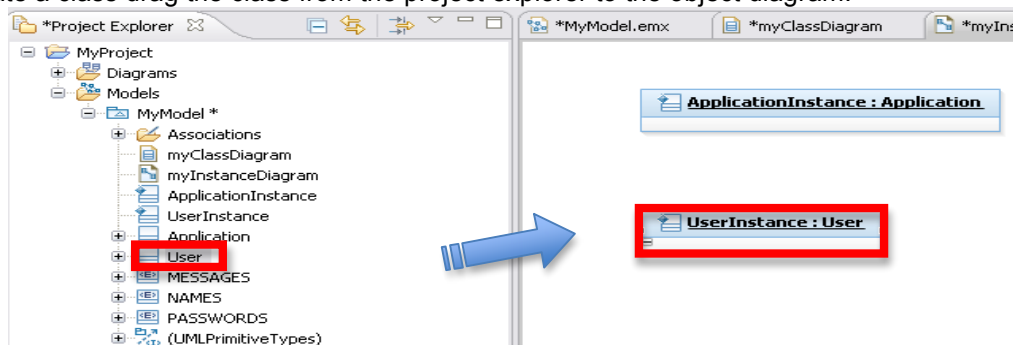
	<p align="center">Initial Security Testing Tools</p> <p align="center">Deliverable ID: D3.WP3</p>	Page : 54 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public



Class instances can now be added to this diagram.

7.3.4.1 Class instantiation

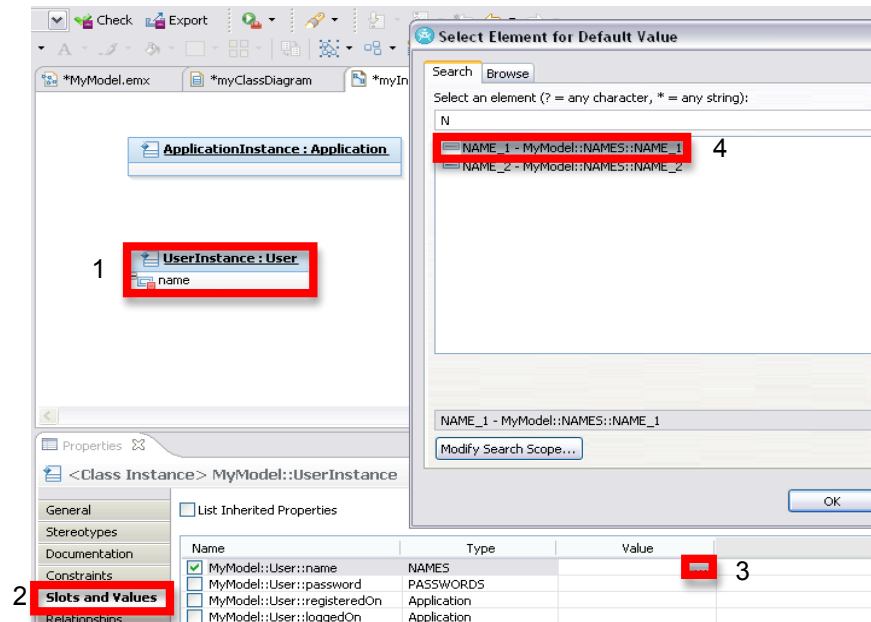
To instantiate a class drag the class from the project explorer to the object diagram.



For each class the attributes can be instantiated.

7.3.4.2 Attribute instantiation

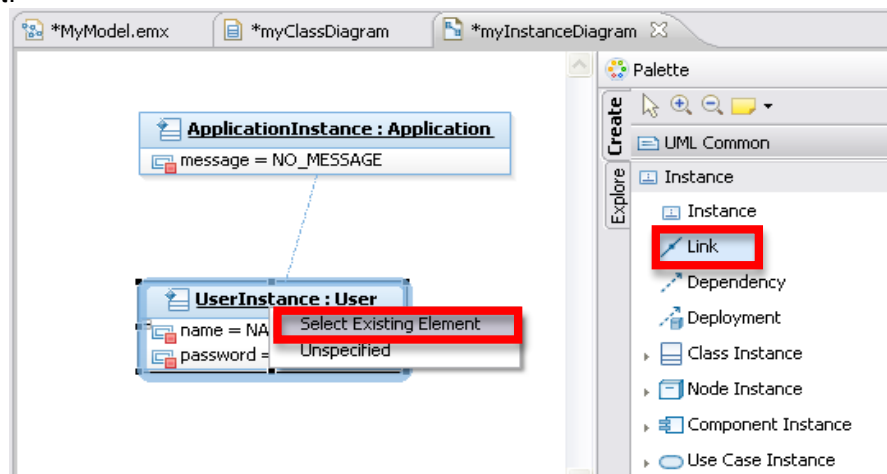
After selecting a class, the *Slots and Values* part of the *Properties* tab enable to valuate each class attribute.




As each class attribute initial value is chosen, let now instantiate the links between the classes.

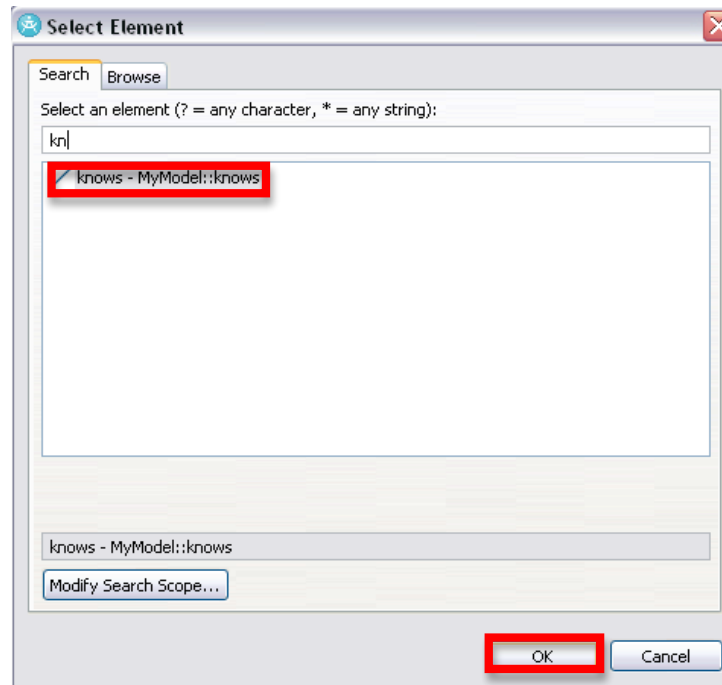
7.3.4.3 Association link instantiation

To create an association link instantiation select the *Link* shape in the Palette. As for the association link creation between classes, drag from one class instance to the other to create the link, then choose *Select Existing Element*.



Choose the association link you want to instantiate, and then confirm by clicking *Ok*.

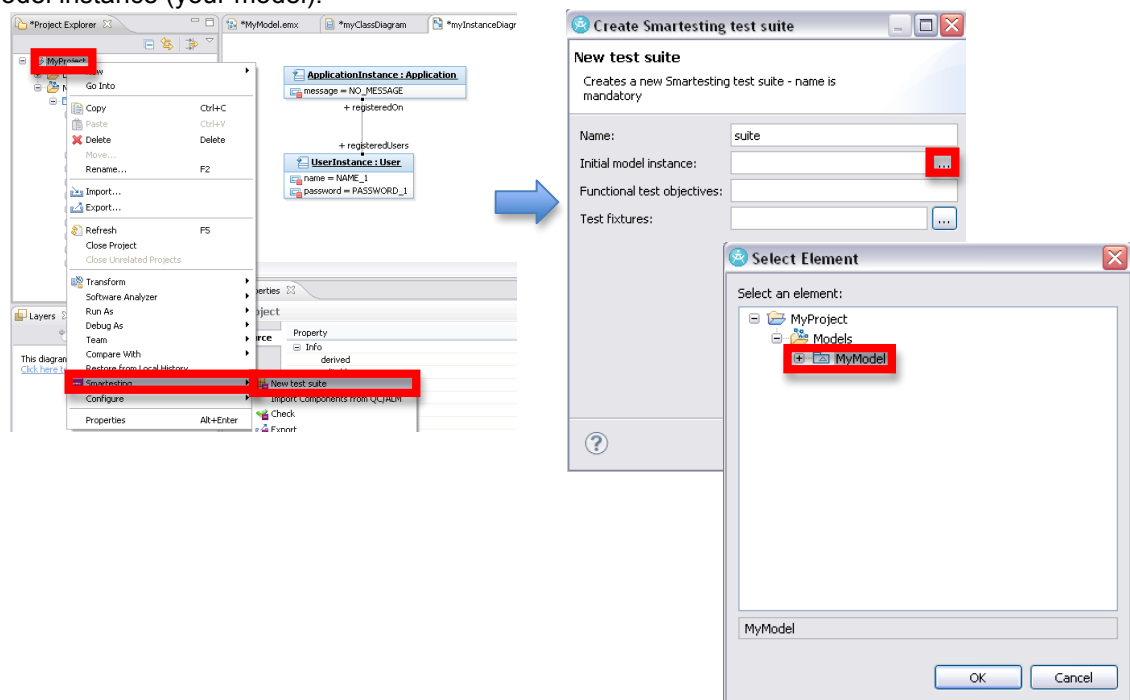
	<p align="center">Initial Security Testing Tools</p> <p align="center">Deliverable ID: D3.WP3</p>	Page : 56 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public




A first initial state represented by the object diagram is now created. The test suite can be now created.

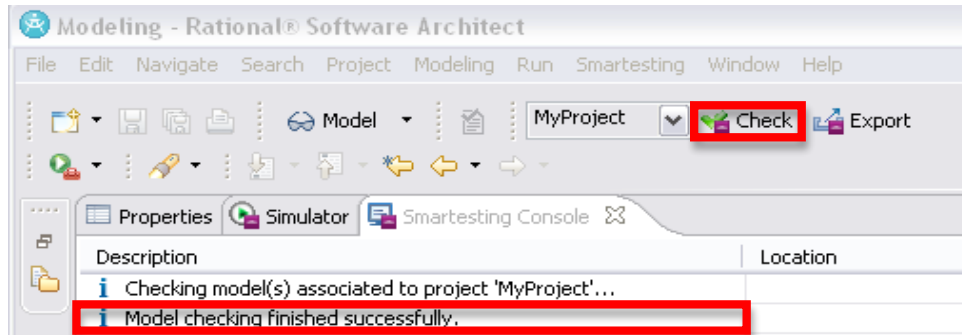
7.3.5 Creating a Test Suite

To create a test suite right click on the project, then select *Smartesting*→*New test suite*. Then choose an initial model instance (your model).



Now at any time you can click on the *Check* button to see if the model is correct.

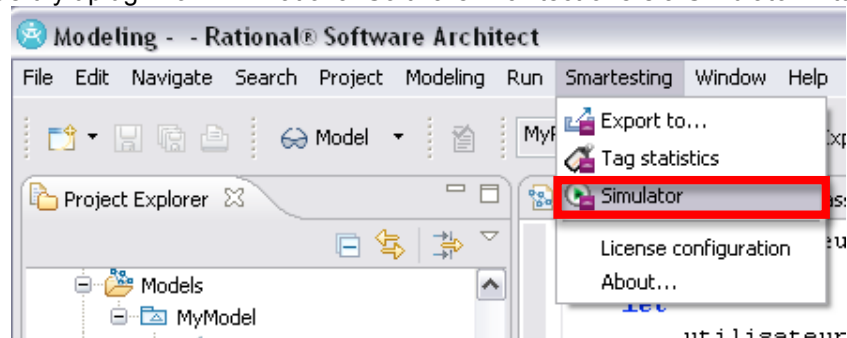
	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 57 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public



While a model is correct it can be used to create test sequences.

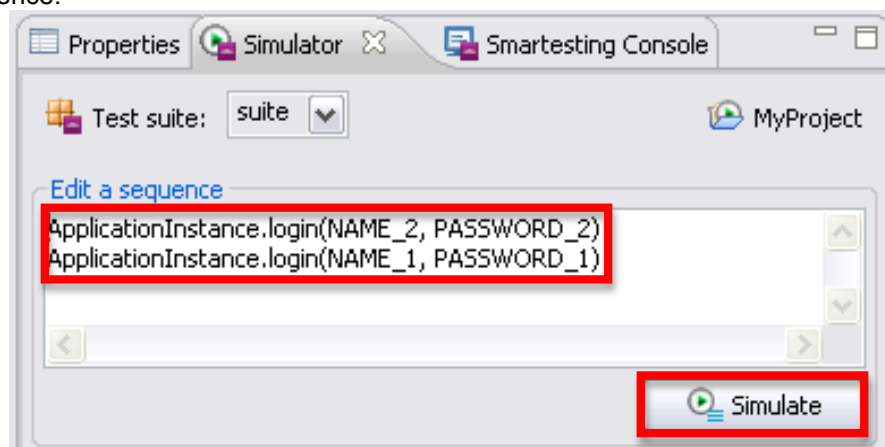
7.3.6 Using Smartesting Simulator

The Smartesting CertifyIt plugin for IBM Rational Software Architect offers a Simulator fixture.

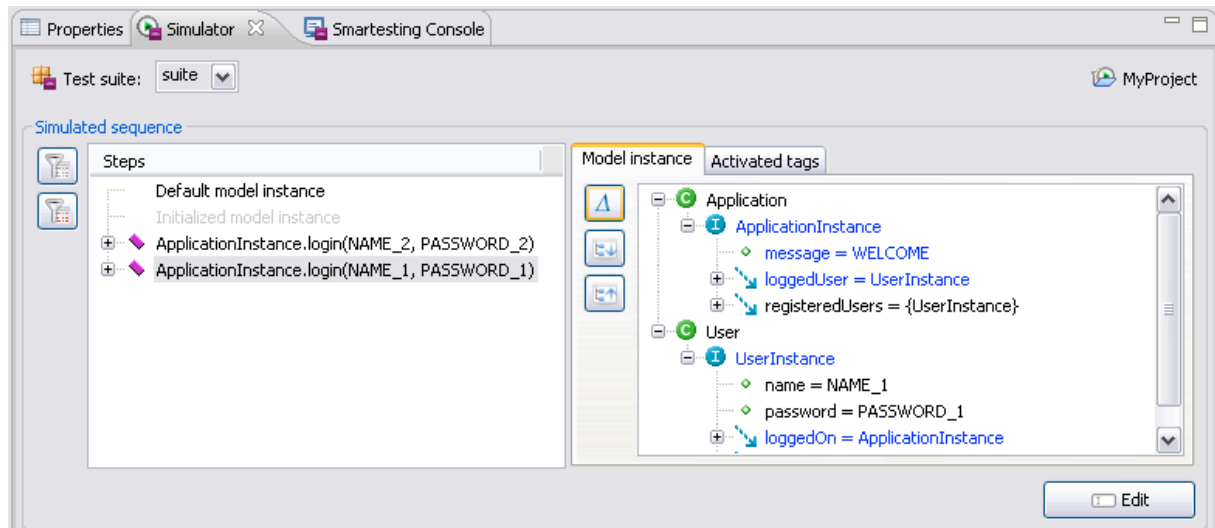


It will enable you to check the correct behavior of the operations you have modeled in regards of the specification.

You can type a sequence of operations in the simulator and then click on *Simulate* to see the result of the simulation sequence.



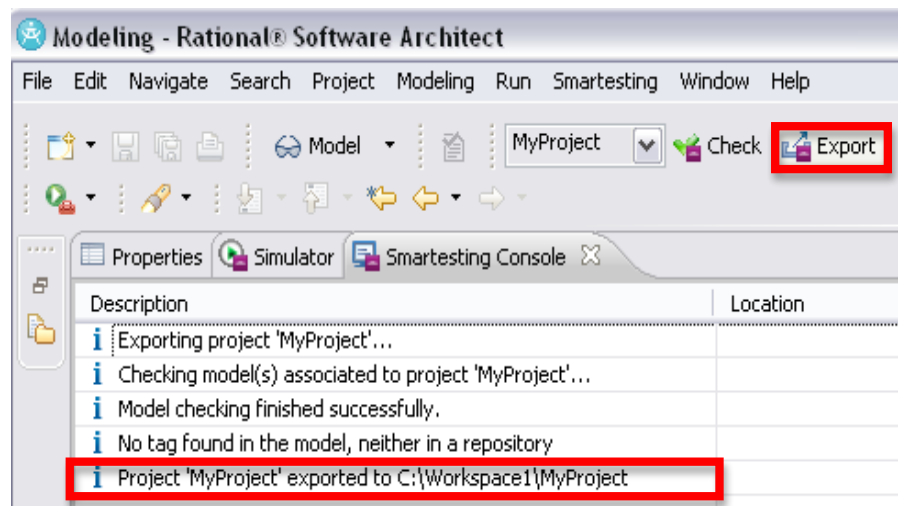
Here is the result of the sequence simulation on the model.




If the operations expected behaviors are correct during the simulation, the model can be exported to the Smartesting CertifyIt generation tool.

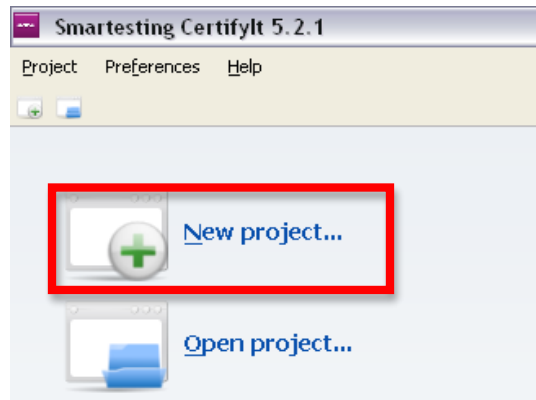
7.3.7 Generating tests with Smartesting CertifyIt

The *Export* button create a « .smtmodel » file that will be used by the Smartesting CertifyIt generation tool.

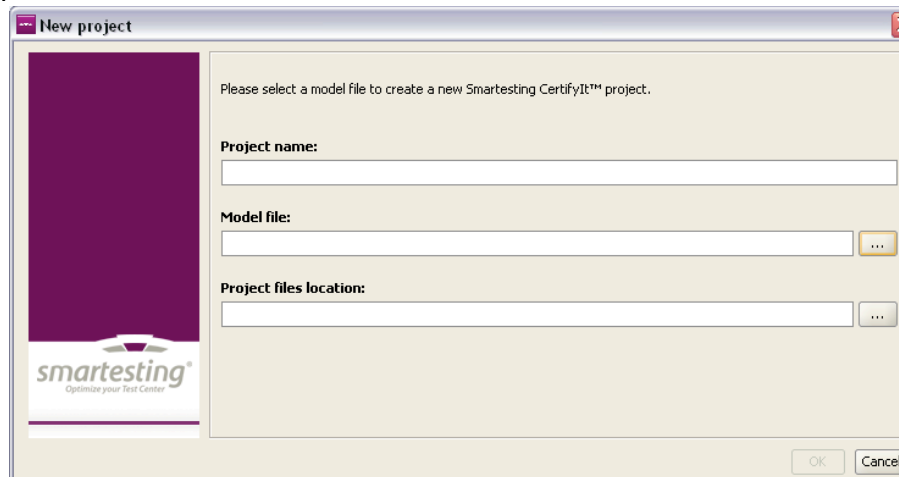


As it is exported, after launching Smartesting CertifyIt a new project can be created by clicking on *New project...*

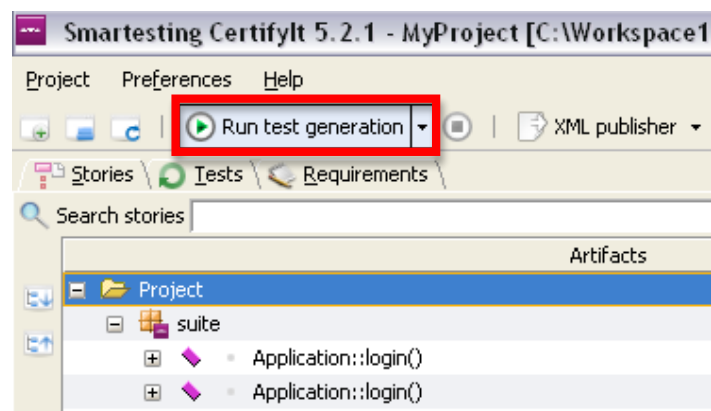
	<p align="center">Initial Security Testing Tools</p> <p align="center">Deliverable ID: D3.WP3</p>	Page : 59 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public



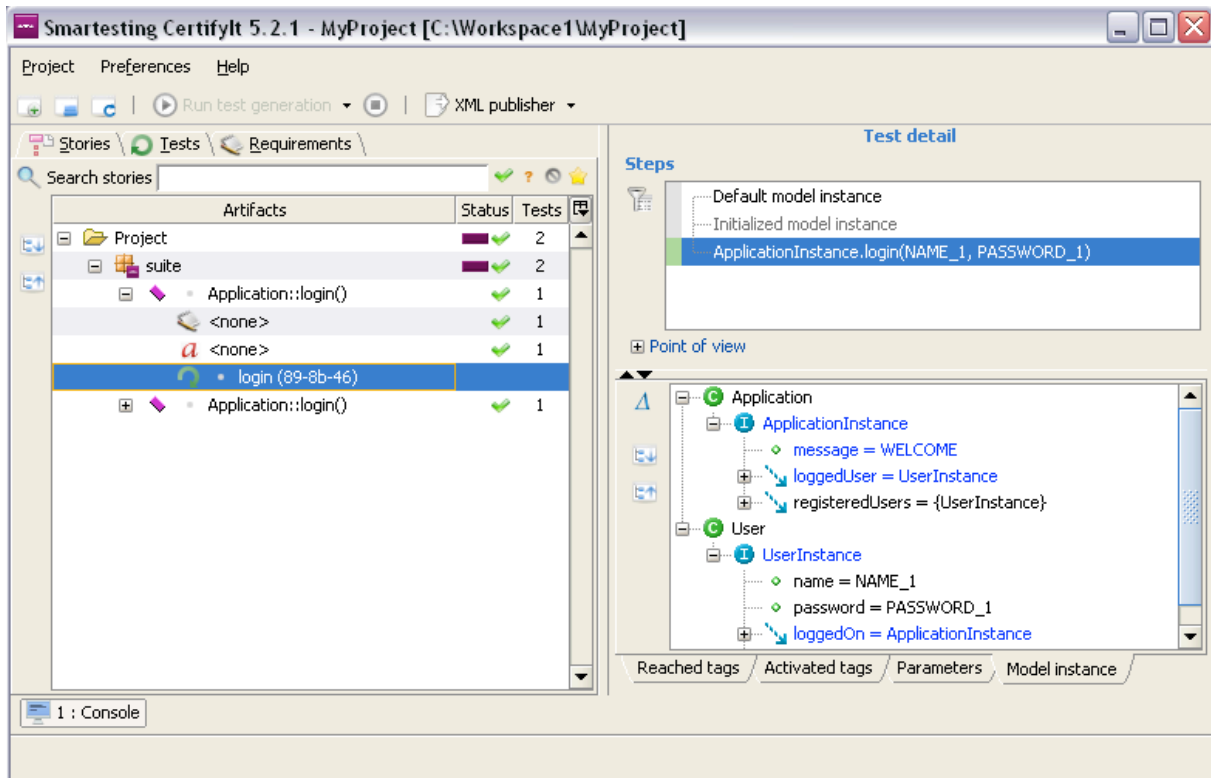
Name the project and choose the exported model file in the *Model file* field. A click on *Ok* will terminate the project creation.



Once the project is created, clicking on the *Run test generation* button launches the test generation.



For each behavior represented in the model, the generator creates a test sequence that will activate that behavior.



7.3.8 A step further: observations and traceability

Observation operations can also be created to check that the system reacted well to a dedicated stimulation.

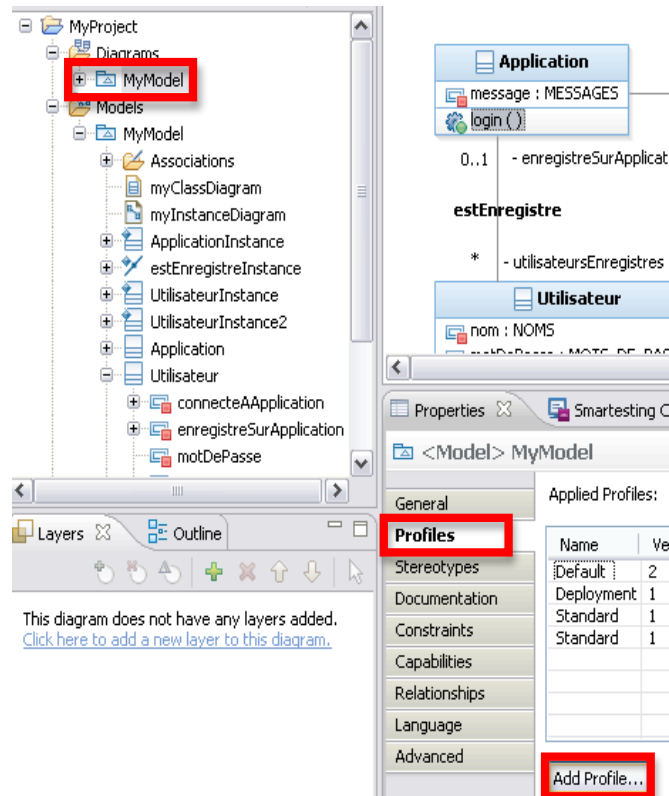
7.3.8.1 Creating an Observation operation

The Smartesting CertifyIt for IBM Rational Software Architect embeds a dedicated profile that extends UML to create specific kinds of operations.

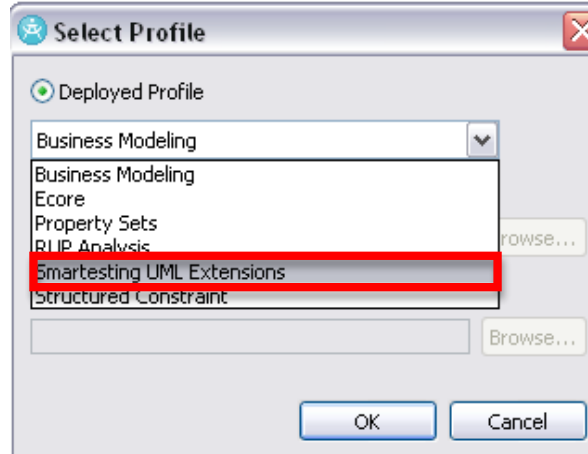
- The *setup* operation will be added at the beginning of each generated test sequence.
- The *teardown* operation will be added at the end of each generated test sequence.
- The *observation* operation will be added after a specific system state or operation call.

All those three kinds of operations cannot change the model state.

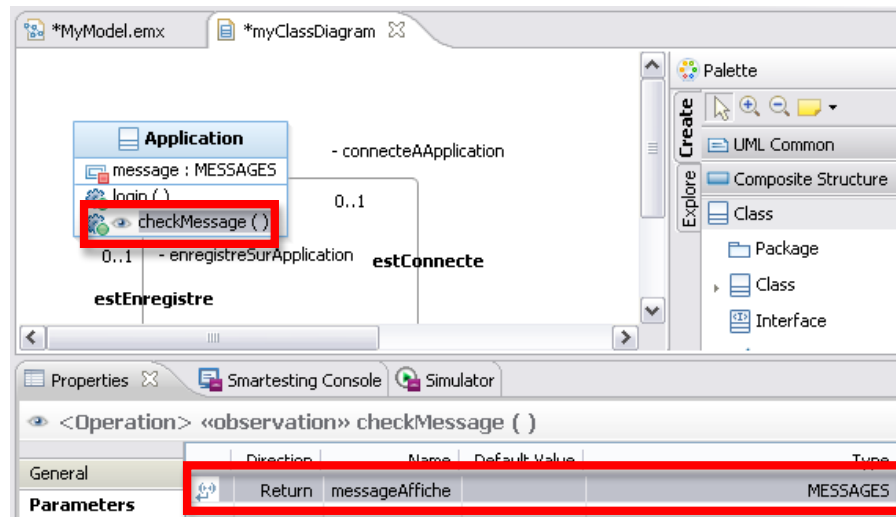
To activate those functionalities, select the model, and then in the *Profiles* tab of the *Properties* panel, choose *Add Profile...*



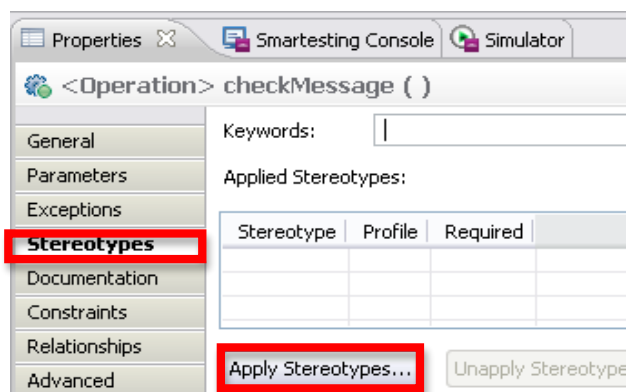
Then select the *Smartesting UML Extensions* and validate by clicking the *Ok* button.



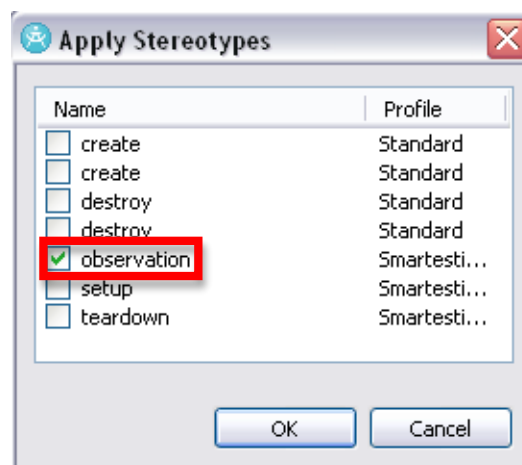
Once the profile deployed, you can create observation operations. To do so, create an operation with a return parameter as seen before (see §7.3.3.7).




You can now select the operation, and go in the *Stereotypes* part of the *Properties* tab, then click on *Apply Stereotypes*.

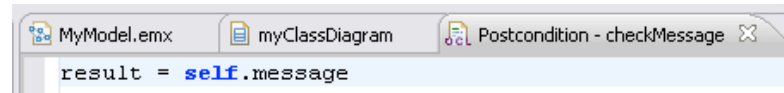


Select *observation* and validate by clicking the *Ok* button.

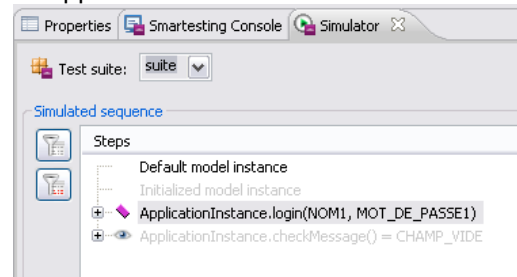


As seen in §7.3.3.9, add the following Postcondition to the operation.

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 63 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public



Now each time the “message” attribute value will change after an operation call on the model, the “checkMessage” operation will be called and return the current displayed message. This way we can observe the correct behavior of the application.

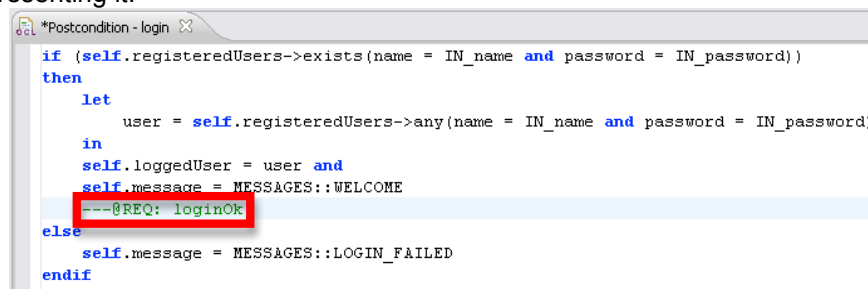


We have now seen how to create test sequences that use control points and observations of the System Under Test. A mechanism provided by the Smartesting tool enable the traceability link from the specification to the generated test case.

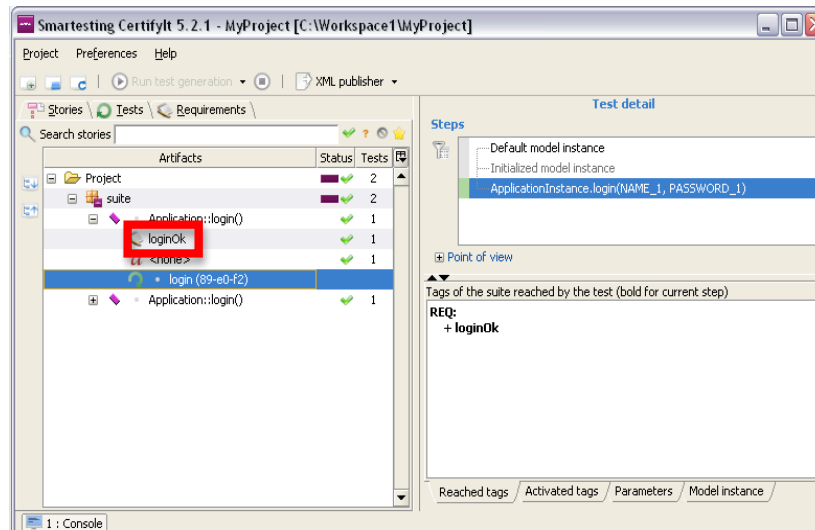
7.3.8.2 Adding Traceability tags

In any OCL code, annotations can be added in a specific format (`---@REQ: My_Requirement_Identifier`).

For example here the loginOk requirement presented in the application description is inserted in the portion of OCL code representing it.



The tool will automatically take it into account, and add this requirement identifier to the corresponding test.



It makes it easier to understand the test aim, and to measure the requirements coverage by the tests.

We have now seen all the steps to create the UML Model. By doing the same way all the functional requirements concerning the “cart” management can be represented.

The Test Generation Model being designed, Security Test objectives can now be defined with the Test Purpose language.

7.4 TEST PURPOSE LANGUAGE FOR SECURITY ORIENTED TEST GENERATION

7.4.1 Test Purpose Language Objectives

The test purposes are test selection criteria that define the way to generate tests from the test generation model.

The language is designed to be expressive enough to create tests from the test generation model.

The Test Purpose expresses a « pattern » (incomplete scenario) that the tests will have to cover, in respect to the functional model. It allows using the test generation model artifacts to express chains of specific states of the System Under Test (SUT) and specific behaviors of the SUT.

7.4.2 Test Purpose Language

To be able to generate test cases, the generation engine uses a formalized test generation criteria. The Test Purpose language grammar rules are the following:



Initial Security Testing Tools

Deliverable ID: **D3.WP3**

Page : 65 of 81

Version: 1.1

Date : 29.6.2012

Status : Final

Confid : Public

```

test_purpose ::= (quantifier_list COMA)? seq EOF;
quantifier_list ::= quantifier (COMA quantifier)*;
quantifier ::=
  | FOR_EACH BEHAVIOR var FROM behaviour_choice
  | FOR_EACH OPERATION var FROM op_choice
  | FOR_EACH LITERAL var FROM literal_choice
  | FOR_EACH INSTANCE var FROM instance_choice
  | FOR_EACH INTEGER var FROM integer_choice
  | FOR_EACH CALL var FROM call_choice;
op_choice ::=
  | ANY_OPERATION
  | ANY_OPERATION_BUT op_list;
call_choice ::= call_list;
behaviour_choice ::=
  | ANY_BEHAVIOR_TO_COVER
  | ANY_BEHAVIOR_TO_COVER_BUT behaviour_list;
literal_choice ::= IDENTIFIER (OR IDENTIFIER)*;
instance_choice ::= instance (OR instance)*;
integer_choice ::= CURLY_OPEN INT (COMA INT)+ CURLY_CLOSE;
var ::= DOLLAR IDENTIFIER;
state ::= ocl_constraint ON_INSTANCE instance;
instance ::= IDENTIFIER;
ocl_constraint ::= STRING_LITERAL;
seq ::= bloc (THEN bloc)*;
bloc ::= USE control restriction? target?;
restriction ::=
  | AT_LEAST_ONCE
  | ANY_NUMBER_OF_TIMES
  | INT TIMES
  | var TIMES;
target ::=
  | TO_REACH state
  | TO_ACTIVATE behaviour
  | TO_ACTIVATE var;
control ::=
  | op_choice
  | behaviour_choice
  | var
  | call_choice;
call_list ::= call (OR call)*;
op_list ::= operation (OR operation)*;
operation ::= IDENTIFIER;
call ::= instance '?' operation parameters;
parameters ::= PARENTHESIS_OPEN (parameter (COMA parameter)*)? PARENTHESIS_CLOSE;
parameter ::=
  | FREE_VALUE
  | IDENTIFIER
  | var
  | INT;
behaviour_list ::= behaviour (OR behaviour)*;
behaviour ::=
  | BEHAVIOR_WITH_TAGS tag_list
  | BEHAVIOR_WITHOUT_TAGS tag_list;
tag_list ::= CURLY_OPEN tag (COMA tag)* CURLY_CLOSE;
tag ::=
  | REQ COLON IDENTIFIER
  | AIM COLON IDENTIFIER;


```

The Test Purpose language terminals are:

```

TIMES ::= 'times';
FOR_EACH ::= 'for_each';
BEHAVIOR ::= 'behaviour';
OPERATION ::= 'operation';
INTEGER ::= 'integer';
CALL ::= 'call';
INSTANCE ::= 'instance';
LITERAL ::= 'literal';
FROM ::= 'from';
THEN ::= 'then';
USE ::= 'use';
TO_REACH ::= 'to_reach';
TO_ACTIVATE ::= 'to_activate';
ON_INSTANCE ::= 'on_instance';
ANY_OPERATION ::= 'any_operation';
ANY_OPERATION_BUT ::= 'any_operation_but';
OR ::= 'or';
ANY_BEHAVIOR_TO_COVER ::= 'any_behaviour_to_cover';
ANY_BEHAVIOR_TO_COVER_BUT ::= 'any_behaviour_to_cover_but';
BEHAVIOR_WITH_TAGS ::= 'behaviour_with_tags';
BEHAVIOR_WITHOUT_TAGS ::= 'behaviour_without_tags';
AT_LEAST_ONCE ::= 'at_least_once';
ANY_NUMBER_OF_TIMES ::= 'any_number_of_times';
COMA ::= ',';
CURLY_OPEN ::= '{';
CURLY_CLOSE ::= '}';
PARENTHESIS_OPEN ::= '(';
PARENTHESIS_CLOSE ::= ')';
COLON ::= ':';
DOLLAR ::= '$';
REQ ::= 'REQ';
AIM ::= 'AIM';
FREE_VALUE ::= 'FREE_VALUE';
DOT ::= '.';
IDENTIFIER ::= 'identifier';
EOF ::= '<EOF>';

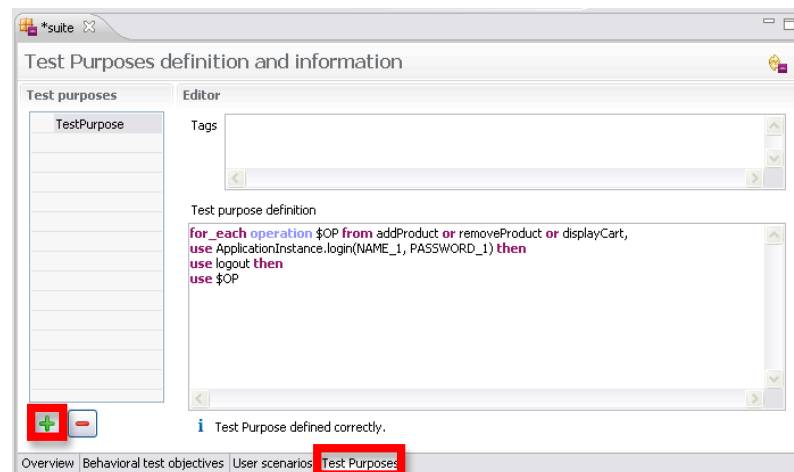
```

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 66 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

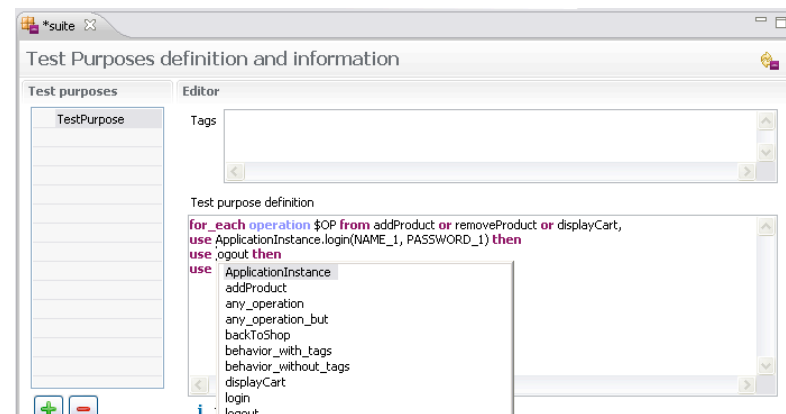
This grammar is used to construct test selection criteria in a dedicated Test Purpose Language Editor.

7.4.3 Test Purpose Language Editor

The Smartesting plugin for IBM Rational Architect offers a Test Purpose Editor. This is accessible via the test suite definition panel, by selecting the Test Purpose tab.



As seen on the previous image, it offers the possibility to add to the test suite several Test Purposes. Syntax highlighting and code completion with the model elements are also provided.



All the test purposes defined here will be taken into account by the test generator. A strategy has been implemented to make it possible for the generator to interpret the test purposes.


7.4.4 Test Generation Strategy from Test Purpose

Each Test Purpose is constructed in the same way. A first part of the test purpose defines variables that will be used by the second part, which is dedicated to the states and behaviors chains descriptions.

For our example, we identified the following risk:

After logout, the cart should not be accessible, and no operation (adding or removing product) should be done on it.

Testing this would consist in login one user, disconnect him the try to add, remove products, or navigate to the cart. This can be translated this way into Test Purpose language:

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 67 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

```

for_each operation $OP from addProduct or removeProduct or displayCart,
use login to_reach "not (self.loggedUser.ocIsUndefined())" on_instance ApplicationInstance then
use logout then
use $OP

```

The strategy consists in creating one test objective for each possible instantiation of the variables in the first part of the test purpose. We would obtain here three test objectives:

```

use login to_reach "not (self.loggedUser.ocIsUndefined())" on_instance ApplicationInstance then
use logout then
use addProduct

```

```

use login to_reach "not (self.loggedUser.ocIsUndefined())" on_instance ApplicationInstance then
use logout then
use removeProduct

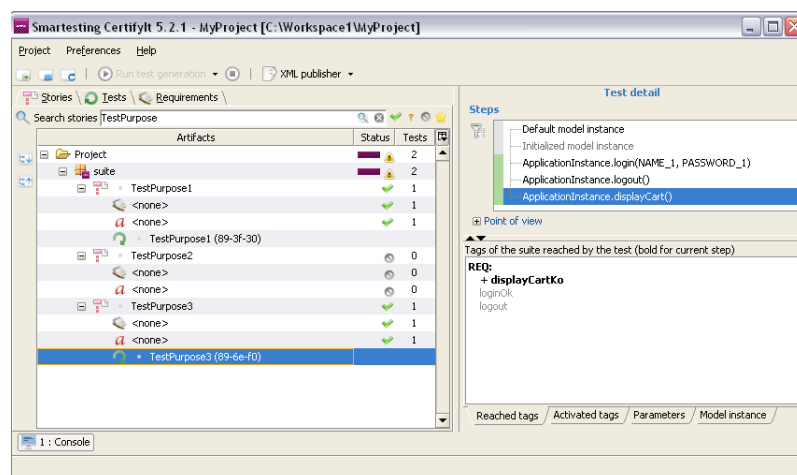
```

```


use login to_reach "not (self.loggedUser.ocIsUndefined())" on_instance ApplicationInstance then
use logout then
use displayCart

```

For each of those objectives the test generation will create, when possible, a test case.



Several variables can be created in the first part of the Test Purpose. When more than one is defined, the Test Purpose is instantiated with all the possible value combinations for those variables. For example of there are 2 defined variables, one with 2 possible values and the second with 3 possible values, $2 \times 3 = 6$ test objectives will be created.

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 68 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

8. FRAMEWORK FOR ACTIVE SECURITY TESTING (FSCOM)

FSCOM, member of ETSI and technology partner of Testing Technologies, is focused on active testing targeting communication protocols security and conformance. FSCOM provides solutions based on TTCN-3 [2], standardized and promoted by ETSI. FSCOM has developed a prototype for active security testing based on Thales Waveform case study [1].

Typical threat scenarios for the Thales use case occur when an attacker tries to use the vulnerabilities of the Waveform components like the PHY or the MAC layer in order to:

- Usurp the identity of another node by spoofing MAC addresses of other nodes (intrusion);
- Cause severe degradation of network performance (Dos and DDoS);
- Inject protocol messages in the network traffic (network injection);

The present framework provides a 'black box' approach based on functional test case design based on an analysis of the specification of the Waveform components without reference to its internal functional structure. Only Service Access Points and their Service Primitives are considered. The modelization of several radio nodes acting as 'good' and 'malicious' nodes and their dynamic interaction with the radio node(s) under test (IUT) will be done by using TTCN-3 components.

Attacks on a network can be detected by passive monitoring (by listening to the traffic without disturbing the network) or active attacks.

The FSCOM approach is focused on active attacks applying the strength of the TTCN-3 related test methodologies to new fields, i.e. testing of security aspects and testing within the environment of a dynamically re-routing ad-hoc radio network. One or more malicious nodes modeled with TTCN-3 components will actively disturb the normal operation of the network using attacks such as modification, impersonation and fabrication of protocol messages.

However, due to the nature of the chosen test methodology, physical attacks (that may involve a powerful transmitter broadcasting a constant noise in the used frequencies) are out of scope of this approach.

8.1 INTRODUCTION TO THE SECURITY TESTING FRAMEWORK

The security testing framework is made up by a set of tools that provide:


- Identification of the candidate implementations under test (IUT) for security testing;
- Definition of the applicable tests, i.e. answering the question "what and how to be tested";
- Development of the resulting test specifications.

For the Thales radio network use case the following security testing related topics will be covered in the subsequent clauses:

- Identification of candidate EUTs/IUTs.
- Identification of test scenarios.
- Definition of test bed architecture.
- Identification of test bed interfaces.

8.2 SECURITY TESTING

The following clauses provide security testing methodologies on which FSCOM based the active security test framework.

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 69 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

8.2.1 Candidate EUTs/IUTs

For security testing, both "Implementation Under Test" (IUT) and "Equipment Under Test" (EUT) are considered. An EUT is a physical implementation of one or more network layers (IUT), which interact with one or several other EUTs via one or more reference points (RPs).

8.2.2 Test scenarios

In security, a large number of use cases are already identified. In a specific implementation of an IUT, very likely only a sub-set of these use cases is supported. In order to perform the tests, IUTs supporting the same use cases are required.

8.2.3 Test bed architecture


The "System Under Test" (SUT) contains (refer to [2], [3] and [4]):

- The "Implementation Under Test" (IUT), i.e. the object of the test.
- The "Upper tester application" enables to simulate sending or receiving service primitives from protocol layers above the IUT or from the management/security entity.
- The "Lower tester" enables to establish a proper connection to the system under test (SUT) over a physical link (Lower layers link). The lower layers link is located at a "Reference Point" as Service Access Point (SAP).
- The "Upper tester transport" enables the test system to communicate with the upper tester application. Then the upper tester can be controlled by a TTCN-3 test component as part of the test process.

The "Security test system" contains (refer to [2], [3] and [4]):

- The "TTCN-3 test components" are processes providing the test behaviour. The test behaviour may be provided as one single process or may require several independent processes.
- The "Codec" is a functional part of the test system to encode and decode messages between the TTCN-3 internal data representation and the format required by the related base protocol standard.
- The "Test Control" enables the management of the TTCN-3 test execution (parameter input, logs, test selection, etc.).
- The "Test adapter" (TA) realizes the interface between the TTCN-3 ports using TTCN-3 messages, and the physical interfaces provided by the IUT.

Figure 1 describes the functional architecture of the security test bed:

	<p align="center">Initial Security Testing Tools</p> <p align="center">Deliverable ID: D3.WP3</p>	Page : 70 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

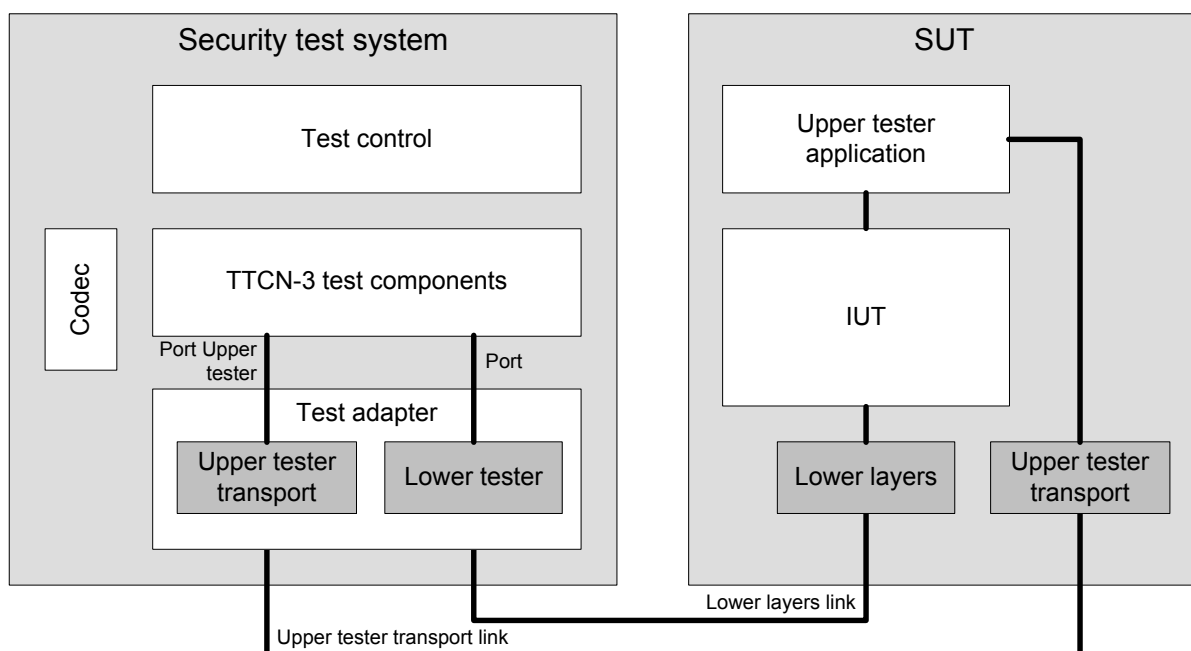


Figure1: Security test system architecture

8.3 IDENTIFICATION OF ABSTRACT TEST METHOD

An abstract protocol tester presented in figure 2 below is a process providing the test behaviour for testing an IUT. Consequently it will emulate a peer IUT of the same layer/the same entity. This type of test architecture provides a situation of communication which is equivalent to real operation between real devices. The security test system will simulate valid and invalid protocol behaviour, and will analyse the reaction of the IUT. Then the test verdict, e.g. pass or fail, will depend on the result of this analysis. Thus this type of test architecture enables to focus the test objective on the IUT behaviour only. In order to access an IUT, the corresponding abstract protocol tester needs to use lower layers to establish a proper connection to the system under test (SUT) over a physical link (Lower layers link).

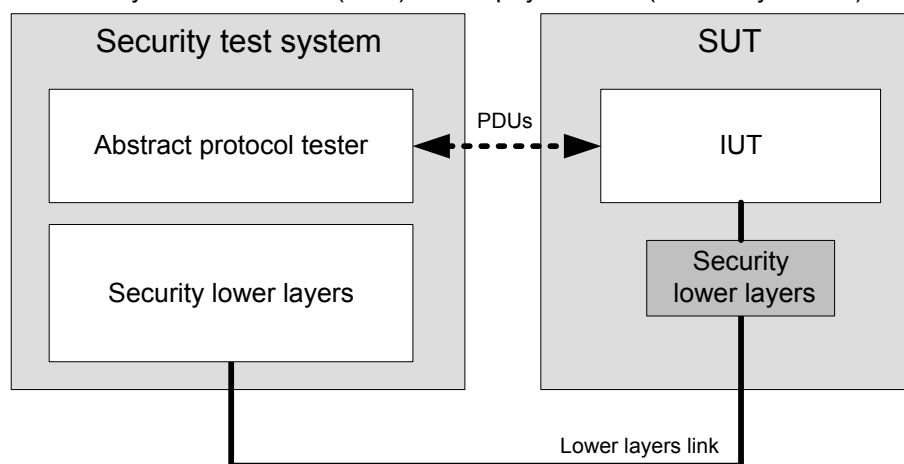


Figure2: Generic abstract protocol tester

The "Protocol Data Units" (PDUs) are the messages exchanged between the IUT and the abstract protocol tester as specified in the base standard of the IUT. These PDUs are used to trigger the IUT and to analyse

the reaction from the IUT on a trigger. Comparison of the result of the analysis with the requirements specified in the base standard allows assigning the test verdict.

8.4 PROTOCOL DESCRIPTION USING ASN.1

To use our framework efficiently, the protocol messages and information elements to be tested are described using ASN.1 [5] & [7]. An automatic tool, part of the framework, converts the ASN.1 protocol messages and information elements descriptions into TTCN-3 types. This typing shall be used to develop the Abstract Test Suite (ATS).

8.5 INTEGRATION OF THE SECURITY FRAMEWORK WITH OMNET++

In the case of an IUT executed in the OMNeT++ environment, the security test bed is configured as described in figure 3:

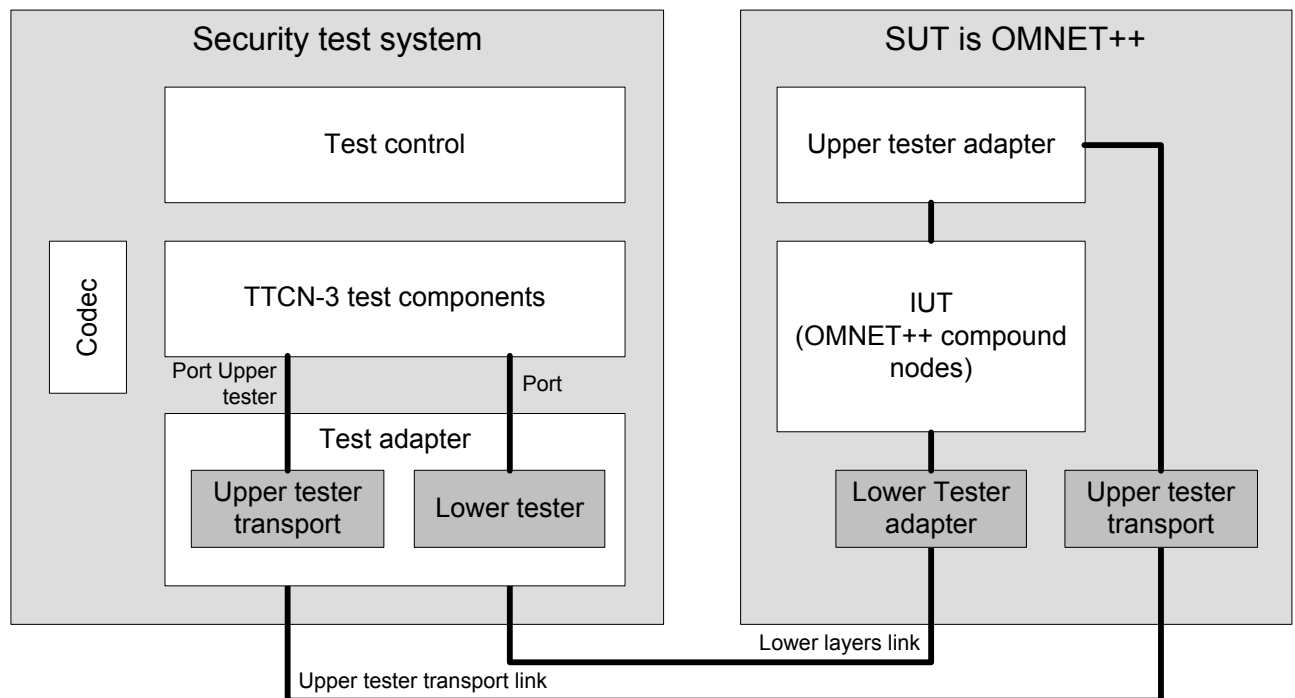


Figure 3: Security test system architecture with OMNeT++ support

Both software entities Upper tester adapter and Lower tester adapter are now part of the OMNeT++ environment. Both entities are separated into two software components:

- The first one is generic and shall be re-used with any IUT implemented on OMNeT++. This component communicates with the test adapter;
- The second one is specific to each IUT implementation. This component provides a 'design pattern' implementation (as strategy, decorator...) which facilitates adaptation to any IUT to be validated.

Different communication ways should be supported between these entities and the test adapter. Currently, UDP is used. Figure 4 below shows the architecture of the test bed inside OMNeT++: The node cUMacModule is the IUT.

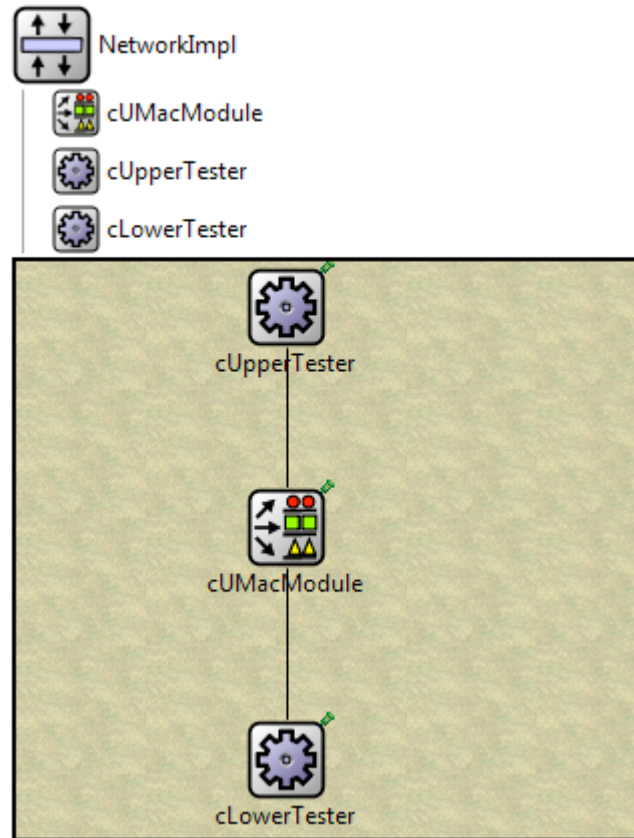


Figure4: Security test system architecture on OMNeT++ side


8.6 TOOLS TO CONVERT MONTIMAGE SECURITY RULES INTO ETSI TPLAN LANGUAGE

ETSI has introduced TPlan in 2009 [8]. TPlan is defined with a minimal set of test-oriented keywords but owns the capability that permits users to define extensions to the notation. The benefits of using TPlan are:

- Consistency in test purpose descriptions - less room for misinterpretation;
- Clear identification of the TP pre-conditions, test body, and verdict criteria;
- Automatic syntax checking and syntax highlighting in text editors;
- A basis for a TP transfers format and representation in tools.


FSCOM envisages to develop a tool, part of our security test framework, to translate security rules introduced by MonImage [9] into TPlan [8]. The report on this activity will be included together with the results of the active testing activities in FSCOM's contribution to the next deliverable of WP3.

This truly innovative tool will help developers during the development of test specifications (refer to clause **Error! Reference source not found.**). Based on security rules defined on the protocol to be tested, this tool translates them into a set of test purpose descriptions using standardized language TPlan [8]. This is one of the most important steps for developing an ATS (Abstract Test Suite). Effectively, each test purpose description will be transformed in a TTCN-3 test case where the test behaviour of the radio nodes is modelled.


	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 73 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

REFERENCES

- [1] Diamonds_FPP_v1_3.pdf: " DIAMONDS: Development and Industrial Application of Multi-Domain Security Testing Technologies".
- [2] ETSI standard ES 201 873-1 V3.4.1: The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language. European Telecommunications Standards Institute.
- [3] ETS 300 406 (1995): "Methods for testing and Specification (MTS); Protocol and profile conformance testing specifications; Standardization methodology".
- [4] ETSI ES 201 873-1: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language".
- [5] Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation (ISO/IEC 8824-1:2008, IDT).
- [6] ETSI ES 201 873-7: "Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 7: Using ASN.1 with TTCN-3".
- [7] 3GPP TR 25.921: "Guidelines and principles for protocol description and error handling"
- [8] ETSI ES 202 553: Methods for Testing and Specification (MTS); TPLan: A notation for expressing Test Purposes
- [9] DIAMONDS Consortium D2.WP3: Initial Design of Security Testing Tools V1.0 30112011
- [10] The CreMA tool (itemis): <http://www.guersoy.net/knowledge/crema>, as of date 4.5.2012
- [11] The Eclipse Connected Data Objects (CDO) Model Repository: <http://www.eclipse.org/projects/project.php?id=modeling.emf.cdo>, as of date 8.5.2012
- [12] What is JoSQL, <http://josql.sourceforge.net/>, as of date 8.5.2012
- [13] The Papyrus tool: <http://www.eclipse.org/modeling/mdt/papyrus/>, as of date 8.5.2012
- [14] The ProR tool: <http://www.eclipse.org/rmf/pror/>, as of date 8.5.2012
- [15] Requirements Interchange Format (ReqIF): Object Management Group (OMG), <http://www.omg.org/spec/ReqIF/1.0.1/>, 2011-04-02
- [16] OMG Systems Modeling Language (OMG SysML™): Object Management Group (OMG), <http://www.sysml.org/specs/>, 2010-06-02
- [17] Charles Miller: Fuzz By Number. CanSecWest 08. <http://cansecwest.com/csw08/csw08-miller.pdf>
- [18] Jack Koziol: Fuzzers - The ultimate list. InfoSec Institute. <http://www.infosecinstitute.com/blog/2005/12/fuzzers-ultimate-list.html>
- [19] C DeMott, J., Miller. C.: Fuzzing for Software Security Testing and Quality Assurance. Artech House, Boston (2008)
- [20] W. Mallouli, Diamonds deliverable D2.WP2: "Concepts for Model-Based Security Testing".
- [21] Object Management Group (OMG): MOF 2 XMI Mapping. <http://www.omg.org/spec/XMI/>
- [22] Eclipse: MDT/UML2. <http://wiki.eclipse.org/MDT-UML2>
- [23] Eclipse: MDT-UML2-Tool-Compatibility. <http://wiki.eclipse.org/MDT-UML2-Tool-Compatibility>
- [24] The Jython Project. <http://www.jython.org/>

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 74 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

- [25] A. Takanen, Diamonds deliverable D3.WP1: “Initial case study results”
- [26] M. Utting and B. Legeard: “Practical Model-Based Testing - a tools approach”, Morgan Kaufmann, 2006

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 75 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

APPENDIX

Appendix A: Extraction code examples for MMT extraction library

With “pcap”

Note that to compile this main the “pcap” open source library is need. Plugins for the more common internet protocols are included in the MMT package.

In this example the `debug_extracted_attributes_printout_handler` function is used as callback. It is part of the MMT-Extract library and will print out all the data, corresponding to the registered attributes, extracted from each processed packet. An other callback can be implemented by the user. For instance, to perform traffic analysis, Montimage has developed an a callback function that will analyse the captured information and publish the results in a database that is used by the MMT-Operator application that will display the results graphically in near real-time.

```
#include <stdio.h>
#include <stdlib.h>
#include <pcap.h>
#include "mmt_core.h"

int main(int argc, char** argv) {
    pcap_t *pcap;
    const unsigned char *data;
    struct pcap_pkthdr p_pkthdr;
    struct pkthdr header;
    char errbuf[1024];

    init_extraction(); //This will initialize the extraction library

    //We will register a number of attributes to extract
    register_extraction_attribute_by_name("BASE", "UTIME");
    register_extraction_attribute_by_name("ARP", "ARP_OPCODE");
    register_extraction_attribute_by_name("IP", "IP_PROTO_ID");
    register_extraction_attribute_by_name("ETHERNET", "ETH_PROTOCOL");
    register_extraction_attribute_by_name("UDP", "UDP_SRC_PORT");

    //We register a packet handler
    /* print_extracted_attributes is a utility function that prints out
     * registered attributes
     */
    register_packet_handler(1, debug_extracted_attributes_printout_handler, NULL);

    pcap = pcap_open_offline(argv[1], errbuf); // open offline trace
    if (!pcap) { /* pcap error ? */
        fprintf(stderr, "Error 105: pcap_open failed for the following reason:
%s\n", errbuf);
        return;
    }

    while ((data = pcap_next(pcap, &p_pkthdr))) {
        header.ts = p_pkthdr.ts;
        header.caplen = p_pkthdr.caplen;
    }
}
```




Initial Security Testing Tools

Deliverable ID: **D3.WP3**

Page : 76 of 81

Version: 1.1

Date : 29.6.2012

Status : Final

Confid : Public

```
header.len = p_pkthdr.len;

//Send the packet to the MMT-Core for processing
if (!packet_process(&header, data)) {
    fprintf(stderr, "Error 106: Packet data extraction failure.\n");
}


close_extraction(); //Close the extraction before exiting

return (EXIT_SUCCESS);
}
```

With “Thales protocol”

Note that to execute this main, the Thales protocol plugin is needed.

[illegible]

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 77 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

```

header.slot_nb = 5; // just an example

// set the data link to Thales proto
setDataLinkType (THALES_TDMA_PROTO);

//Send the packet to the MMT-Core for processing
packet_process(&header, pkt1);

close_extraction(); //Close the extraction before exiting

return (EXIT_SUCCESS);
}

```

Appendix B: Security analysis example

With “Thales protocol”

In this example, the traces are read from a file generated by the the Thales simulator Omnet+. Thales has also adapted this code, integrating it with the simulator, so that the Security analysis is done online.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>

#define THALES_TDMA_PROTO 0x1000

static FILE * OutputFile = NULL; //XML results output file
static FILE * InputFile = NULL; //XML properties input file
static int file_no = 0;
typedef void (*result_callback) (char *xml_string);

typedef struct pkthdr {
    struct timeval ts; //Time stamp that indicates the packet arrival time
    unsigned int caplen; //Length of portion of the packet that is present
    unsigned int len; //Length of the packet (off wire)
    int slot_nb; //Slot number used for TDMA protocols
} pkthdr_t;

//MMT_SecurityLib and MMT_ExtractLib functions used by main


//MMT_SecurityLib function to initialise the MMT_SecurityLib library
extern void init_sec_lib(char * property_file,
    short option_satisfied, short option_not_satisfied,
    result_callback todo_when_property_is_satisfied_or_not);

//MMT_SecurityLib function to recuperate the summary of results in XML format
extern char * xml_summary();

//MMT_SecurityLib function to indicate protocol used
extern void setDataLinkType(int dltype);

//MMT_ExtractLib function to process read or captured packet or message
extern int packet_process(struct pkthdr *header, const char * packet);

```

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 78 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

```
//Next three functions can be changed to do any necessary treatment of results.

void todo_at_start() {
    file_no++;
    char file_name[100];
    //XML file that will contain the results (see bellow for more details)
    sprintf(file_name, "result_%d.xml", file_no);
    OutputFile = fopen(file_name, "w");
    char *xml_string = "<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>\n<?xml-stylesheet type=\"text/xsl\" href=\"results.xsl\"?>\n<results>\n<detail>\n";
    fprintf(OutputFile, "%s\n", xml_string);
}

void todo_when_property_is_satisfied_or_not(char * xml_string) {
    //The xml_string contains the XML fragments describing the detected property
    and
    //the list of events that lead to it (see bellow for more details)
    fprintf(OutputFile, "%s\n", xml_string);
};

void todo_at_end() {
    //The xml_string contains XML fragment with a summary of results
    //(see bellow for more details)
    char * xml_string;
    xml_string = xml_summary();
    fprintf(OutputFile, "%s\n", xml_string);
    free(xml_string);
    fclose(OutputFile);
}

int main(int argc, char **argv) {

    //Execute the initialisation function to define the XML properties file,
    //options and a pointer to the callback function. The following arguments
    //are used:
    // Argument 1: properties file (can be changed)
    // Argument 2: if 1 then results will contain details on properties
    //satisfied
    // Argument 3: if 1 then results will contain details on properties not
    //satisfied
    // Argument 4: name of function that will be executed each time a property
    //is
    // satisfied or not

    if (argc < 3) {
        fprintf(stderr, "usage: program_name properties_file trace_file \n");
        exit(-1);
    }
    init_sec_lib(argv[1], 1, 1, todo_when_property_is_satisfied_or_not);

    char *recuperated_message = malloc(3000); //Holds the recuperated message

    //Can change name of input trace file (but should have extension .tdma)
    InputFile = fopen(argv[2], "r");
    static struct pkthdr header; //Header containing slot number
    setDataLinkType(THALES_TDMA_PROTO); //Set protocol to be TDMA
```



Initial Security Testing Tools

Deliverable ID: **D3.WP3**

Page : 79 of 81

Version: 1.1

Date : 29.6.2012

Status : Final

Confid : Public

```
todo_at_start();


//Loop that reads a message from input trace file
while (fgets(recuperated_message, 3000, InputFile) != NULL) {
    //Process each message
    char message[3000]; //Holds the message ready for processing
    int i, j;
    char * tmp_line = NULL;
    char *token = NULL;
    char *search = "=: ()[]\t\n\0";
    int char_count = 0;
    strcpy(message, recuperated_message);
    tmp_line = message;
    token = strtok(tmp_line, search);

    //Set header for the message (slot number)
    if (strcmp(token, "TS") == 0) {
        token = strtok(NULL, search);
        header.ts.tv_sec = 0;
        header.ts.tv_usec = 0;
        header.caplen = 0;
        header.len = 0;
        header.slot_nb = atoi(token);

        //Find begining of message
        if (strchr(token, '_') == NULL) {
            char_count = token - message;
            while (*(message + char_count) != '\t') {
                char_count++;
            }
            while (*(message + char_count) == '\t' || *(message +
char_count) == ' ') {
                char_count++;
            }
        }

        //Prepare message
        i = 0;
        j = char_count;
        while (message[j] != '\n' && message[j] != '\0') {
            if (message[j] != ' ') {
                message[i] = message[j];
                i++;
            }
            j++;
        }
        message[i] = '\0';

        //Call MMT_ExtractLib function that will parse the message and
analyse it.
        //When a security property is satisfied or not, the callback
        //function todo_when_property_is_satisfied_or_not will be executed.
        packet_process(&header, message);
    }
}
```


	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 80 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

```
//To finish results file (e.g. write summary in the XML file)
todo_at_end();

return (EXIT_SUCCESS);
}
/*
Description of the results obtained by the MMT_Security example module
-----
In this example (i.e. main.c file) we provide functions necessary to write out
the results obtained by the MMT_Security module applied to a TDMA trace file.
The code can also be used as an example of how to analyse on-line capture of
messages. The XML generated by the module gives all the results obtained. This
XML can be printed out to a XML results file and viewed using a browser or it
can be parsed to treat it as wished. The XML results file produced by this
example will contain:
* header lines starting with "<?xml";
* a results section (tags <results> and </results>) that contains the following
  two sub-sections;
* a detail sub_section (tags <detail> and </detail>) that gives all the
  information on the properties found or not and the events that caused
  this result;
* a summary sub-section (tags <summary> and </summary>) that gives the total
  number of properties found or not for each different property.
These sections are described below.

The todo_at_start function will just write the beginning of the XML result file.
This function can be eliminated if this file does not correspond to the results
wanted.
This is the XML text written in this example:
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="results.xsl"?>
<results>      (start of the results section)
  <detail>      (start of the details section)
The results.xsl and results.css files define respectively the transformations
and style to obtain a user friendly format when viewed by a browser.

The todo_when_property_is_satisfied_or_not function will be executed when a
property specified in the properties file (properties.xml in this example) is
satisfied or not. This is the XML text fragment that will be returned with
comments explaining each line:
<occurence>      (XML tag identifying a result obtained for one
                  particular property)
  <pid>x1</pid>    (x1 will be the number identifying the property)
  <verdict>x2</verdict> (x2 will be a word: respected or not_respected or
                  detected or not_detected giving respectively the
                  status of a security rule and an attack)
  <description>    (XML tag identifying text x3 describing the property)
    x3
  </description>
  <event>          (XML tag identifying an event that occurred that was
                  part of the event sequence that lead to this
                  occurrence. Several events could exist and are listed
                  by repeating this tag)
    <description>  (XML tag identifying text x4 describing the event)
      x4
    </description>
```

	<p style="text-align: center;">Initial Security Testing Tools</p> <p style="text-align: center;">Deliverable ID: D3.WP3</p>	Page : 81 of 81
		Version: 1.1 Date : 29.6.2012
		Status : Final Confid : Public

```

<attribute>          (XML tag identifying the start of the list of
                      attributes names and values defining the event)
  <attribute_value>   (XML tag identifying an attribute name x5 and value
                      x6. Several attribute values could exist and are
                      listed by repeating this tag)
    - - - - - x5=x6 (note that the - are used for indentation and should be
                      ignored)
  </attribute_value>
</attribute>
</event>
</occurrence>

```

The todo_at_end function will write a summary of the results obtained and the end of the XML result file.

This function can be eliminated if this file does not correspond to the results wanted.

This is the XML text written in this example:

```

</detail>           (XML tag that ends preceeding section)
<summary>           (XML tag that starts summary section)
  <sp>               (for each security rule, this XML text segment will give the
                      total number of rules respected and or violated)
    <id>x1</id>      (x1 gives the number identifying the rule)
    <description>SECURITY RULE: x2</description> (x2 gives a description)
    <respected>x3</respected> (x3 gives the number of times the rule was
                              respected)
    <violated>x4</violated>  (x4 gives the number of times the rule was
                              violated)
  </sp>              (this XML segment will exist for each security rule specified
                      in the XML properties file)
  <spb>              (same as for the above XML text segment, but for attacks)
    <id>x1</id>
    <description>ATTACK: x2</description>
    <respected>x3</respected>
    <violated>x4</violated>
  </spb>              (this XML segment will exist for each attack specified in
                      the XML properties file)
</summary>          (end of the summary section)
</results>          (end of the results section)
*/

```