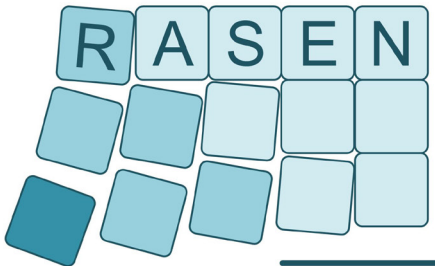

Compositional Risk Assessment and Security Testing of Networked Systems

Deliverable D5.4.2

A Toolbox for Security Risk Assessment and Security Testing

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

Work package:	WP5
Deliverable number:	D5.4.2
Nature of deliverable:	Report
Dissemination level:	PU
Internal version number:	1.0
Contractual delivery date:	2014-09-30
Actual delivery date:	2014-09-30
Responsible partner:	FhG FOKUS

Contributors

Editor(s)	Jürgen Großmann (FOKUS)
Contributor(s)	Fredrik Seehusen (SINTEF), Jürgen Großmann, Michael Berger (FOKUS).
Quality assuror(s)	Arthur Molnar (Info World), Frank Werner (Software AG)

Version history

Version	Date	Description
0.1	14-06-10	TOC
0.2	14-09-05	Input SINTEF
0.3	14-09-05	Input FOKUS
0.4	14-09-10	Final input FOKUS
0.5	14-09-11	Schema changes
0.6	14-09-19	Integration of internal review results
1.0	14-09-29	Final quality check

Abstract

The RASEN risk assessment and security testing toolbox provides integration support for the RASEN approach to risk-based security testing and test-based security risk assessment. This deliverable contains updates of the RASEN Data Integration Model, the specification of the export and import interfaces and the definition of RASEN Data Exchange Format.

Keywords

RASEN, Security Risk Assessment, Security Testing, Tool Integration

Executive Summary

The RASEN risk assessment and security testing toolbox provides integration support for the RASEN approach to risk-based security testing and test-based security risk assessment. The overall integration approach aims for defining a lightweight data exchange format on basis of XML. The format specifies the artifacts and elements that are required by multiple tools. These artifacts and elements are modeled by means UML in the RASEN Data Integration Model. The model is taken to serve as basis for the interface definitions at tools and as the source for XML schema generation.

This deliverable contains updates of the RASEN Data Integration Model, updates of the export and import interfaces and, finally, the actual definition of the RASEN Data Exchange Format. The RASEN Data Exchange is given by means of modular XML schema definitions and can be used for bilateral data exchange between the tools of the RASEN toolbox. In addition, this deliverable provides a short documentation how the XML schema files could be used to actual implement export/import tool adapters based on the Eclipse EMF framework.

Table of Contents

1 INTRODUCTION 5

2 UPDATE OF THE CONCEPTUAL RASEN TOOLS 6

3 UPDATE OF THE RASEN DATA INTEGRATION MODELS 8

3.1 UPDATE OF THE FOUNDATION PACKAGE 8
3.2 UPDATE OF THE RISK ASSESSMENT PACKAGE 9
3.3 UPDATE OF THE TESTING PACKAGE 9
3.3.1 Test Pattern 9
3.3.2 Test Report 11
3.4 UPDATE FOR THE RELATION BETWEEN RISK ASSESSMENT AND TESTING 11

4 UPDATE OF THE INTEGRATION INTERFACES 12

5 THE RASEN DATA EXCHANGE FORMAT AND ITS APPLICATION 14

5.1 PRINCIPLES OF THE RASEN DATA EXCHANGE FORMAT 14
5.2 HOW TO CREATE AN ADAPTER 15
5.2.1 The Eclipse Example Workspace 15
5.2.2 The Structure of an Export/Import Adapter 16
5.2.2.1 Example 1: Test report is available as plain text in proprietary format 18
5.2.2.2 Example 2: Test format is available in proprietary XML format 18
5.2.3 The Generation Process Step by Step 19

6 SUMMARY AND OUTLOOK 23

REFERENCES 24

APPENDIX A: RISK MODEL SCHEMA 25

APPENDIX B: TEST PATTERN SCHEMA 27

APPENDIX B: TEST REPORT SCHEMA 30

APPENDIX D: FOUNDATION SCHEMA 33

1 Introduction

The RASEN techniques and methodologies aim explicitly for the integration of data from security risk assessment and security testing. The RASEN deliverables D5.2.1 and D5.3.1 have provided the conceptual basis for a Data Exchange Format that allows for a systematic data exchange between the RASEN tools. This deliverable provides updates to the specifications that have been already provided by D5.2.1 and D5.3.1 so that the conceptual model and the interfaces become aligned with results and changes that are introduced by the other technical work packages. On basis of that, this deliverable introduces the specification of an XML-based data exchange format that allows for a systematic exchange of data between RASEN tools.

In general, the overall tool integration approach in the RASEN approach has been defined as follows:

1. Create deployable generic security testing and security risk assessment models. These models will be designed in UML.
2. Create a deployable generic risk assessment and testing model (RASEN Data Integration Model) on basis of the models defined beforehand.
3. Define conceptual tools and their interfaces
4. Instantiate a common XML-based exchange format on basis of the RASEN Data Integration Model.
5. Develop export/import adapter for the relevant instantiations of the conceptual tools (concrete tools) with respect to the XML format from 4 and the interfaces from 3.
6. Identify additional integration use cases and update the models if necessary.

While this document describes updates for the steps 1 to 3, it specifically adds the RASEN Data Exchange Format that allows exchanging artifacts between the RASEN tools. It specifically addresses the processes of risk based security testing and test based risk assessment and thus concentrates on the interchange of data between the RASEN risk assessment tools and the RASEN testing tools. Section 2 presents updates with respect to the classification of the conceptual RASEN tools. Section 3provides an update of the data integration model, which is used as the conceptual basis for the RASEN Data Exchange Format. Based on these data models, Section 4 specifies the integration interfaces for the RASEN toolbox and Section 5 introduces the RASEN Data Exchange Format and introduces a development environment that can be used to easily develop adapters to serve the exchange format. Section 6 concludes this report. Finally, the actual XML schema specification of the RASEN Data Exchange Format is given in Appendix A-D.

2 Update of the Conceptual RASEN Tools

The following table introduces five different conceptual tools as targeted by the RASEN project. This table is an update of a similar table in D5.2.1.

Conceptual tool	Concrete tool
Security Risk Assessment Tool (SRAT) for supporting compositional risk assessment	**CORAS** risk assessment tool (developed by SINTEF) **RACOMAT** risk assessment and testing tool (developed by Fraunhofer) **ARIS Business Architect** providing models for risk assessment (developed by Software AG)
Security Test Pattern Database (PDB) for managing test patterns	**RASEN Test Pattern Database** (developed by Fraunhofer)
Security Test Derivation Tool (STDT) for supporting the derivation and prioritization of test cases based on the risk assessment	**CertifyIt for Security Testing** (developed by Smartesting) **RACOMAT** risk assessment and testing tool (developed by Fraunhofer)
Security Testing Tool (STET) for adapting the test item and executing test cases	**CertifyIt for Security Testing** (developed by Smartesting) **RACOMAT** risk assessment and testing tool (developed by Fraunhofer)
Security Test Management Tool and Test Result Aggregation Tool (STRAT) for supporting the aggregation of security test results into a format that allows us to verify the risk picture and to update the risk picture based on the results	**RACOMAT** risk assessment and testing tool (developed by Fraunhofer) **CORAS risk monitor prototype** (developed by SINTEF), which defines rules for updating the risk assessment at run-time **ARIS Business Architect** providing the risk aggregation and risk rating functionality (developed by Software AG)

Table 1 – Conceptual RASEN tools

Figure 1 shows the direct relationship between the conceptual RASEN tools and the concrete RASEN tools. The conceptual RASEN tools are represented by UML interfaces and the concrete RASEN tools are specified by UML classes. We consider that the integration interfaces for data exchange are defined on the level of the RASEN conceptual tools, thus by means by the UML interfaces that represent these tools. Thus, we can provide a flexible mapping between the conceptual RASEN tools and the concrete RASEN tools by considering that each concrete RASEN tool that takes the role of a conceptual RASEN tool has to formally implement the interface that represents the conceptual RASEN tool.

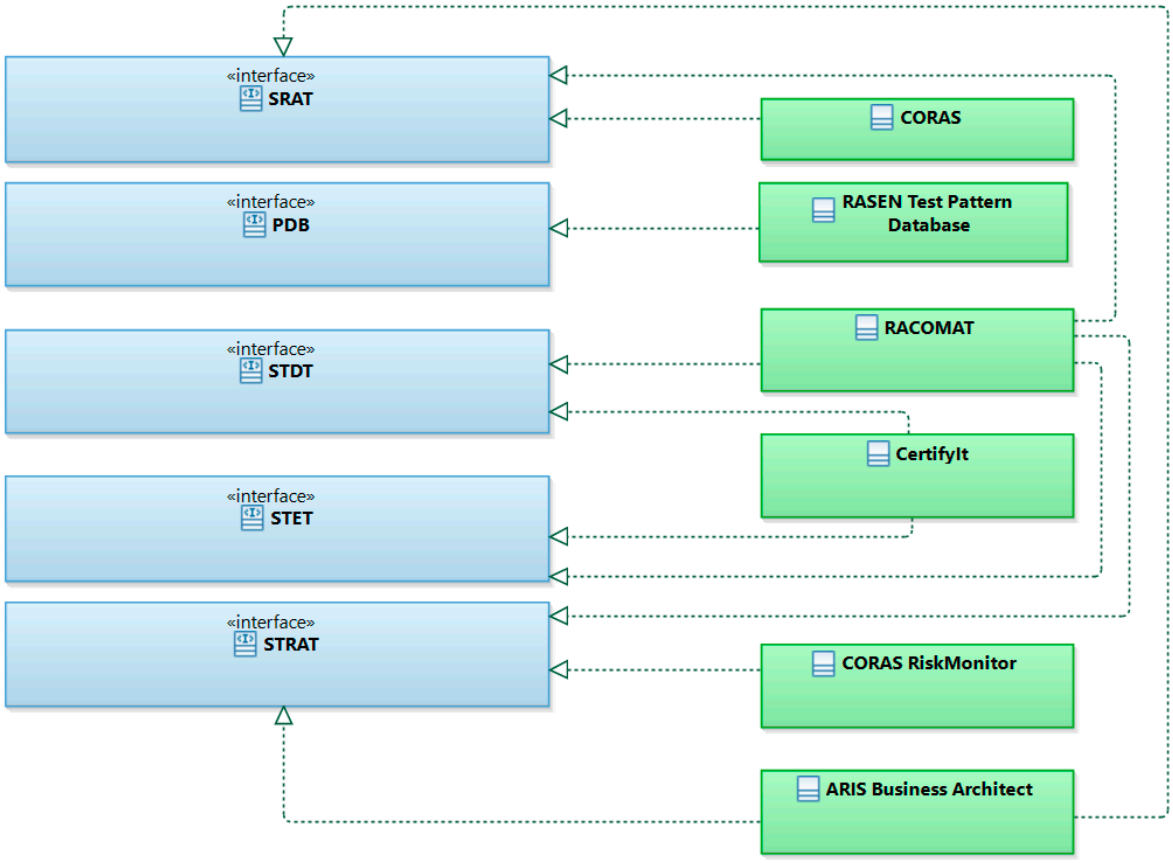

Figure 1 – UML diagram modeling the relationship between conceptual and concrete tools

3 Update of the RASEN Data Integration Models

The RASEN Data Integration has been initially defined in RASEN Deliverable D5.4.1. During the RASEN project, we have simplified the model and adapted it to the current project requirements. The current state of the model is explained in the following sections.

3.1 Update of the Foundation Package

The classes of the foundation package have been defined to reduce redundancy in the specification of the other packages in the data model. Most classes defined in the other packages extend an element of the foundation package, thereby inheriting attributes and relations, which are common for many classes.

The foundation package is shown in Figure 2. The main class of the foundation package is *Element.* This class acts as super type to most of the classes defined elsewhere in the data model. The Element class has three attributes that allow for providing a unique identifier (attribute identifier), a descriptive user defined name (attribute name), and a user defined description (attribute description)

To allow for the expression of typed and parameterized elements, we have introduced the classes *Type* and *Parameter* which are referenced by the Element class. Concrete type values and parameter values can be expressed by using the class *ParameterValue* and *ValueElement*, respectively.

Custom types may be defined through the use of the *CustomType* class. This is for instance needed in order to describe likelihood and consequence scales (in the risk assessment domain) and to distinguish different kinds of likelihood values such as probability, frequencies, intervals of these and properties of likelihoods such as mutual exclusiveness and statistical independence.

The main update of the foundation package w.r.t. to the previous version described in deliverable D5.4.1, is the introduction of the *Parameter* class. This class allows us to express the data model in a more general way. For instance, in the risk assessment domain, we previously had specific data model classes for common concepts of the risk domain such as risk, threat, and so on. This had the potential problem of making the data model too static and specific to a particular terminology for expressing concepts in the risk domain. With the introduction of the *Parameter* class, we are now able express risk domain concepts in the data model as opposed to the meta model level.

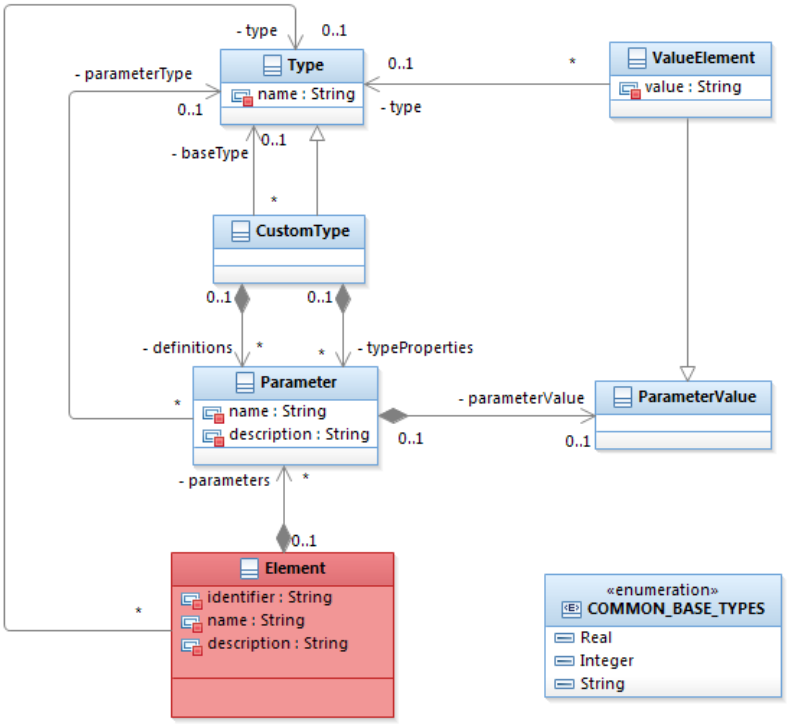

Figure 2 – Structure of the foundation package

3.2 Update of the Risk Assessment Package

The classes of the risk assessment domain are shown in Figure 3. The classes of this domain allow for the expression of risk graphs whose nodes and edges may be of various kinds. The top container class is *RiskManagementModel*, which can contain a set of risk models expressed by *RiskModel*, which in turn may contain a set of risk elements (*RiskElement)* and risk functions (*RiskFunction*). The risk elements may be of one of two kinds: *RiskNode* and *RiskRelation*. These classes allow us to express risk graphs.

The main difference between the risk domain package shown in Figure 3 and the risk domain package described in the previous RASEN deliverable D5.4.1, is that *RiskNode* and *RiskRelation* have not in this data format been specialized further into concepts that are specific to the risk domain such as Threat, Unwanted Incident, Risk, etc. Instead, this information is now intended to be expressed as a parameter value of the risk node and risk relation classes.

The enumerations *COMMON_RISK_PARAMTER_NAMES* and *COMMON_RISK_TYPE_VALUES* have been introduced into the model to express parameter names and types that are common within the risk assessment domain.

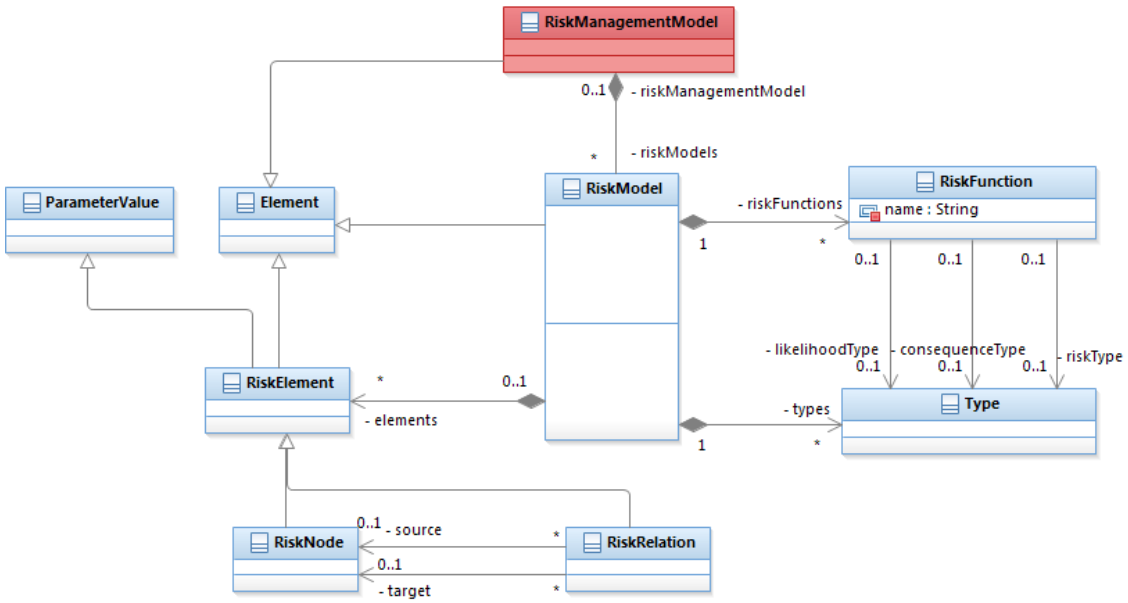

Figure 3 – Risk Management Model structure

3.3 Update of the Testing Package

The models of the testing domain have been simplified by concentrating of the two main data exchange use case: The export/import of predefined test pattern and the export/import of test results in form of a test report. The data for both use cases are described in the following two subsections. The main difference of the testing package described here w.r.t. to the previous version described in deliverable D5.4.1, is the role of the concept test pattern. While a test pattern in deliverable D5.4.1 has been defined as a part of a test plan, it has a more prominent role in this version and is now one of the major concepts for data exchange.

3.3.1 Test Pattern

A test pattern is a test design draft to generate a set of test cases addressing certain vulnerabilities or a certain category of vulnerabilities (see also RASEN Deliverables D4.2.1 and D4.2.2). A test pattern generally provides information that is relevant for (semi-)automatic test case generation. The test design technique identifies a particular method for test case generation and test strategies specify in which way the test design technique shall be applied in order to generate test cases. Test strategies can be implemented by test case generators.

Additionally, the effort of testing for such a vulnerability as well as the effectiveness are recorded. These are for a solution that estimate the effort for testing using the specified solution, i.e. the manual effort, and the effectiveness, i.e. how likely it is to find a vulnerability using the described solution. The specification of security testing metrics allows aggregating test results and estimating the exploitability of a revealed vulnerability.

The following model describes the general structure of a test pattern. The class *TestPattern* is associated with *TestStrategies* for stimulation and observation. Each *TestStrategy* has a number of *TestStrategyParameters* to describe the input, output and return values, and a *TestStrategyProcesses* as a concrete description of how the test pattern could be applied. The latter can contain code fragments in different programming languages.

The attributes values *Effort* and *Effectiveness* provide parameters for test prioritization. The class *TestPatternReferences* is used to depict references to public databases that provide additional information like security attack pattern (e.g. MITRE CAPEC) or for known weaknesses (e.g. MITRE CWE or OWASP).

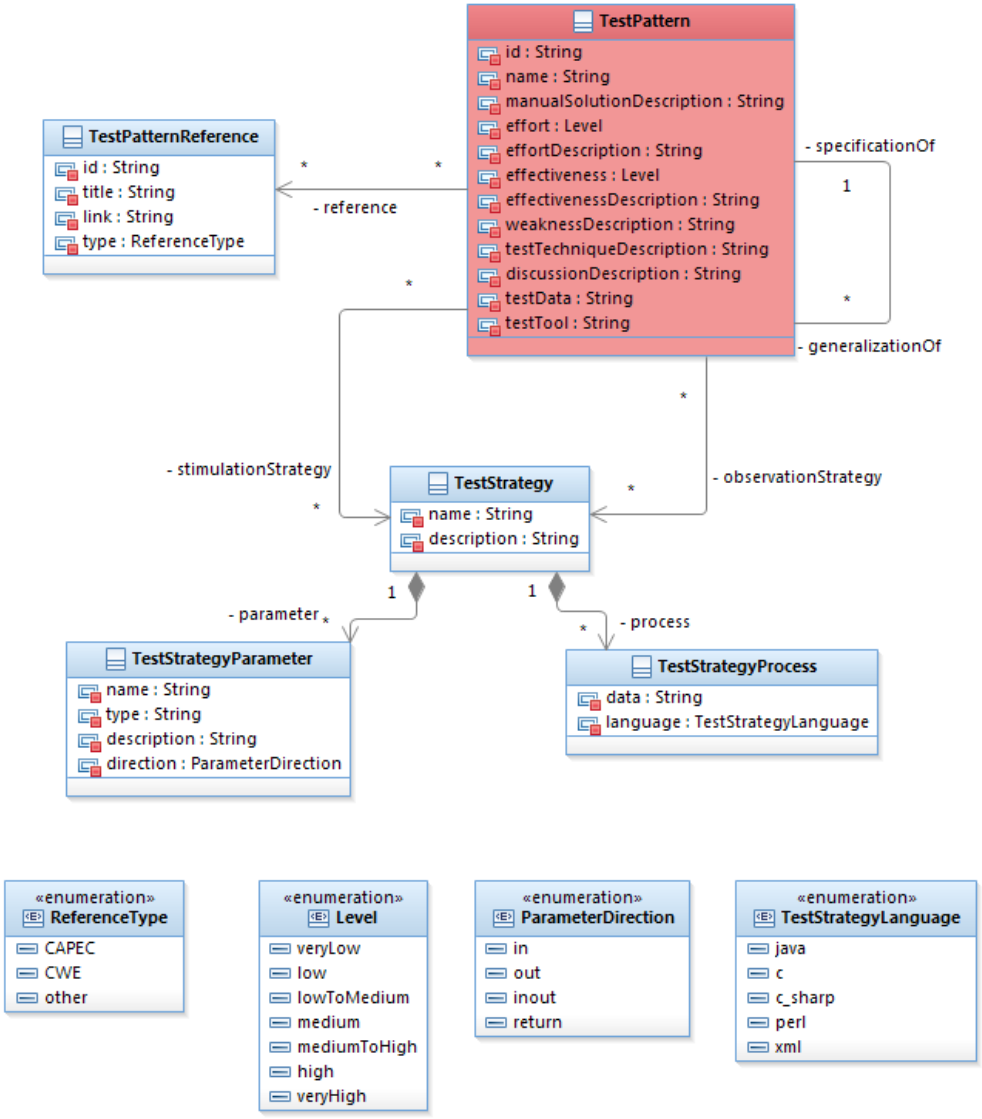

Figure 4 – Test Pattern structure

3.3.2 Test Report

A *TestReport* is a document that, according to ISO/IEC/IEEE 29119, contains the *TestLog* and the *TestIncidentReport.* A *TestReport is* associated with a *TestItem.* Each *TestLog* contains the *TestResults* of a test run. The *TestResult* is qualified by its *verdict* attribute that may evaluate to *none, pass, inconclusive, error* or *fail*. Each *TestIncidentReport* contains the *TestIncidents* of a test run.

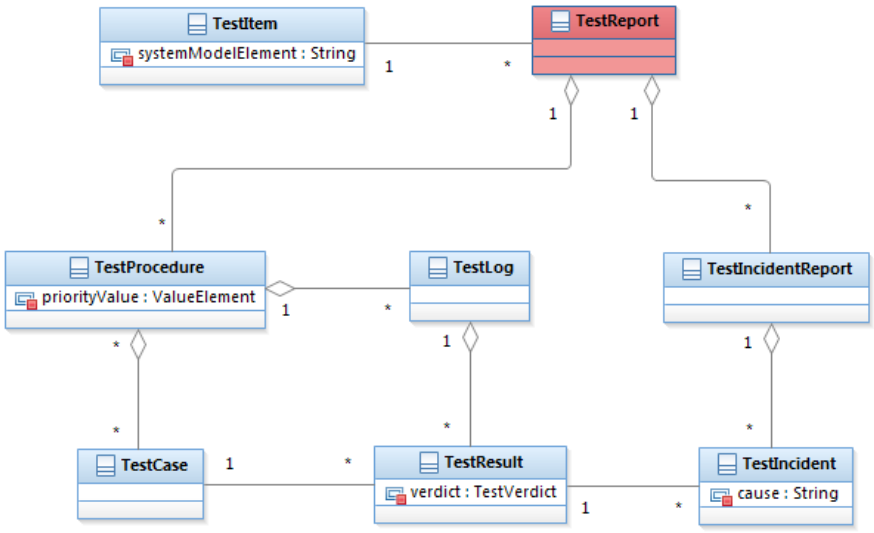

Figure 5 – Test Report structure

3.4 Update for the Relation between Risk Assessment and Testing

The Test Pattern supported Risk Based Security Testing methodology in D5.3.1 describes test pattern as a central element of the integration of risk assessment and testing. This relationship is depicted in Figure 6 by the associations between the *TestPattern* class and the *RiskElement* class.

The *TestProcedure* is the other main class used for linking the risk assessment to the testing domain. In Figure 6 this is expressed by the association between the *TestProcedure* class from the testing domain and the *RiskElement* class from the risk assessment domain.

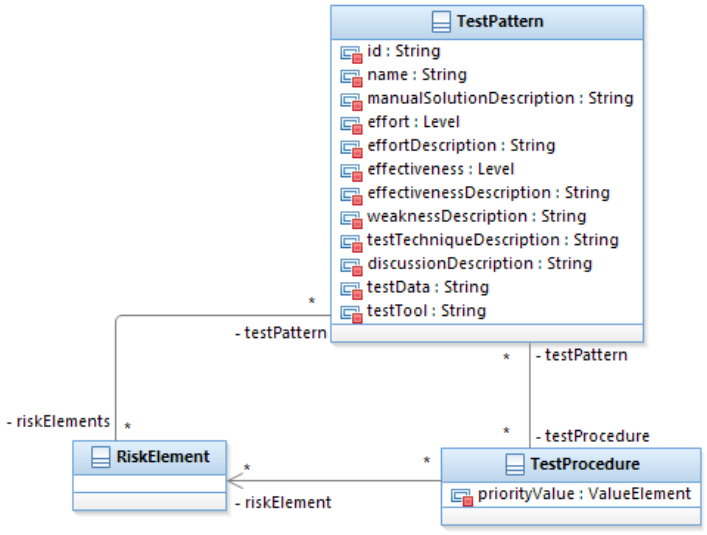

Figure 6 – Relation between risk assessment and testing

4 Update of the Integration Interfaces

This section provides updates of the integration interfaces and their implementation by the conceptual and concrete RASEN tools. The integration interfaces define the data that are exported and imported by the individual tools. The data are expressed by means of the models defined in Section 3.

Conceptual Tool	**Security Risk Assessment Tool (SRAT)**
Concrete Tools	CORAS, RACOMAT, ARIS Business Architect
Requires	TestReport (see Section 3.3.2)
Provides	RiskModel (see Section 3.2)

Table 2 – Security Risk Assessment Tool (SRAT)

Conceptual Tool	**Security Pattern Data Base (PDB)**
Concrete Tools	RASEN Test Pattern Database
Requires	
Provides	TestPattern (see Section 3.3.1)

Table 3 – Security Pattern Data Base (PDB)

Conceptual Tool	**Security Test Derivation Tool (STDT)**
Concrete Tools	Smartesting CertifyIt [5] RACOMAT
Requires	RiskModel (see Section 3.2) TestPattern (see Section 3.3.1)
Provides	

Table 4 – Security Test Derivation Tool (STDT)

Conceptual Tool	**Security Testing Tool (STET)**
Concrete Tools	Smartesting CertifyIt [5] RACOMAT
Requires	
Provides	TestReport (see Section 3.3.2)

Table 5 – Security Testing Tool (STET)

Conceptual Tool	**Security Test Management Tool and Test Result Aggregation Tool (STRAT)**
Concrete Tools	RACOMAT + RISKTest [1] CORAS Risk Monitor ARIS Business Architect
Requires	TestReport RiskModel (see Section 3.2)
Provides	TestReport (see Section 3.3.2)

Table 6 – Security Test Management Tool and Test Result Aggregation Tool (STRAT)

The formal UML specification that is given in Figure 7 contains an *import* function for each entry in the required rows in the tables above and an *export* function for each entry in the provided rows in the tables above.

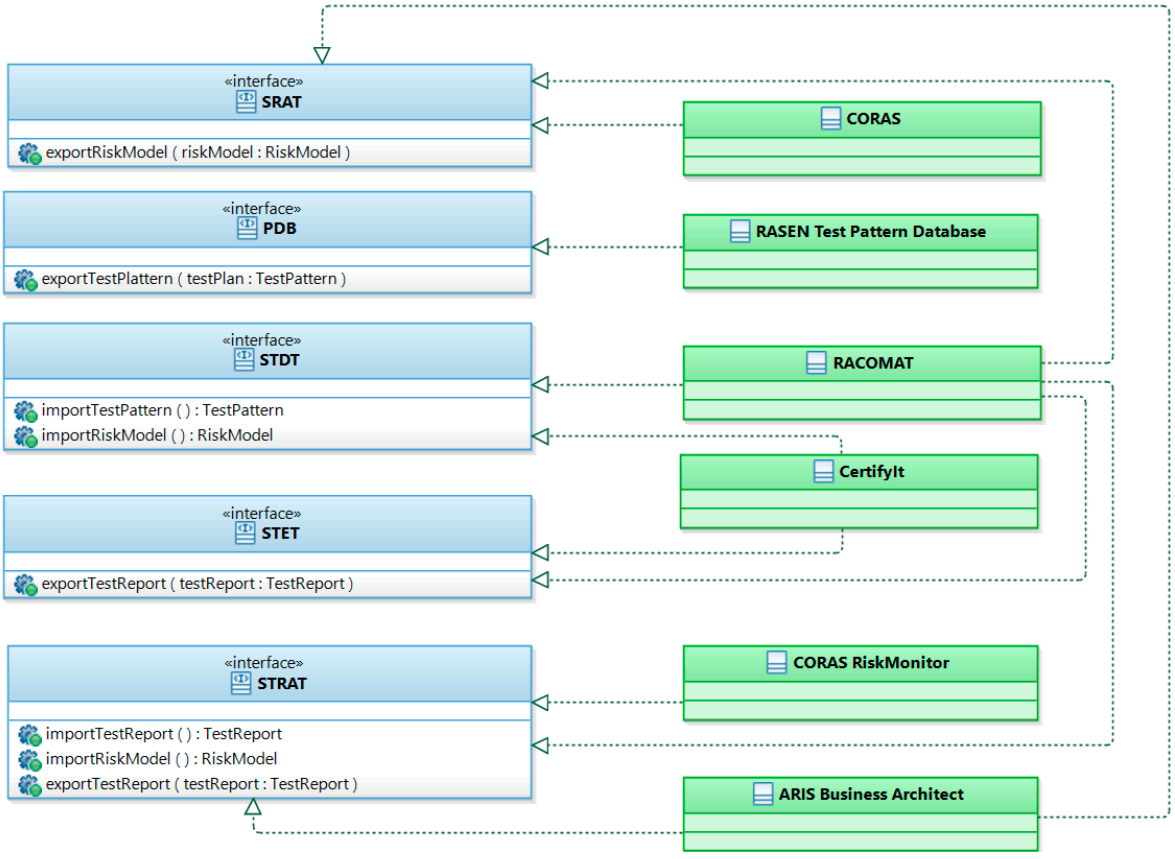

Figure 7 – UML diagram modeling the interfaces for data exchange

5 The RASEN Data Exchange Format and its Application

The RASEN Data Exchange Format aims for simple but effective data exchange between the major tools that are deployed within the RASEN toolbox. The format has been specified by and derived from the RASEN Data Integration Models defined in Section 3. The following subsections describe the principles of the RASEN Data Exchange Format and introduce an Eclipse-based infrastructure that allows for an easy set up and realization of software based adapters. These adapters allow transformation of internal data structures handled by the individual RASEN tools to the data structures that are defined in the RASEN Data Export Format and vice versa.

5.1 Principles of the RASEN Data Exchange Format

The RASEN project aims to provide the data exchange facilities between the major conceptual tools to establish support for risk-based security testing and test based risk assessment. We distinguish the RASEN Data Exchange Format itself from an adapter infrastructure that allows realizing individual tool adapters. While the RASEN Data Exchange Format provides a set of data structures that are necessary to support dedicated integration use cases (e.g. the exchange of risk model, the exchange of test results), the adapters provide services to ease the transformation between the internal data representation of the tools and the structures that are defined by the RASEN Data Exchange Format and thus allow to export/import instances of the RASEN Data Exchange Format. The RASEN Data Exchange Format has three different subsets:

- A risk model data exchange format to propagate the security risk assessment information,
- A test pattern data exchange format to set-up a test pattern database and to export sets of or individual security test pattern, and
- A test report data exchange format to propagate test results.

All formats are specified by means of an individual XML schema. The XML schema is documented in the Appendix A to C.

Figure 8 shows how the data exchange between tools is specified. Each of the subsets of the RASEN Data Exchange Format fulfils and supports a distinct integration use case that has been defined in Deliverable D5.2.1.

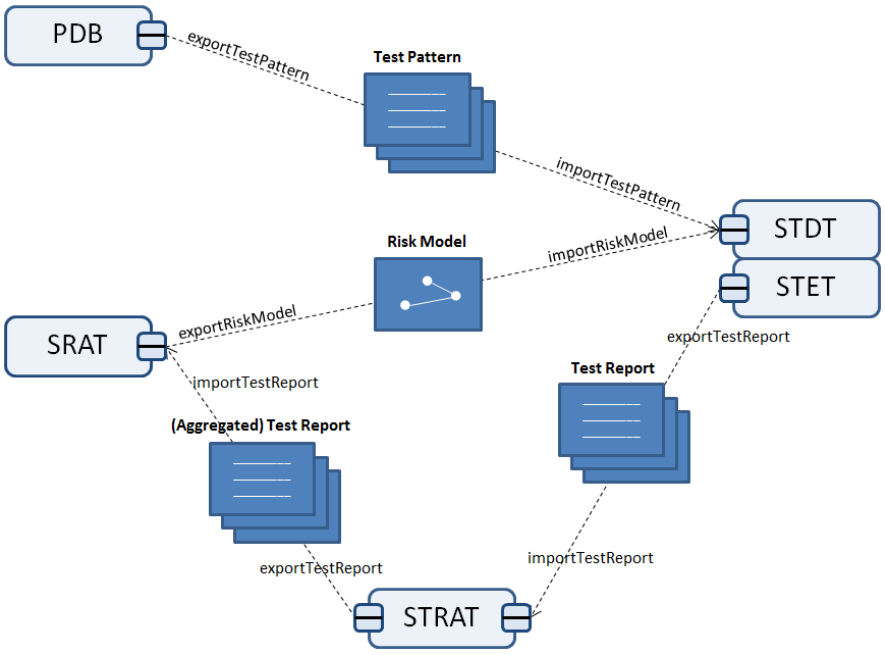

Figure 8 – Overview of data exchange

The export/import adapters need to be implemented for each tool individually. The ease the implementation process, the adapters could be partially generated on basis of the given XML schemas. The following sections describe the process of generating an adapter stub.

5.2 How to Create an Adapter

This section provides a short how-to that explains how an adapter stub for the RASEN Data Exchange Format can be generated. The process is based on the generation facilities of the Eclipse EMF Framework. To ease the process we provide an exemplary Eclipse workspace that already contains the individual XML schemas for the RASEN Data Exchange Format in a preconfigured environment. In the following, we provide a step-by-step explanation that introduces the example workspace, explains the structure of an adapter and explains the overall generation process of the adapter stub by an example and on basis of the workspace.

5.2.1 The Eclipse Example Workspace

To ease the overall generation process we have set up an Eclipse example workspace that can be used as a basis of the activities that are described in the following. The workspace can be used as a starting point for the generation and development process for export/import adapters.

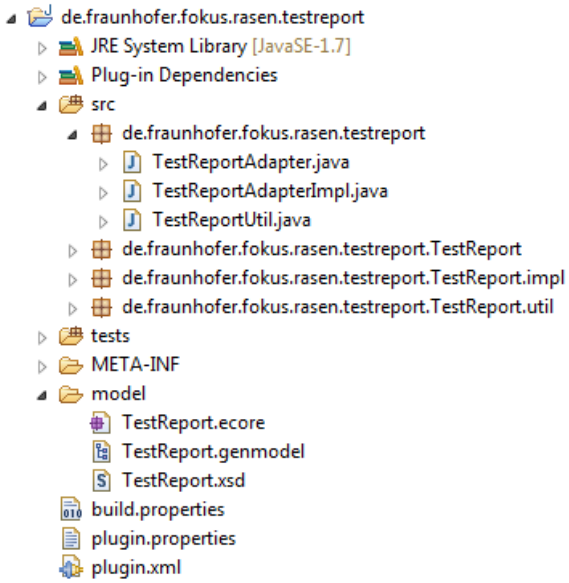

Figure 9 – File structure of the example project

The workspace consists of three Eclipse projects. Each project shows a directory *./model* that contains one of the RASEN Data Exchange Format schemas. We have additionally defined a fully fledged example project that already provides all generation artifacts for the RASEN test report schema. The project is called *de.fraunhofer.fokus.rasen.testreport* and shows an adapter stub for the RASEN test report schema called *TestReport.xsd*. In addition to the other projects it further contains the generated artefacts *TestReport.genmodel* (the EMF generation model) and *TestReport.ecore* (the EMF model) and additionally shows a source folder that contains the generated EMF source files. The sources are structured as generated by the EMF framework i.e. the interface specification can be found in the base package (*de.fraunhofer.fokus.rasen.testreport*), the implementation in the *.impl* sub-package and additionally utility classes in the *.util* sub-package (for detailed description on how to generate the EMF model from the XML schema file, please refer to Section 5.2.3 and to public available EMF documentations e.g. [6]).

In addition to the generated artifacts, the example project already contains an exemplary but ready to use test report adapter. The adapter realizes the export/import functionality for a test report as specified in Appendix B. It can be used as a template for the set up and development of other adapters and yields as an example on how to structure the export/import functionality. In general the location of the sources of the actual export/import adapter can be freely chosen. In the illustrated

example the package is named *de.fraunhofer.fokus.rasen.testreport* and is also located in the already existing source folder. The workspace and the example project additionally contain instances of the test report schema (e.g. test cases with test results and incident reports) so that the Test Report Adapter can be tested with valid export/import data.

5.2.2 The Structure of an Export/Import Adapter

The adapter consists of the test report adapter interface specification (TestReportAdapter.java), the implementation of the interfaces (TestReportImpl.java) and a utility class called *TestReportUtil.* The latter directly supports the export, import and generation of a test report model instances.

The overall structure of an export/import adapter is shown in Figure 10. The adapter has an import and an export method to read and write test reports that conform to the RASEN test report schema.

```
/**
 * The test report adapter is responsible for import and export test reports.
 * Test reports are used to exchange the test report data and are formatted
 * according to the test report exchange format specification from
 * deliverable D5.4.2 (see schema file TestReport.xsd). The adapter
 * generates a test report instance with importing from an XML document and
 * export a test report instance to a URL location in an XML document.
 */
public interface TestReportAdapter {

    /**
     * Imports the test report model from a specified URL and generates a test
     * report model instance. The source is an XML document according to the test
     * report exchange format specification (from deliverable D5.4.2).
     *
     * @param source URL
     * @return the accordingly test report.
     * @throws IOException if the source is not reachable or invalid for importing.
     */
    TestReport importTestReport (URL source) throws IOException;

    /**
     * Exports the test report model instance to a specified URL.
     * The exported test report is an XML document according to the test report
     * exchange format specification (from deliverable D5.4.2).
     *
     * @param source model instance
     * @param target URL
     * @throws IOException if the target is not reachable for exporting.
     */
    void exportTestReport (TestReport source, URL target) throws IOException;
```

Figure 10 – Interface of the test report adapter

The test tool provider (i.e. the developer of an adapter) must realize the actual mapping from the data structures of the RASEN test report schema to the internal data structures of a particular RASEN tool. A dedicated utility class will provide an interface to actually realize this functionality of importing and exporting XML based test reports and of generating the test report model instance.

The generated EMF utility classes and the standard Eclipse XML modeling framework can be used to read the XML documents in a standardized procedure. The class *TestReportUtil* provides the import and export methods that realize a standardized file import (see Figure 11) and export (see Figure 12).

```java
/**
 * Import a test report from an XML document and get the
 * test report model instance.
 *
 * @param source
 * @return the test report model, or null if no test
 *   report was found.
 * @throws IOException
 */
public static TestReport importTestReport(URL source)
        throws IOException {
```

Figure 11 – Class TestReportUtil - import test report

```java
/**
 * Export a test report instance into an XML document.
 * The target will be overwrite if it already exists.
 *
 * @param source
 * @param target
 * @throws IOException
 */
public static void exportTestReport(TestReport source, URL target)
        throws IOException {
```

Figure 12 – Class TestReportUtil - export test report

To realize the export of a test report we require a test report model instance that can be automatically converted in its XML representation by means of the Eclipse EMF framework. Usually, the tool specific data are normally in a proprietary format, so that they must be converted to match with the classes of the test report model instance. Normally, one has to build a test report model instance from scratch to achieve this. EMF generated models do provide generated factory class for creating the individual model instance objects. Since the *TestReportUtil* class provides a set of tailored factory classes, the developer need not use the EMF factory classes directly but can make use of the specialized factory methods provided by the adapter (see Figure 13 and Figure 14).

```java
/**
 * Create a test report. The test report is empty without referenced source
 * and URI information.
 *
 * @return
 */
public static TestReport createTestReport() {
    return TestReportFactory.eINSTANCE.createTestReport();
}
```

Figure 13 – Class TestReportUtil - create element TestReport

```
/**
 * Create a test item element with all parameters
 * and relate the element to a report.
 *
 * @param report
 * @param name
 * @param id
 * @param description
 * @param modelElement
 */
public void createTestItem(TestReport report, String name,
        String id, String description, String modelElement) {

    TestItem item = TestReportFactory.eINSTANCE.createTestItem();

    item.setName(name);
    item.setIdentifier(id);
    item.setDescription(description);
    item.setSystemModelElement(modelElement);

    report.setItem(item);
}
```

Figure 14 – Class TestReportUtil - create element TestItem

In the following, we illustrate the creation of test report model instances by means of two examples.

5.2.2.1 Example 1: Test report is available as plain text in proprietary format

Let's consider the test tool report consists of the four textual elements: test cases, test items, test results and incident reports. The elements are given in form of a comma-separated list. Each line starts with the element type, which is followed by a list of attributes. An example line for a test case element with the attributes name, id and comment looks like this:

"*TestCase, testRandomLoginSuccess, 277ecc41-3f8e-4ad4-9df5-37d5fe3d1442, test login with a random admin password*"

The test report adapter has to read the test report file, parse each line, check the type and the arguments and create a corresponding element of the test report model:

```
for (Line line: readLines(document)) {
 switch (line.getType) {
  case(TESTCASE): util.createTestCase(report,line.getName(),line.getId(), line.getDescription()); break;
  case(TESTRESULT): ...
...
}
```

Figure 15 – Example Code Fragment of Parsing a Propietary Test Report Document

After having created the model instance, the adapter can export the test report model in form of an XML file that conforms to the schema given in Appendix B.

5.2.2.2 Example 2: Test format is available in proprietary XML format

Let's consider the test tool already has XML documents for its test reports but these documents are not conforming to the RASEN test report schema. Let's also consider the test tool provides a schema for this XML document format (If not, the free and open source eclipse plugin Rinzo XML1 can be used to generate a schema). Following the description in 5.2.3, it is possible to realize an importer for the proprietary XML format so that there is a model instance representing the test report in the

[1] http://editorxml.sourceforge.net/

proprietary format. Finally, the test report adapter should be able to import the proprietary test report. The imported test report instance can be used to generate a model instance of the exchange format. Example code for such a transformation is given below.

```
TestReport getExchangeFormatedTestReport(ProprietaryFormatedTestReport propietaryReport) {
  TestReport report = util.createTestReport();
  util.createTestItem(report, propietaryReport.getItem());
  for (ProprietaryTestCase tc : propietaryReport.getTestcases()) {
    util.createTestCase(report, tc.getName(), tc.getId(), tc.getDesciption());
  }
  ...
```

Figure 16 – Example Code Fragment of Parsing a Propietary Test Report Model

After having created the model instance, the adapter can also export the test report model in form of an XML file that conforms to the schema given in Appendix B.

5.2.3 The Generation Process Step by Step

In the following we describe the generation process for an adapter stub that allows the export/import of instances of the RASEN Data Exchange Format. The process is described by means of the Test Report format that is handled by a Test Report Adapter.

The generation process requires Java, the Eclipse environment and the Eclipse EMF framework that is shipped with the Eclipse modelling tools.

As a first step of the generation process we create a new Java project (e.g. *de.fraunhofer.fokus.rasen.TestReport*) in our example workspace. We copy the XSD schema file (i.e. TestReport.xsd) into a subfolder of this project (e.g. *de.fraunhofer.fokus.rasen.TestReport /model*). We start the generation process by selecting the Test Report schema file in the Project Explorer window of our Eclipse environment. In the context menu we choose the creation of a new ***EMF Generator Model*** as depicted in Figure 17 and Figure 18 (first press New-> Other and then select EMF Generator Model under Eclipse Modelling Framework).

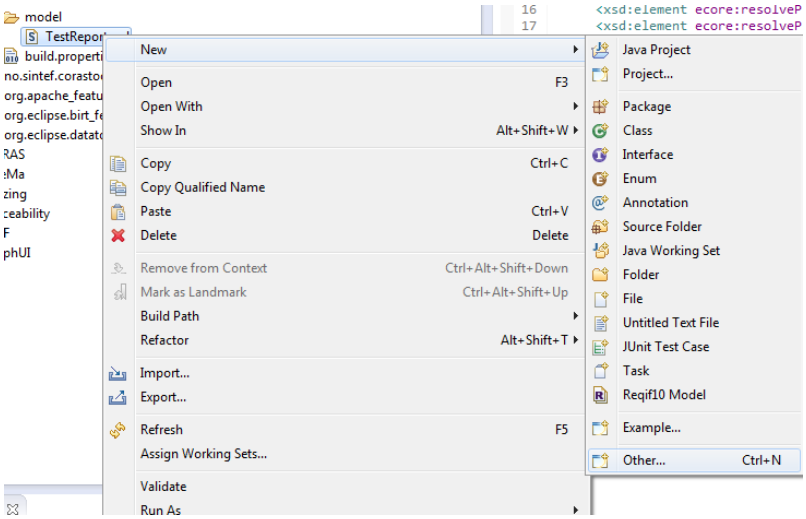

Figure 17 – Create data exchange adapter - generate Genmodel

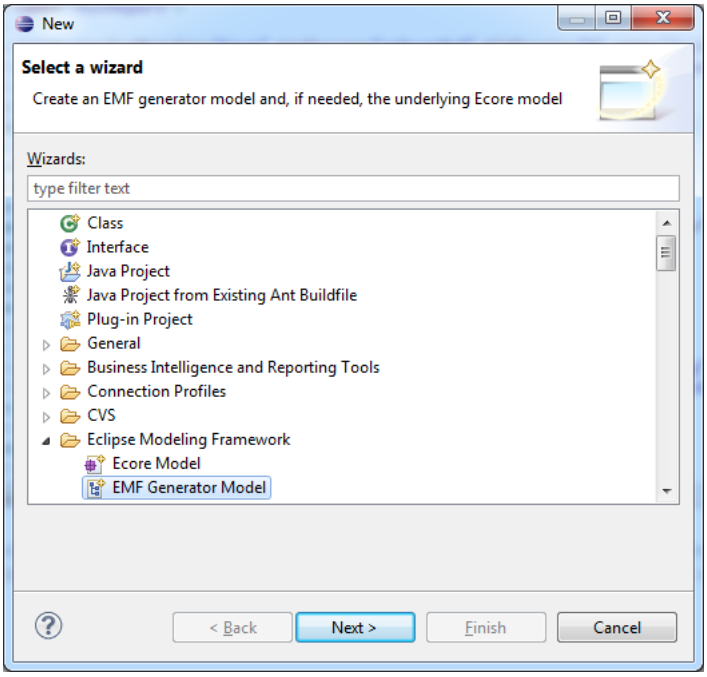

Figure 18 – Create data exchange adapter - generate Genmodel (2)

Afterwards, we select ***XML Schema*** as ***Model Importer*** as shown in Figure 19.

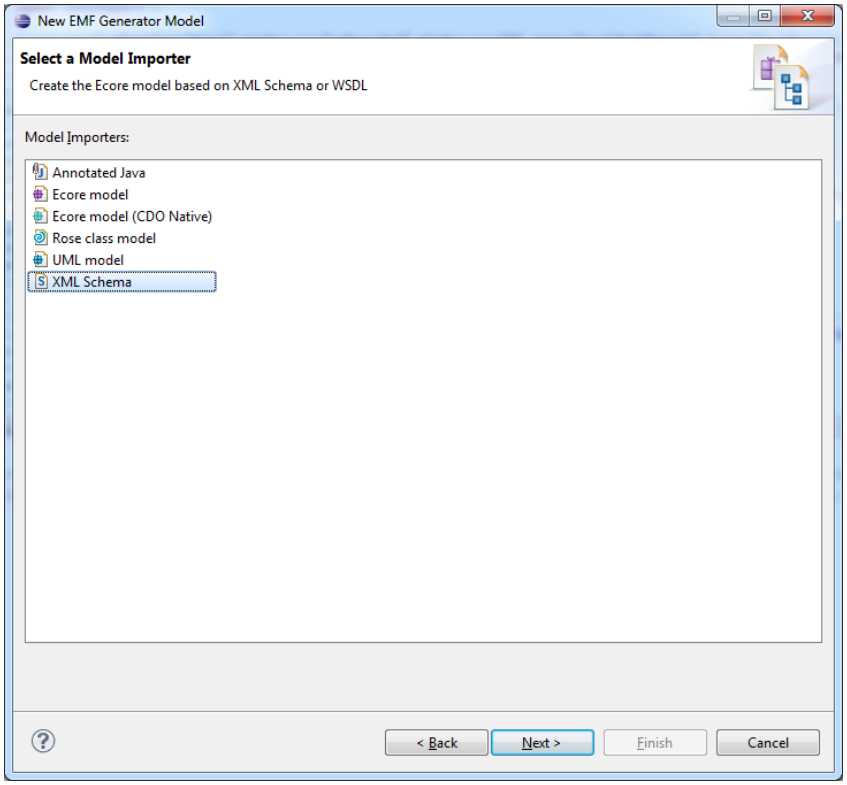

Figure 19 – Create data exchange adapter - generate Genmodel (3)

In the following, the Package Selection allows for setting the individual parameters of the generation process. In our case, we set the test report package (*de.fraunhofer.fokus.rasen.TestReport*) as root package and the Ecore (*org.eclipse.emf.ecore*) package as a referenced generator model (see Figure 20).

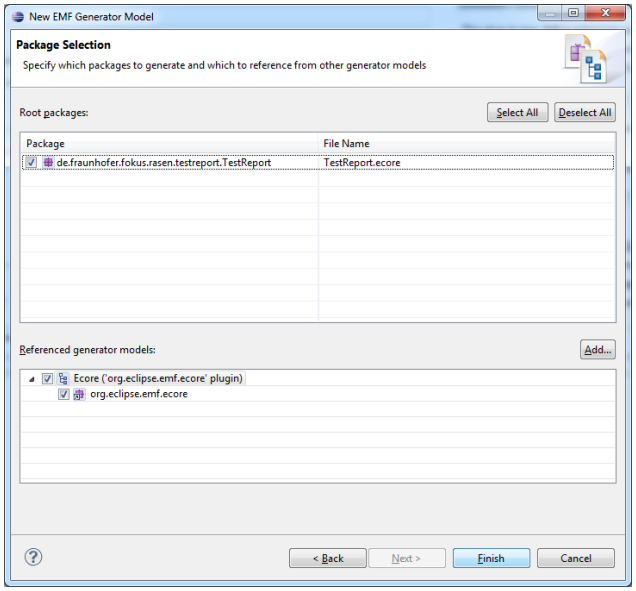

Figure 20 – Create data exchange adapter - generate Genmodel (4)

After pressing the finish button, the Eclipse generates the generator file <name> in the same directory where the TestReport schema is already located (e.g. *de.fraunhofer.fokus.rasen.TestReport/model*). We can now generate the model code and all corresponding factory classes for the Test Report adapter by selecting the generator file and choose the option ***Generate Model Code*** for the model as shown in Figure 21.

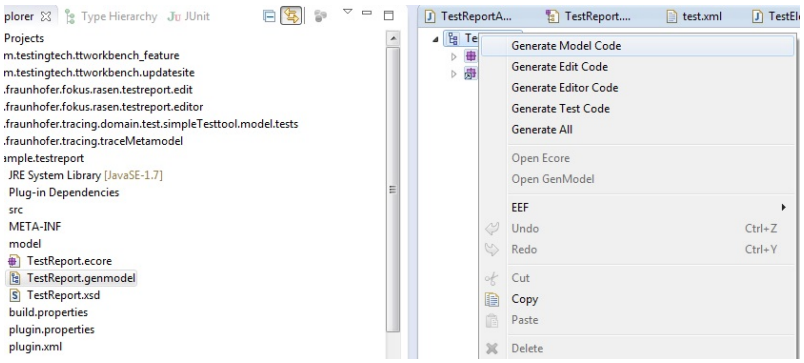

Figure 21 – Create data exchange adapter - generate Data Model

The generator generates the java classes into the source folder of the project. The result is shown in Figure 22.

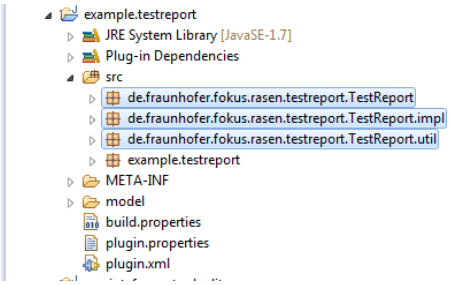

Figure 22 – Create data exchange adapter - generate Data Model (2)

Finally, we have generated all elements of the Test Report format as java class and are able to use and extend them as described in Section 5.2.2.

6 Summary and Outlook

This deliverable specifies the RASEN Data Exchange Format on basis of the RASEN Data Integration Model. The Data Integration Model has been derived from the RASEN Conceptual Model, introduced in D5.3.1. The RASEN Data Exchange Format addresses the integration between risk assessment and testing and depicts the technical aspects that are necessary for feasible data exchange between tools. In addition to the XML schemata, this report contains guidance on how to realize export/import adapters for the RASEN tools.

As next step we will develop the adapters for the individual RASEN tools (i.e. bullet 5 in the overall activity list, see Section 1). This will enable the export/import of risk models, test pattern and test reports and thus establish the RASEN Data Exchange Format as basis for tool interoperability in RASEN.

References

[1] FhG FOKUS: Fokus!MBT + RISKTest+ RACOMAT, platform for model and risk based testing from FhG FOKUS (2013)

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

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

[4] SINTEF: CORAS Risk Assessment Tool from SINTEF (2013)

[5] Smartesting: CertifyIt, tool suite for model based testing (2013)

[6] Eclipse Foundation: Generating an EMF Model using XML Schema (XSD), http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.emf.doc%2Ftutorials%2Fxlibmod%2Fxlibmod.html

Appendix A: Risk Model Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns:RiskModel="http://RiskModel"
xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
xmlns:foundation="http://foundation" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
ecore:nsPrefix="RiskModel"
ecore:package="de.fraunhofer.fokus.rasen.dataexchange.RiskModel"
targetNamespace="http://RiskModel">
  <xsd:import namespace="http://www.eclipse.org/emf/2002/Ecore"
schemaLocation="platform:/plugin/org.eclipse.emf.ecore/model/Ecore.xsd"/>
  <xsd:import namespace="http://foundation" schemaLocation="foundation.xsd"/>
  <xsd:simpleType ecore:name="COMMON_RISK_PARAMETER_NAMES"
name="COMMON_RISK_PARAMETER_NAMES">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Likelihood"/>
      <xsd:enumeration value="Consequence"/>
      <xsd:enumeration value="RiskValue"/>
      <xsd:enumeration value="RiskNodeType"/>
      <xsd:enumeration value="RiskRelationType"/>
      <xsd:enumeration value="Vulnerability"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType ecore:name="COMMON_RISKNODE_TYPE_VALUES"
name="COMMON_RISKNODE_TYPE_VALUES">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Threat"/>
      <xsd:enumeration value="UnwantedIncident"/>
      <xsd:enumeration value="ThreatScenario"/>
      <xsd:enumeration value="Asset"/>
      <xsd:enumeration value="Event"/>
      <xsd:enumeration value="EventSource"/>
      <xsd:enumeration value="Risk"/>
      <xsd:enumeration value="Vulnerability"/>
      <xsd:enumeration value="Treatment"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType ecore:implements="foundation:ParameterValue"
name="RiskElement">
    <xsd:complexContent>
      <xsd:extension base="foundation:Element">
        <xsd:attribute ecore:name="testPatterns"
ecore:reference="RiskModel:TestPattern" name="testPattern">
          <xsd:simpleType>
            <xsd:list itemType="xsd:anyURI"/>
          </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute ecore:name="testProcedures"
ecore:reference="RiskModel:TestProcedure" name="testProcedure">
          <xsd:simpleType>
            <xsd:list itemType="xsd:anyURI"/>
          </xsd:simpleType>
        </xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="RiskFunction">
    <xsd:attribute ecore:reference="foundation:Type" name="consequenceType"
type="xsd:anyURI"/>
    <xsd:attribute ecore:reference="foundation:Type" name="likelihoodType"
type="xsd:anyURI"/>
    <xsd:attribute name="name" type="ecore:EString"/>
    <xsd:attribute ecore:reference="foundation:Type" name="riskType"
type="xsd:anyURI"/>
  </xsd:complexType>
  <xsd:complexType name="RiskManagementModel">
    <xsd:complexContent>
      <xsd:extension base="foundation:Element">
        <xsd:sequence>
```

```
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded"
minOccurs="0" name="riskModels" type="RiskModel:RiskModel"/>
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded"
minOccurs="0" name="testPatterns" type="RiskModel:TestPattern"/>
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded"
minOccurs="0" name="testProcedures" type="RiskModel:TestProcedure"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="RiskModel">
    <xsd:complexContent>
      <xsd:extension base="foundation:Element">
        <xsd:sequence>
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded"
minOccurs="0" name="riskFunctions" type="RiskModel:RiskFunction"/>
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded"
minOccurs="0" name="types" type="foundation:Type"/>
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded"
minOccurs="0" name="elements" type="RiskModel:RiskElement"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="RiskNode">
    <xsd:complexContent>
      <xsd:extension base="RiskModel:RiskElement"/>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="RiskRelation">
    <xsd:complexContent>
      <xsd:extension base="RiskModel:RiskElement">
        <xsd:attribute ecore:reference="RiskModel:RiskNode" name="source"
type="xsd:anyURI"/>
        <xsd:attribute ecore:reference="RiskModel:RiskNode" name="target"
type="xsd:anyURI"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="TestPattern">
    <xsd:complexContent>
      <xsd:extension base="foundation:Element"/>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="TestProcedure">
    <xsd:complexContent>
      <xsd:extension base="foundation:Element">
        <xsd:attribute ecore:reference="foundation:ValueElement"
name="priorityValue" type="xsd:anyURI"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:schema>
```

Appendix B: Test Pattern Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
xmlns:foundation="http://foundation" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
ecore:documentRoot="TestPatternDokument" ecore:nsPrefix="Testpattern"
ecore:package="de.fraunhofer.fokus.rasen.dataexchange.Testpattern">
  <xsd:import namespace="http://foundation" schemaLocation="foundation.xsd"/>
  <xsd:element name="CAPEC_ID" type="xsd:string"/>
  <xsd:element name="CWE_ID" type="xsd:string"/>
  <xsd:element name="Description_Of_Test_Coverage_Items"
type="Description_Of_Test_Coverage_Items_._type"/>
  <xsd:element name="Discussion" type="xsd:string"/>
  <xsd:element name="Effectiveness" type="Level"/>
  <xsd:element name="Effectiveness_Description" type="xsd:string"/>
  <xsd:element name="Effort" type="Level"/>
  <xsd:element name="Effort_Description" type="xsd:string"/>
  <xsd:element name="Generalization_Of" type="xsd:string"/>
  <xsd:element name="Manual_Solution" type="xsd:string"/>
  <xsd:element name="Other" type="xsd:string"/>
  <xsd:element name="Parameter" type="Parameter_._type"/>
  <xsd:element name="Process" type="Process_._type"/>
  <xsd:element name="Reference" type="Reference_._type"/>
  <xsd:element name="References" type="References_._type"/>
  <xsd:element name="Strategy" type="Strategy_._type"/>
  <xsd:element name="Test_Coverage_Item" type="xsd:string"/>
  <xsd:element name="Test_Data" type="xsd:string"/>
  <xsd:element name="Test_Observation_Strategies"
type="Test_Observation_Strategies_._type"/>
  <xsd:element name="Test_Pattern" type="Test_Pattern_._type"/>
  <xsd:element name="Test_Pattern_Catalog" type="Test_Pattern_Catalog_._type"/>
  <xsd:element name="Test_Stimulation_Strategies"
type="Test_Stimulation_Strategies_._type"/>
  <xsd:element name="Test_Technique" type="xsd:string"/>
  <xsd:element name="Test_Tool" type="xsd:string"/>
  <xsd:element name="Weakness_Description" type="xsd:string"/>
  <xsd:complexType ecore:name="DescriptionOfTestCoverageItems"
name="Description_Of_Test_Coverage_Items_._type">
    <xsd:sequence>
      <xsd:element ecore:name="testCoverageItem" maxOccurs="unbounded"
ref="Test_Coverage_Item"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType name="Language">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="c"/>
      <xsd:enumeration value="c_sharp"/>
      <xsd:enumeration value="java"/>
      <xsd:enumeration value="perl"/>
      <xsd:enumeration value="xml"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="Level">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="veryLow"/>
      <xsd:enumeration value="low"/>
      <xsd:enumeration value="lowToMedium"/>
      <xsd:enumeration value="medium"/>
      <xsd:enumeration value="mediumToHigh"/>
      <xsd:enumeration value="high"/>
      <xsd:enumeration value="veryHigh"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType ecore:name="TestStrategyParameter" name="Parameter_._type">
    <xsd:attribute ecore:name="description" name="Description" type="xsd:string"/>
    <xsd:attribute ecore:name="direction" name="Direction"
type="ParameterDirection" use="required"/>
    <xsd:attribute ecore:name="name" name="Name" type="xsd:string"
use="required"/>
    <xsd:attribute ecore:name="type" name="Type" type="xsd:string"/>
```

```
  </xsd:complexType>
  <xsd:simpleType name="ParameterDirection">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="in"/>
      <xsd:enumeration value="inout"/>
      <xsd:enumeration value="out"/>
      <xsd:enumeration value="return"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType ecore:name="TestStrategyImplementation" name="Process_._type">
    <xsd:sequence>
      <xsd:element ecore:name="data" name="Data" type="xsd:string"/>
    </xsd:sequence>
    <xsd:attribute ecore:name="language" name="Language" type="Language"
use="required"/>
  </xsd:complexType>
  <xsd:complexType ecore:name="TestPatternReference" name="Reference_._type">
    <xsd:attribute ecore:name="iD" name="ID" type="xsd:string"/>
    <xsd:attribute ecore:name="referenceLink" name="ReferenceLink"
type="xsd:anyURI"/>
    <xsd:attribute ecore:name="title" name="Title" type="xsd:string"/>
    <xsd:attribute ecore:name="type" name="Type" type="RefType" use="required"/>
  </xsd:complexType>
  <xsd:complexType ecore:name="TestPatternReferences" name="References_._type">
    <xsd:sequence>
      <xsd:element ecore:name="reference" maxOccurs="unbounded" ref="Reference"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:simpleType ecore:name="ReferenceType" name="RefType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="CAPEC"/>
      <xsd:enumeration value="CWE"/>
      <xsd:enumeration value="other"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:complexType ecore:name="TestStrategy" name="Strategy_._type">
    <xsd:sequence>
      <xsd:element ecore:name="parameter" maxOccurs="unbounded" minOccurs="0"
ref="Parameter"/>
      <xsd:element ecore:name="implementation" maxOccurs="unbounded" minOccurs="0"
ref="Process"/>
    </xsd:sequence>
    <xsd:attribute ecore:name="description" name="Description" type="xsd:string"/>
    <xsd:attribute ecore:name="name" name="Name" type="xsd:string"/>
  </xsd:complexType>
  <xsd:complexType ecore:name="TestObservationStrategies"
name="Test_Observation_Strategies_._type">
    <xsd:sequence>
      <xsd:element ecore:name="strategy" maxOccurs="unbounded" minOccurs="0"
ref="Strategy"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType ecore:name="TestStimulationStrategies"
name="Test_Stimulation_Strategies_._type">
    <xsd:sequence>
      <xsd:element ecore:name="strategy" maxOccurs="unbounded" minOccurs="0"
ref="Strategy"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType ecore:name="TestPattern" name="Test_Pattern_._type">
    <xsd:complexContent>
      <xsd:extension base="foundation:Element">
        <xsd:sequence>
          <xsd:element ecore:name="effort" minOccurs="0" ref="Effort"/>
          <xsd:element ecore:name="effectiveness" minOccurs="0"
ref="Effectiveness"/>
          <xsd:element ecore:name="testTechnique" minOccurs="0"
ref="Test_Technique"/>
          <xsd:element ecore:name="weaknessDescription" minOccurs="0"
ref="Weakness_Description"/>
```

```
          <xsd:element ecore:name="manualSolution" minOccurs="0"
ref="Manual_Solution"/>
          <xsd:element ecore:name="effortDescription" minOccurs="0"
ref="Effort_Description"/>
          <xsd:element ecore:name="effectivenessDescription" minOccurs="0"
ref="Effectiveness_Description"/>
          <xsd:element ecore:name="discussion" minOccurs="0" ref="Discussion"/>
          <xsd:element ecore:name="descriptionOfTestCoverageItems" minOccurs="0"
ref="Description_Of_Test_Coverage_Items"/>
          <xsd:element ecore:name="testData" maxOccurs="unbounded" minOccurs="0"
ref="Test_Data"/>
          <xsd:element ecore:name="testTool" maxOccurs="unbounded" minOccurs="0"
ref="Test_Tool"/>
          <xsd:element ecore:name="references" minOccurs="0" ref="References"/>
          <xsd:element ecore:name="testStimulationStrategies" minOccurs="0"
ref="Test_Stimulation_Strategies"/>
          <xsd:element ecore:name="testObservationStrategies" minOccurs="0"
ref="Test_Observation_Strategies"/>
        </xsd:sequence>
        <xsd:attribute ecore:name="generalizationOf"
ecore:opposite="specificationOf" ecore:reference="Testpattern:Test_Pattern_._type"
name="Generalization_Of">
          <xsd:simpleType>
            <xsd:list itemType="xsd:anyURI"/>
          </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute ecore:name="specificationOf"
ecore:opposite="generalizationOf"
ecore:reference="Testpattern:Test_Pattern_._type" name="Specification_Of"
type="xsd:anyURI"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType ecore:name="TestPatternCatalog"
name="Test_Pattern_Catalog_._type">
    <xsd:sequence>
      <xsd:element ecore:name="testPattern" maxOccurs="unbounded" minOccurs="0"
ref="Test_Pattern"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Appendix B: Test Report Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema
xmlns:TestReport="platform:/resource/de.fraunhofer.fokus.rasen.testreport/model/Te
stReport.xsd" xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
xmlns:foundation="http://foundation" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
ecore:nsPrefix="TestReport"
ecore:package="de.fraunhofer.fokus.rasen.dataexchange.TestReport"
targetNamespace="platform:/resource/de.fraunhofer.fokus.rasen.testreport/model/Tes
tReport.xsd">
  <xsd:import namespace="http://www.eclipse.org/emf/2002/Ecore"
schemaLocation="platform:/plugin/org.eclipse.emf.ecore/model/Ecore.xsd"/>
  <xsd:import namespace="http://foundation" schemaLocation="foundation.xsd"/>
  <xsd:complexType name="RiskElement">
    <xsd:complexContent>
      <xsd:extension base="foundation:Element"/>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="TestCase">
    <xsd:complexContent>
      <xsd:extension base="foundation:Element">
        <xsd:attribute ecore:opposite="testCase"
ecore:reference="TestReport:TestResult" name="testResult">
          <xsd:simpleType>
            <xsd:list itemType="xsd:anyURI"/>
          </xsd:simpleType>
        </xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="TestIncident">
    <xsd:complexContent>
      <xsd:extension base="foundation:Element">
        <xsd:attribute name="cause" type="ecore:EString"/>
        <xsd:attribute ecore:opposite="incident"
ecore:reference="TestReport:TestResult" name="testResult" type="xsd:anyURI"
use="required"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="TestIncidentReport">
    <xsd:complexContent>
      <xsd:extension base="foundation:Element">
        <xsd:sequence>
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded"
minOccurs="0" name="incident" type="TestReport:TestIncident"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="TestItem">
    <xsd:complexContent>
      <xsd:extension base="foundation:Element">
        <xsd:attribute name="systemModelElement" type="ecore:EString"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="TestLog">
    <xsd:complexContent>
      <xsd:extension base="foundation:Element">
        <xsd:sequence>
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded"
minOccurs="0" name="testResult" type="TestReport:TestResult"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="TestPattern">
    <xsd:complexContent>
```

```
      <xsd:extension base="foundation:Element"/>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="TestProcedure">
    <xsd:complexContent>
      <xsd:extension base="foundation:Element">
        <xsd:sequence>
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded"
minOccurs="0" name="log" type="TestReport:TestLog"/>
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded"
minOccurs="0" name="testCase" type="TestReport:TestCase"/>
          <xsd:element ecore:resolveProxies="true" minOccurs="0"
name="priorityValue" type="foundation:ValueElement"/>
        </xsd:sequence>
        <xsd:attribute ecore:reference="TestReport:RiskElement"
name="riskElement">
          <xsd:simpleType>
            <xsd:list itemType="xsd:anyURI"/>
          </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute ecore:reference="TestReport:TestPattern"
name="testPattern">
          <xsd:simpleType>
            <xsd:list itemType="xsd:anyURI"/>
          </xsd:simpleType>
        </xsd:attribute>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="TestReport">
    <xsd:sequence>
      <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded" minOccurs="0"
name="incidentReport" type="TestReport:TestIncidentReport"/>
      <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded" minOccurs="0"
name="procedure" type="TestReport:TestProcedure"/>
      <xsd:element ecore:resolveProxies="true" name="item"
type="TestReport:TestItem"/>
      <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded" minOccurs="0"
name="testPatterns" type="TestReport:TestPattern"/>
      <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded" minOccurs="0"
name="riskElements" type="TestReport:RiskElement"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="TestResult">
    <xsd:complexContent>
      <xsd:extension base="foundation:Element">
        <xsd:attribute ecore:opposite="testResult"
ecore:reference="TestReport:TestIncident" name="incident">
          <xsd:simpleType>
            <xsd:list itemType="xsd:anyURI"/>
          </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute ecore:opposite="testResult"
ecore:reference="TestReport:TestCase" name="testCase" type="xsd:anyURI"
use="required"/>
        <xsd:attribute ecore:unsettable="false" name="verdict"
type="TestReport:TestVerdict"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:simpleType name="TestVerdict">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="pass"/>
      <xsd:enumeration value="fail"/>
      <xsd:enumeration value="none"/>
      <xsd:enumeration value="incomplete"/>
      <xsd:enumeration value="error"/>
    </xsd:restriction>
  </xsd:simpleType>
```

```
</xsd:schema>
```

Appendix D: Foundation Schema

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schema xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
xmlns:foundation="http://foundation" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
ecore:nsPrefix="foundation"
ecore:package="de.fraunhofer.fokus.rasen.dataexchange.foundation"
targetNamespace="http://foundation">
  <xsd:import namespace="http://www.eclipse.org/emf/2002/Ecore"
schemaLocation="platform:/plugin/org.eclipse.emf.ecore/model/Ecore.xsd"/>
  <xsd:element ecore:ignore="true" name="Element" type="foundation:Element"/>
  <xsd:element ecore:ignore="true" name="Parameter" type="foundation:Parameter"/>
  <xsd:element ecore:ignore="true" name="ParameterValue"
type="foundation:ParameterValue"/>
  <xsd:element ecore:ignore="true" name="CustomType"
type="foundation:CustomType"/>
  <xsd:element ecore:ignore="true" name="Type" type="foundation:Type"/>
  <xsd:element ecore:ignore="true" name="ValueElement"
type="foundation:ValueElement"/>
  <xsd:complexType name="Element">
    <xsd:sequence>
      <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded" minOccurs="0"
name="parameters" type="foundation:Parameter"/>
    </xsd:sequence>
    <xsd:attribute name="identifier" type="ecore:EString" use="required"/>
    <xsd:attribute name="name" type="ecore:EString"/>
    <xsd:attribute name="description" type="ecore:EString"/>
    <xsd:attribute ecore:reference="foundation:Type" name="type"
type="xsd:anyURI"/>
  </xsd:complexType>
  <xsd:complexType name="Parameter">
    <xsd:sequence>
      <xsd:element ecore:resolveProxies="true" minOccurs="0" name="parameterValue"
type="foundation:ParameterValue"/>
    </xsd:sequence>
    <xsd:attribute name="name" type="ecore:EString"/>
    <xsd:attribute name="description" type="ecore:EString"/>
    <xsd:attribute ecore:reference="foundation:Type" name="parameterType"
type="xsd:anyURI"/>
  </xsd:complexType>
  <xsd:complexType name="ParameterValue"/>
  <xsd:complexType name="CustomType">
    <xsd:complexContent>
      <xsd:extension base="foundation:Type">
        <xsd:sequence>
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded"
minOccurs="0" name="definitions" type="foundation:Parameter"/>
          <xsd:element ecore:resolveProxies="true" maxOccurs="unbounded"
minOccurs="0" name="typeProperties" type="foundation:Parameter"/>
        </xsd:sequence>
        <xsd:attribute ecore:reference="foundation:Type" name="baseType"
type="xsd:anyURI"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:complexType name="Type">
    <xsd:attribute name="name" type="ecore:EString"/>
  </xsd:complexType>
  <xsd:complexType name="ValueElement">
    <xsd:complexContent>
      <xsd:extension base="foundation:ParameterValue">
        <xsd:attribute name="value" type="ecore:EString"/>
        <xsd:attribute ecore:reference="foundation:Type" name="type"
type="xsd:anyURI"/>
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:simpleType ecore:name="COMMON_BASE_TYPE" name="COMMON_BASE_TYPE">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="REAL"/>
```

```
      <xsd:enumeration value="INTEGER"/>
      <xsd:enumeration value="STRING"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```