MODEL-BASED TESTING FOR SCA CONFORMANCE TESTING

Julien BOTELLA[1], Eddie JAFFUEL[2], Bruno LEGEARD[1,3], Fabien PEUREUX[1,3]
[1] Smartesting R&D Center, France,
{julien.botella, bruno.legeard, fabien.peureux}@smartesting.com;
[2] eConsult, France, eddie.jaffuel@econsult.fr;
[3] Institut FEMTO-ST - UMR CNRS 6174, University of Franche-Comté, France,
{bruno.legeard, fabien.peureux}@femto-st.fr

ABSTRACT

The Software Communications Architecture (SCA) is a software architecture provided and published by the JTNC (Join Tactical Networking Center). Facing the multiplicity of the waveforms and the diversity of the platform architectures and form factors, the original aims of the SCA are to facilitate the waveform development in terms of portability and waveform deployments onto heterogeneous SDR platforms. In this paper, we present an approach using Model-Based Testing (MBT) to ensure the conformance of a software radio platform to SCA requirements. In this approach, an MBT model is developed on the basis of SCA specifications, and conformance tests and scripts are generated and then run on the targeted software radio platform. This approach has been developed within a National Research Project called OSeP, with results regarding modeling for automated test generation for SCA conformance testing. The techniques involved in this project focus on functional requirements and automatically generate Java executable test scripts, which aim to evaluate the functional conformance of the software implementation with respect to their associated requirements.

Keywords: Software Communications Architecture (SCA), conformance testing, model-based testing, dynamic testing.

1. INTRODUCTION

Conformance testing is done to determine whether a system meets a specified standard. One key goal of conformance testing is to ensure interoperability between systems, on the basis of agreed norms and standards. Conformance tests are designed to concentrate on areas critical to interoperability, including testing the system reaction to erroneous behavior. One specific challenge in the area of conformance testing is the design of the right test suites: how can be designed the right tests? How can be agreed, at the level of standard working group committees, on the content of the conformance test suite? How the bidirectional traceability matrix between conformance tests and the standard can be developed and maintained when the specifications change?

In this paper, we provided first results on using Model-Based Testing (MBT) [1] from UML models [2] to evaluate the functional conformance of the software implementation with respect to the Software Communications Architecture (SCA) [3]. MBT refers to the processes and techniques for the automatic derivation of abstract test cases from abstract models, the generation of concrete tests from abstract tests, and the manual or automated execution of the resulting concrete test cases [4]. Compared with a manual design approach, MBT brings the following benefits:

- MBT Modeling is a process which fosters close communication of the stakeholders.
- The forced communication process builds up a common perception and understanding of the requirements in the given domain and helps to concentrate on areas critical to interoperability.
- Reducing information and emphasizing different perspectives in the conformance MBT model makes it easier to master tradeoff and balance of the generated conformance test suite.
- It helps to reduce maintenance costs due to the "single-point" information in the MBT model and the "by-design" traceability between the model and standard.

MBT is an increasingly widely-used approach that has gained much interest in recent years. It is today getting closer and closer to an industrial reality: theoretical concepts (and associated tools) to derive test cases from specifications are indeed now mature enough to be applied in many application areas [5][6]. MBT is already in used for conformance testing in several areas of industry. We can mention for example:

- The ETSI conformance testing process – see [7].
- GlobalPlatform compliance program – see [8]. In section 2, we present the lessons learnt from applying MBT conformance testing for the compliance program of GlobalPlatform 2.2 card specification.

Therefore, our main goal was to evaluate the technical feasibility of applying a Model-Based Testing process for SCA conformance testing. This proof of concept has been done in the context of a National Research Project called OSeP[1] in partnership with DGA MI (French DoD).

The rest of the paper is organized as follows: Section 2 provides a short description of the MBT conformance testing process applied to Global Platform and summarizes the lessons learnt in this context. Section 3 gives a detailed overview of the application of this MBT process to a subset of SCA 2.2.2 specifications and summarizes the lessons learnt from our experiments. We conclude our paper in Section 4 and propose some perspectives to this work.

2. RELATED MBT INDUSTRIAL EXPERIENCE: GLOBALPLATFORM COMPLIANCE PROGRAM

GlobalPlatform is a cross industry and not-for-profit association, which members are payment organizations such as American Express, MasterCard, or Visa International, telecom operators, like AT&T, France Telecom, NTT or Verizon and industrial leaders (AMD, Apple, Blackberry, Gemalto, Nokia, Samsung, etc.).
As shown in Figure 1, GlobalPlatform identifies, develops and publishes specifications facilitating secure and interoperable deployment and management of multiple embedded applications on secure chip technology. Its proven technical specifications are regarded as the international industry standard for building a trusted end-to-end solution serving multiple actors and supporting several business models.

Figure 2: GlobalPlatform Standard Presentation

[1] OSeP - On-line and off-line model based testing of Security Properties – Applications to Security Components and Software Radio – ANR Grant n° ANR-11-ASTR-00 2 – see http://osep.univ-fcomte.fr/ (in French)

The specifications available provide the foundations for market convergence and innovative new cross-sector partnerships. The technology has been adopted globally across finance, mobile/telecom, government, healthcare, retail and transit sectors. Research conducted by Eurosmart confirmed that 2012 shipments of microcontroller smart secure devices (secure chips) is over 7 billion units, of which 2.6 billion units leverage GlobalPlatform technology.
For a standardization body like GlobalPlatform, the compliance program is a strategic mission (see Figure 2).

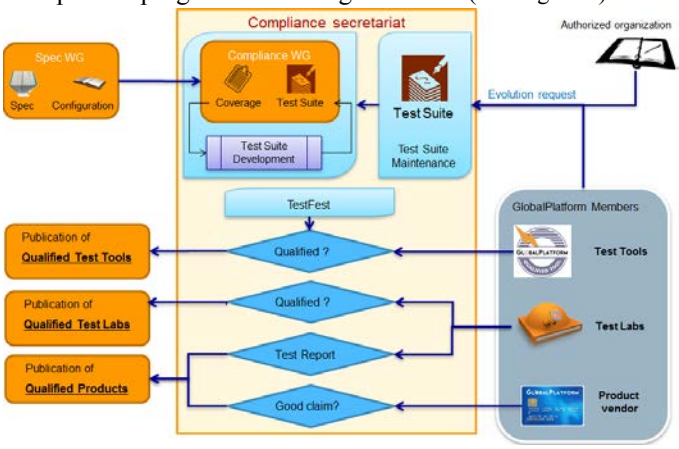

Figure 1: GlobalPlatform Certification Process

GlobalPlatform group has been using Model-Based Testing to produce its compliance test suites for more than 5 years. At GlobalPlatform, Model-Based Testing is therefore a key technology that supports the strategic conformance activity.
Figure 3 gives an overview of the Model-Based testing process to address the GlobalPlatform conformance issues. The process starts on the left at the textual requirements, from which a test designer team derive the Test Objective Charter and a UML test model. This model represents the expected behavior of the Application Protocol Data Unit (APDU) specified in the GlobalPlatform standard. A subset of UML, called UML4MBT [10], is used. It includes UML class diagrams, state machines and OCL [11] constraints to formalize the control points and observation points, the expected dynamic behavior described in the standard, the business entities associated with the test, and some data for the initial test configuration. Model elements such as transitions or decisions are linked to the requirements defined in the Test Objective Charter, in order to ensure bi-directional traceability between these requirements and the model, and later to the generated test cases and related test plan. Models are therefore precise and complete enough to allow automated derivation of tests from these models. This derivation is a fully automated process supported by the Smartesting *CertifyIt* testing tool [12], which generates abstract test cases (abstract because relying on the UML test model) to cover the items of the Test Objective Charter file.

Each generated test case is typically a sequence of APDUs, with input parameters and expected output values for each action. An adaptation layer can be used to link some abstract values from the model with some concrete test values. Such generated test sequences are similar to the high-level test sequences that would be designed manually in action-word testing [13]. They are therefore easily understood by humans, e.g., GlobalPlatform Compliance Testing Group, and complete enough to be delivered to be directly executed on a targeted system by a manual tester.

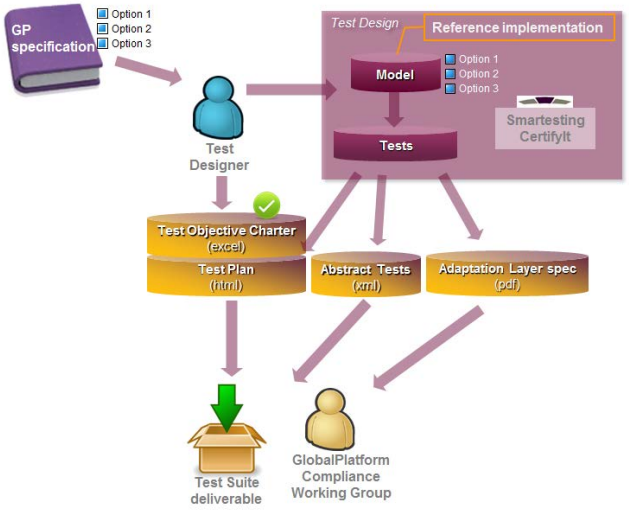

Figure 3: Model-based Compliance Test Suite

In this context, the major added value of the MBT process has been the following:

- It provides Test Suite for integration to the Product vendors in-house systems.
- It remains open to any Test Tools suppliers (let the market decide the best tools).
- It supports product variants or options (enabling to reuse all or some parts of the test model)
- It ensures and maintains the coherence between all deliverable assets of the testing process.
- The Model can be used as the unique reference implementation.

The GlobalPlatform Compliance Program has started in 2007. The metrics of the last GP Compliance Program in 2014 are the following (i.e. the previous versions of the Test Suites are not taken into account) : about 6 000 tests have been generated for 15 active Compliance Test Suites. On the basis of this success story [14], we decided to apply this MBT approach for SCA specifications conformance issues. The next section introduces this work and describes the obtained results.

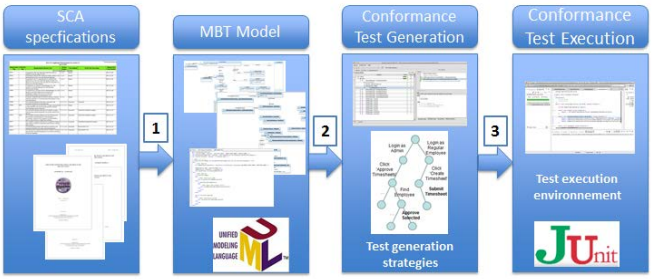

Figure 4: MBT Process for SCA Conformance Testing

3. EXPERIMENTS ON SCA 2.2 SPECIFICATIONS

The development of radio protocols, within Software Defined Radio (SDR) design context, requires the respect of the de facto Software Communication Architecture (SCA) standard [3]. To test SCA compliancy and interoperability between SDR platforms, we have studied the adaptation of the MBT approach introduced in the previous section. Figure 4 describes the overall MBT process that has been deployed on a subpart of SCA 2.2.2 specifications (functional requirements at the level of the SCA core framework). It is structured in three main steps:

1. Modeling for Test Generation from SCA specifications. From functional aspects of SCA 2.2.2, the MBT model is developed using the UML subset UML4MBT (in an eclipse-based modeling environment) and is checked for consistency. This MBT model captures the expected behavior of the SDR platform with respect to the considered perimeter of the SCA specification.
2. Automated Test Generation. Test selection criteria are chosen, to guide the automatic test generation so that it produces a test suite aligned with the test strategy. In the context of SCA conformance testing, SCA requirements are linked to elements of the model, and the coverage of these requirements drives the test generation . The precise and unambiguous meaning of the UML4MBT model makes it possible to simulate the execution of the model, to use it as an oracle by predicting the expected output of the system under test, and finally to provide traceability matrix that gives a clear functional coverage metrics. Within the project, the Smartesting test generator, namely *CertifyIt*, has been used to produce test cases and test scripts.
3. Automated Test Execution on a Test Bench. Once the test suite has been generated, the test cases are run. Test execution may be manual—i.e. by a physical person—or may be automated by a test execution environment that provides facilities to automatically execute the tests and record test verdicts. In our context, the tests are generated in Java language and automatically executed using the JUnit framework.

Requirement Tag	Criterion Tag	Requirement/Criterion Text	Section Number	Test Method	JTAP Test Case Name	Manual Test Case Number
SCA 2.2.2 Specification - Main Body						
AP0011		A log producer shall only output log records that contain an enabled CosLwLog::LogLevel value.	3.1.2.2.1	Manual		APP_TC_001
AP0012		Log producers shall use their component identifier attribute in the producerId field of the CosLwLog::ProducerLogRecord.	3.1.2.2.1	Manual		APP_TC_001
AP0013		Log producers and CF components that are required by this specification to write log records shall operate normally in the absence of a log service or in the case where the connections to a log are nil or an invalid reference.	3.1.2.2.1	Manual		APP_TC_001
AP0063		A component (e.g., Resource, DomainManager, etc.) that consumes events shall implement the CosEventComm PushConsumer interface.	3.1.2.3.1	Manual		APP_TC_029
AP0064		A component (e.g., Resource, Device, DomainManager, etc.) that produces events shall implement the CosEventComm PushSupplier interface and use the CosEventComm PushConsumer interface for generating the events.	3.1.2.3.1	Manual		APP_TC_029
AP0065		A producer component shall not forward or raise any exceptions when the connection to a CosEventComm PushConsumer is a nil or invalid reference.	3.1.2.3.1	Manual		APP_TC_029
AP0069		The connectPort operation shall make a connection to the component identified by its input parameters.	3.1.3.1.1.5.1.3	Automated	ConnectPort	APP_TC_015
AP0069	C002	A port may support several connections. The input connectionId is a unique identifier to be used by the disconnectPort operation when breaking a specific connection.	3.1.3.1.1.5.1.3	Manual		APP_TC_015
AP0070[2]		The connectPort operation shall raise the InvalidPort exception when the input connection parameter is an invalid connection for this port.	3.1.3.1.1.5.1.5	Automated	ConnectPort InvalidPort Exception	APP_TC_014
AP0070	C004[2]	The InvalidPort exception indicates one of the following errors has occurred in the specification of a Port association: 1. errorCode 1 means the Port component is invalid (unable to narrow object reference) or illegal object reference.	3.1.3.1.1.3.1	Automated	ConnectPort InvalidPort Exception	APP_TC_014
AP0071		The connectPort operation shall raise the OccupiedPort exception when unable to accept the connections because the port is already fully occupied.	3.1.3.1.1.5.1.5	Automated	ConnectPort Occupied Port Exception	APP_TC_015
AP0072		The disconnectPort operation shall break the connection to the component identified by the input connectionId parameter.	3.1.3.1.1.5.2.3	Automated	DisconnectPort Test	APP_TC_012
AP0073[2]		The disconnectPort operation shall raise the InvalidPort exception when the input connectionId parameter is not a known connection to the Port component.	3.1.3.1.1.5.2.5	Automated	DisconnectPort Test	APP_TC_012

Figure 5: Studied Excerpt of SCA 2.2.2 Application Requirements List Version 2.2.

The next subsections detail each step in the context of the SCA 2.2.2 functional specification conformance testing. This presentation focuses on the « Domain manager » function to « install / uninstall Application» nominally, including exception management as shown in Figure 5.

3.1. From SCA specifications to the MBT model

The test model is specified on the basis of the UML4MBT modeling language introduced in the previous section. It is composed of a class diagram to represent the static view of the system (using classes, associations, enumerations, class attributes and operations) and an Object diagram to list the concrete objects used to compute test cases and to define the initial state of the system. In addition, Object Constraint Language (OCL) expressions are associated with the UML class operations to provide the expected level of formalization to precisely describe the dynamical behaviors of the system. Conformance requirements traceability is managed by tagging the OCL effects of the class operations. More precisely, ad-hoc comment symbols are introduced to OCL annotations to associate the requirement identifiers with an OCL statement. When a test case covers this statement, this test case is referenced as covering the requirement defined by the annotated identifier. This tagging mechanism makes it very easy to link initial functional requirements with the corresponding model behavior. The global architecture of the model (introduced in Figure 6) conforms with the structure proposed in SCA specification: one dedicated package has been created to model each specified SCA interface. The next subsections give an overview of each artefact of the model.

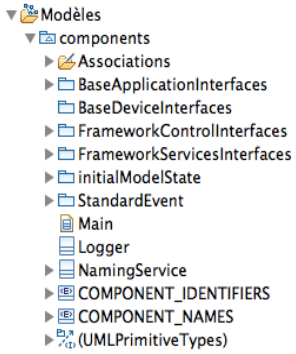

Figure 6: Model Structure using SCA Packages

3.1.1. Class diagram

Figure 7 shows the class diagram of the SCA test model. Each class may contain one or several operations (not displayed in the Figure to keep it readable), which correspond to the services that can be applied to the system.

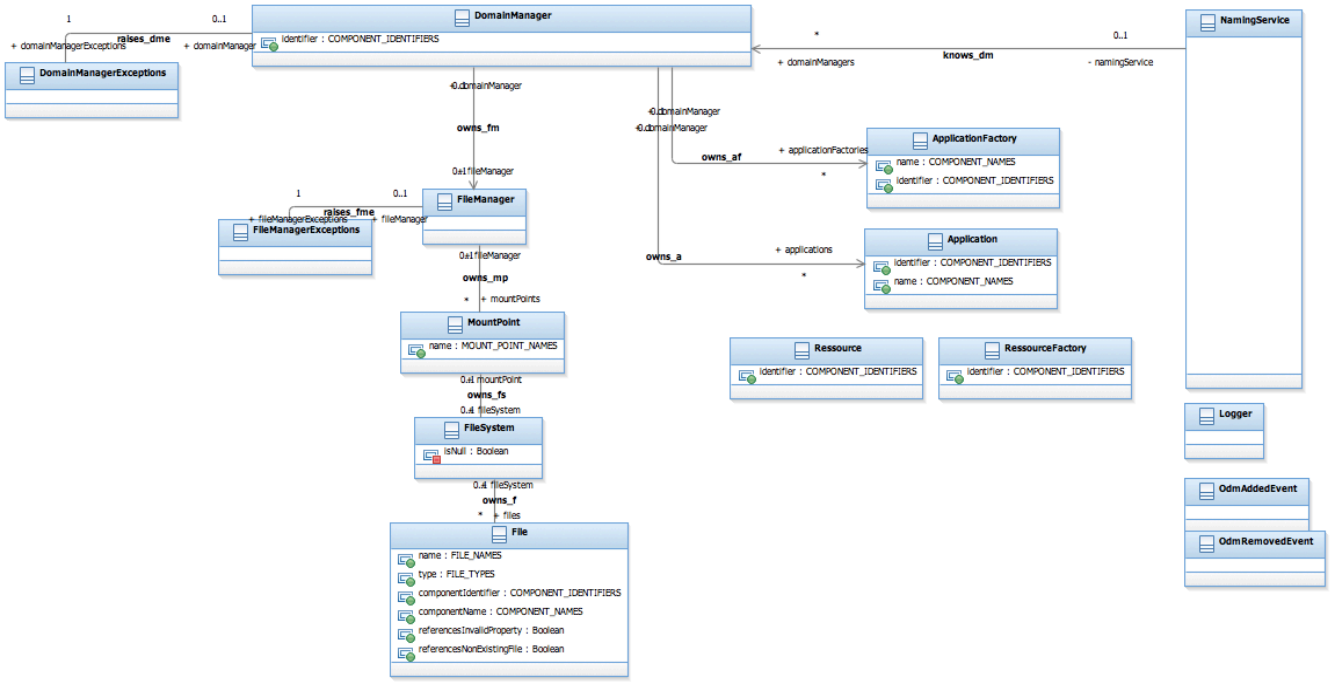

Figure 7: Class Diagram of the SCA Model

*Main *DiagrammeObjet1 mount

```
self.resetExceptions() and
let invalidFileName : Boolean = (mountPointName = MOUNT_POINT_NAMES::INVALID_NAME) in
let mountPointAlreadyExists : Boolean = (self.mountPoints->exists(mp|mp.name=mountPointName)) in
let invalidFileSystem : Boolean = fileSystem.isNull in

---@REQ: 3.1.3.4.3.5.1
if (not(invalidFileName) and not(mountPointAlreadyExists) and not(invalidFileSystem)) = true
then
    ---@AIM: MOUNT_OK
    let newMountPoint:MountPoint = MountPoint.allInstances()->any(mp|mp.name=MOUNT_POINT_NAMES::UNDEFINED_NAME) in

    newMountPoint.name = mountPointName and
    newMountPoint.fileSystem = fileSystem and
    self.mountPoints->includes(newMountPoint)
else
    ---@AIM: MOUNT_KO
    if (invalidFileName)
    then
        ---@AIM: INVALID_FILE_NAME
        self.raiseInvalidFileNameException()
    else
        if (mountPointAlreadyExists)
        then
            ---@AIM: MOUNT_POINT_ALREADY_EXISTS
            self.raiseMountPointAlreadyExistsException()
        else
            true
        endif
    endif

    and if (invalidFileSystem)
    then
        ---@AIM: INVALID_FILE_SYSTEM
        self.raiseInvalidFileSystemException()
    else
        true
    endif
endif
```

Figure 8: OCL Constraints of the Operation *mount()*

3.1.2. OCL constraints

The expected behavior of each specified operations is described by an OCL constraint to determine its effects. It allows the Smartesting *CertifyIt* tool to predict them in an automated manner. For instance, Figure 8 introduces the constraints of the operation *mount()*.

Requirements traceability is managed by tagging these OCL effects. More precisely, these ad-hoc comment symbols are used in the OCL annotations to associate the requirement identifiers with an OCL statement. When a test case covers this statement, this test case is referenced as covering the requirement defined by the annotated comment symbols. As an example, in Figure 8, the green expressions (specific line comment) starting with the keywords REQ (for high level requirement) or AIM (for sublevel requirement) associate the OCL code with the functional requirement identifiers that the OCL statement precisely covers.

This tagging mechanism makes it very easy to link initial functional requirements with the corresponding model behavior. They allow to automatically produce a traceability matrix at the same time as the generated test cases: when a test case executes the annotated statement, this test case is referenced as covering the annotated requirement(s).

3.1.3. Object diagram

Finally, the class diagram is instantiated using an object diagram that allows to determine the initial state of the system to be tested. Several object diagram can be created to cover various scenarios and/or various configurations depending of the testing objectives. Usually, one such diagram is created for each test suite to address the specific testing goals and functional features of them. Figure 9 introduces an excerpt of such model.

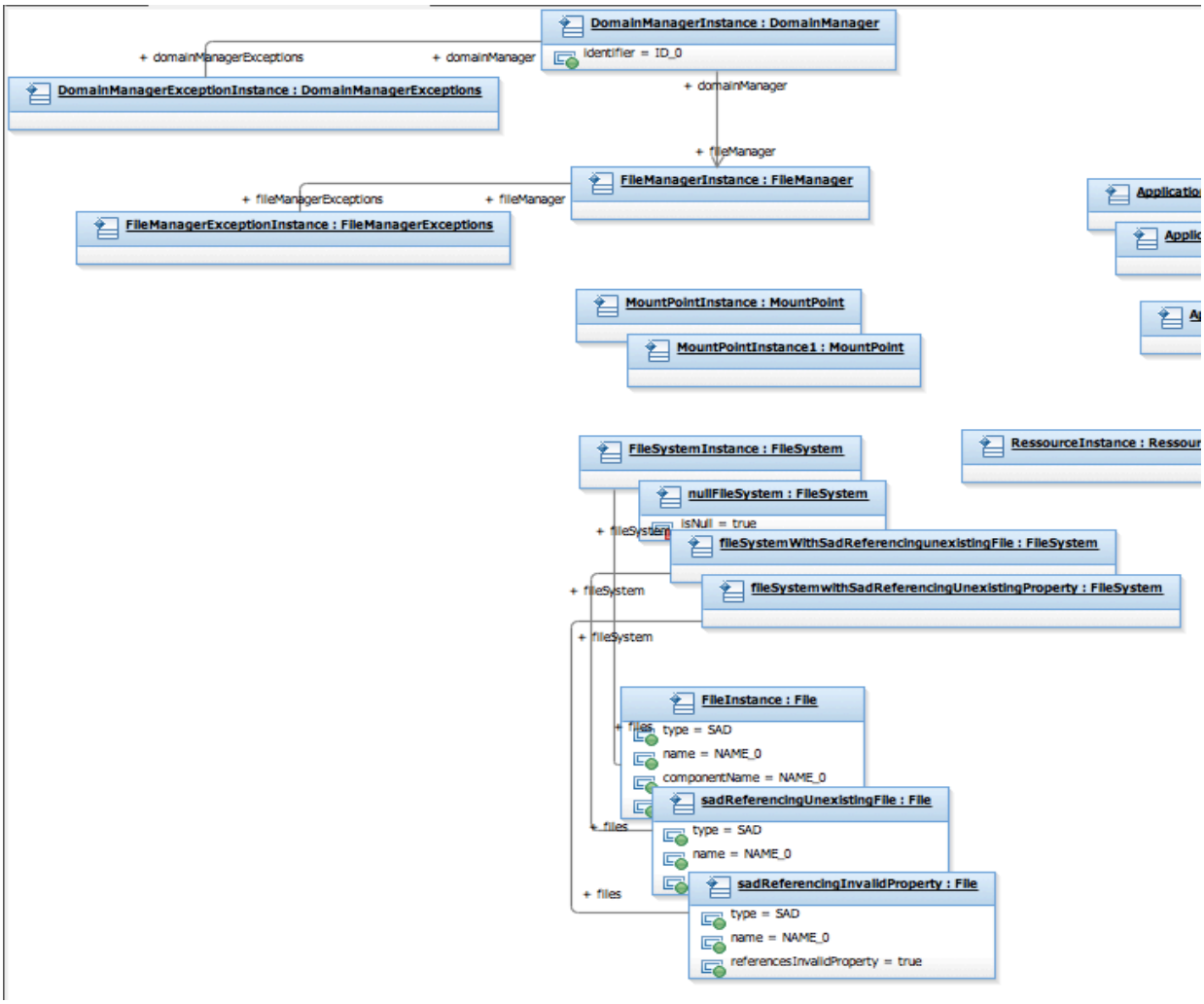

Figure 9: Excerpt of one Object Diagram

3.2. Test generation from the MBT model

Such UML4MBT models have a precise and unambiguous meaning, so that the behavior of those models can be automatically understood and manipulated by the Smartesting *CertifyIt* test generation engine. This precise meaning makes it possible to simulate the execution of the model, to use it as an oracle by predicting the expected output of the system under test, and finally to provide traceability matrix that gives a clear functional coverage metrics from the requirements point of view. Basically, the test generation algorithm carries out a systematic coverage of all the behaviors of the test model, which are tagged with a requirement identifier as shown in previous subsection. Each test corresponds to a sequence of operations taking the form of a 3-part structure: a first subsequence places the system in a specific context (preamble) to exercise a given behavior annotated by a requirement, a second subsequence invokes this behavior, and finally a last subsequence allows returning to the initial state so that test cases can be executed automatically in one single sequence. It should be noted that this 3-part structure can be completed by one or more observation function calls, which allow observing the system state at any time during the test execution to make the verdict assignment more relevant.

After test generation procedure is computed, a dedicated window of the test generator (see Figure 10) shows the set of generated test cases (at the left), and the sequence of called operations (at the top right) with the list of covered requirement identifiers (at the bottom right).

3.3. Test automation and execution

The generated test cases, which therefore include stimuli and expected outputs, can be exported to a large variety of format including customizable HTML or XML files, or directly to a scripted executable format computable in any testing framework (simulated system or real test bench). Within SCA case-study, the generated test cases are published as executable JUnit files.

Automation relies on the implementation of keywords, which are defined by the operations of the UML model, and the test data, which are define by the abstract attributes and values in this model. Finally, to ensure a fully automation, an adaptation layer (that is manually designed) concretizes the abstract test data of the model (operation names, inputs, outputs…) into concrete API calls and values. This layer can be seen as a table mapping the abstract data of the UML model to the concrete ones of the system to be tested. Thus, test publisher and adaptation layer make it possible to automatically derive executable test cases and offers the benefit of providing a structured and repeatable process. Such executable test suites can indeed be delivered to SDR manufacturers and platform providers in order to check, at a early stage of their development process, the compliance of their products. Moreover, automating test execution is a key aspect of regression testing (i.e. re-running test cases from existing test suites to build confidence that software changes have no unintended side-effects). Without test automation, testers have to execute the tests manually for each release of the application: a costly and time-consuming process.

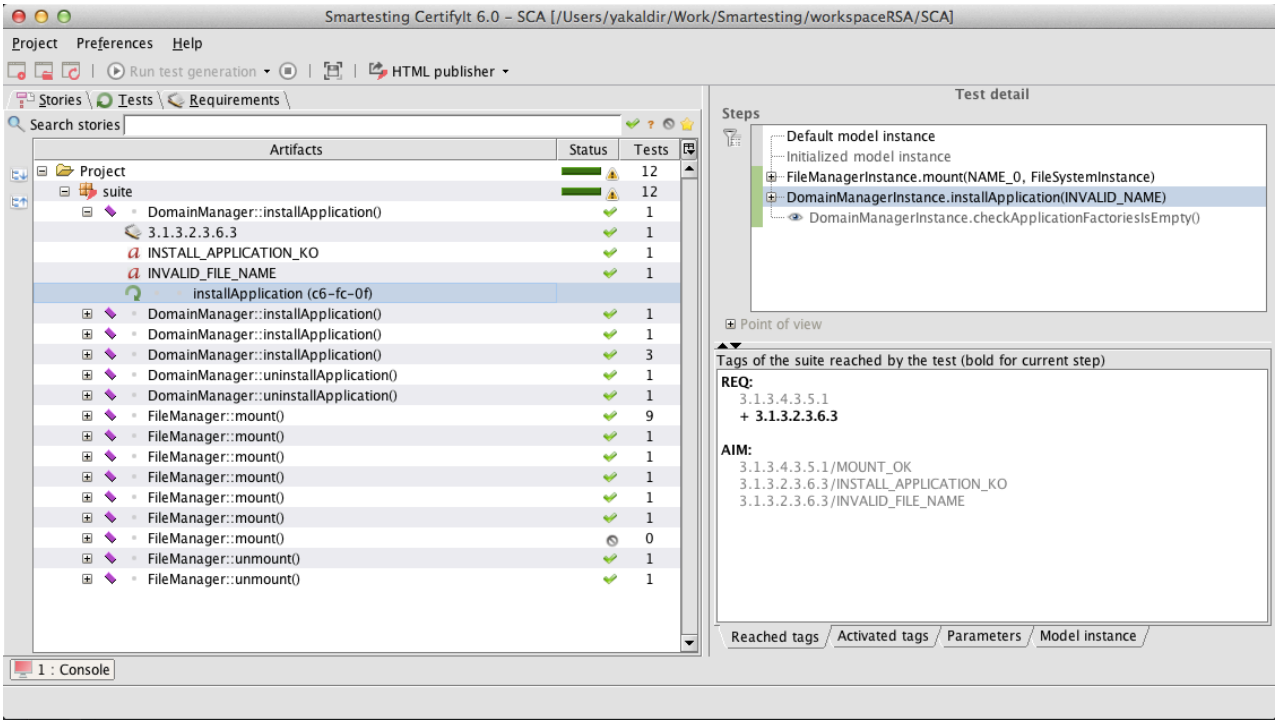

Figure 10: Smartesting CertifyIt GUI with Generated Test Cases

Figure 11 and Figure 12 respectively depict an example of generated JUnit file and the corresponding Java library that declares the UML keywords that have to be implemented. These files are automatically generated by the Smartesting *CertifyIt* testing tool. Hence, the Java keyword library specification is automatically generated from the UML model, but has to be manually documented. To achieve that, for each keyword (representing the UML operations of the model), the implementation activity consists to complete its definition by implementing the stub (see TODO comments) with the corresponding concrete code instructions.

Therefore, this file is manually designed since it directly depends on the implementation to be tested. To perform this task, as shown in Figure 13, the generated files allow the user to document the commands as well as the concrete data to be used in order to concretize and execute the generated test suite. Once this automation design is completed, the generated test cases can be executed using a JUnit engine and the related test execution report can be computed and delivered. Figure 14 shows an example of execution status given by the Eclipse interface.

```
package Smartesting.SCA.suite;

import junit.framework.TestCase;
/*
REQUIREMENTS:
    3.1.3.2.3.6.3
    3.1.3.4.3.5.1
*/
public class InstallApplication__c6_a4_38_ extends TestCase {

    private AdapterImplementation adapter;

    public void setUp() throws Exception {
        adapter = new AdapterImplementation(new TypesAdapterImplementation());
    }

    public void testInstallApplication__c6_a4_38_() throws Exception {
        adapter.componentsFrameworkServicesInterfacesFileManagermount(FileManager.FileManagerInstance, MOUNT_POINT_NAMES.NAME_0, FileSystem.FileSystemInstance);
        adapter.componentsFrameworkControlInterfacesDomainManagerinstallApplication(DomainManager.DomainManagerInstance, FILE_NAMES.INVALID_NAME);
        adapter.componentsFrameworkControlInterfacesDomainManagercheckApplicationFactoriesIsEmpty(DomainManager.DomainManagerInstance);
    }

    public void tearDown() throws Exception {
        adapter.closeAdapter();
    }
}
```

Figure 11: Example of Generated JUnit Test File

```
package Smartesting.SCA;

import Smartesting.SCA.TypesDeclaration.ApplicationFactory;

public class AdapterImplementation implements AdapterInterface {
    private TypesAdapterInterface typesAdapter;

    public AdapterImplementation(TypesAdapterInterface typesAdapter){
        this.typesAdapter = typesAdapter;
    }

    @Override
    public void componentsFrameworkServicesInterfacesFileManagermount(
            FileManager receiverInstance, MOUNT_POINT_NAMES mountPointName,
            FileSystem fileSystem) throws Exception {
        // TODO Auto-generated method stub

    }

    @Override
    public void componentsFrameworkControlInterfacesDomainManagercheckApplicationFactoriesIsEmpty(
            DomainManager receiverInstance) throws Exception {
        // TODO Auto-generated method stub

    }
```

Figure 12: Generated Java Keyword Library

```java
import SCA.TypesDeclaration.COMPONENT_NAMES;
import SCA.TypesDeclaration.FILE_NAMES;
import SCA.TypesDeclaration.MOUNT_POINT_NAMES;
import SCA.TypesDeclaration.OdmAddedEvent;
import SCA.TypesDeclaration.OdmRemovedEvent;
import SCA.TypesDeclaration.SOURCE_CATEGORY_TYPE;
import SCA.TypesDefinition.COMPONENT_IDENTIFIERS;
import SCA.TypesDefinition.FileSystem;

public class AdapterImplementation implements AdapterInterface {

    static String tab = "    ";
    PrintStream ps;
    int ok = 0;
    int ko = 0;
    private String fileLogs = "./fileLog.txt";
    private FileOutputStream fileLog;
    private PrintStream pfileLog;
    private int appFactorySeqLength = 0;

    @Override
    public void componentsFrameworkControlInterfacesDomainManagercheckApplicationFactories(
            SCA.TypesDefinition.DomainManager receiverInstance,
            SCA.TypesDefinition.ApplicationFactory applicationFactory,
            SCA.TypesDeclaration.COMPONENT_IDENTIFIERS out_identifier,
            COMPONENT_NAMES out_name) throws Exception {
        // TODO Auto-generated method stub

    }
    @Override
    public void componentsFrameworkServicesInterfacesFileManagerunmount(
            SCA.TypesDefinition.FileManager receiverInstance,
            MOUNT_POINT_NAMES mountPoint) throws Exception {
        final FileManager fm = TypesAdapter.getConcreteValue(receiverInstance);
        try {
            fm.unmount(mountPoint.toString());
            ps.println("OK : File System unmounted");
            ok++;
            listMountedFileSystems(fm);
        } catch (NonExistentMount e) {
            ps.println("KO : Non Existent Mount Exception : File System");
            ko++;
        }
    }

    private void listMountedFileSystems(FileManagerOperations fm) {
        ps.println(tab + "List of Mounted Types : " + fm.getMounts().toString());
        for (int i = 0; i < fm.getMounts().length; i++) {
            ps.println(tab + "Mount Point : " + fm.getMounts()[i].mountPoint
                    + " \n File System : " + fm.getMounts()[i].fs.toString());
        }
    }
```

Figure 13: Java Implementation of the Keyword Library

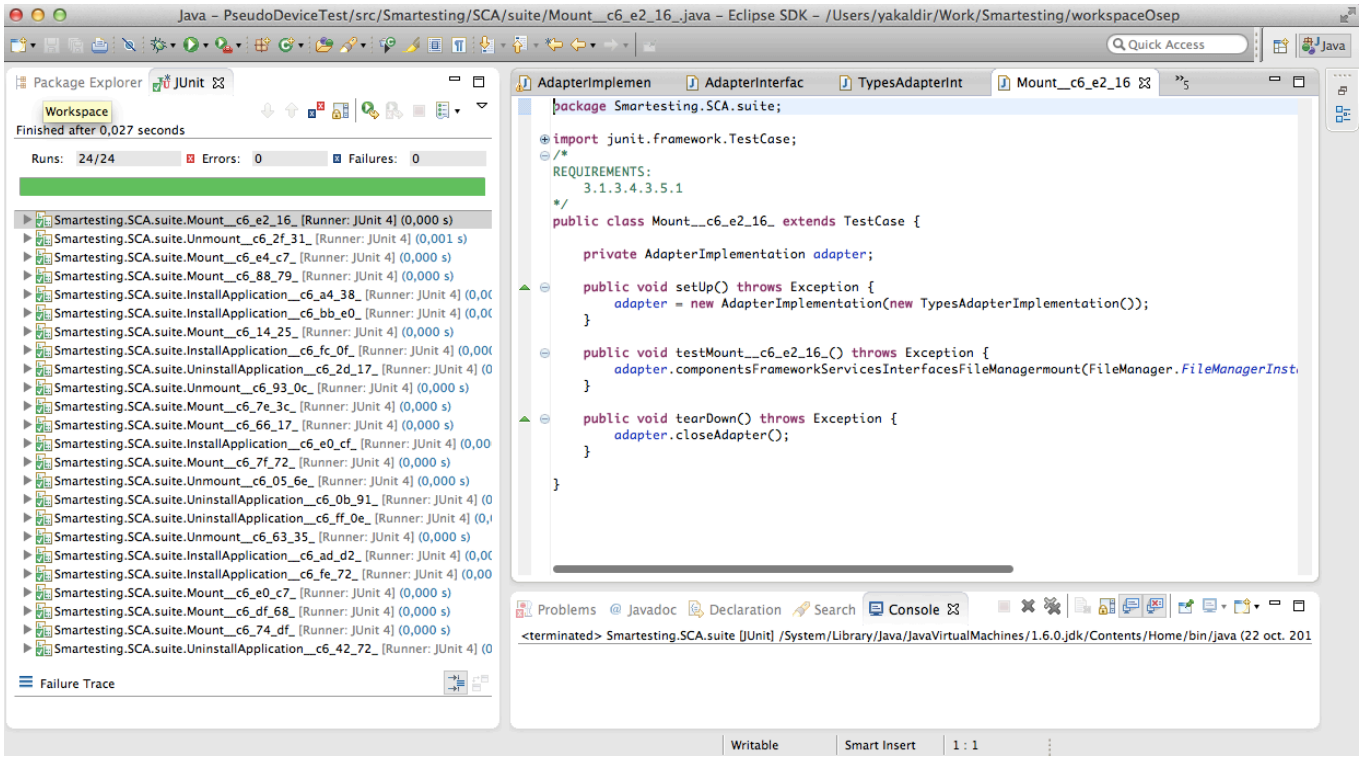

Figure 14: Example of Eclipse Test Execution Report

3.4. Lessons learnt from our experiments

The Model-Based Testing approach has been successfully applied on a subpart of the SCA 2.2.2 specifications, and this project enabled to implement a fully automated and suitable conformance testing approach for SCA standard. The main learned lessons from these experiments are:

1. The UML modeling style (the UML4MBT meta-model) is adequate to design such SCA MBT models. The interpretation of the specification was easy, and no specific issues appears during the modeling phase.
2. The annotation of the MBT model by SCA requirements (at the level of OCL constraints) is a good mean to ensure an appropriate and relevant coverage of the SCA specification during automated test generation.
3. Finally, due to a good mapping between the modeled operation and the SCA APIs of the SDR platform, automated test execution was straightforward managed.

Some other benefits, directly inherited from well-known advantages of the MBT approaches [5] (and already demonstrated on software radio protocol during a previous experiment [15]) have also been noticed. For instance, this MBT approach for SCA compliance testing reduces test maintenance costs because only the test model has to be managed instead of the test cases. Moreover, conformance tests being based on the same model, they are generated for various implementations, releases, and versions of a single application, which ensures efficient regression testing and makes easier all maintenance and upgrade activities.

4. CONCLUSION AND PERSPECTIVES

Model-based Testing (MBT) has seen last 10 years an increasing interest in different industrial area of software and system testing. This is due to the fact that benefits of MBT, such as facilitation to define and automate specialized testing strategies, help to tackle the challenges of ever more complex software and systems. In the context of conformance testing, several deployments of MBT led by industrial consortium (GlobalPlatform for instance) show that this process and technologies may help standardization organization to better conduct and master their compliance program. In this paper, we report a small but successful technical proof of concept on applying MBT for SCA conformance testing. Modeling the functional part of SCA was easy, and automated test generation techniques provide the corresponding tests to be run on the SCA platform. Therefore, experimentation feedback using this automated conformance test generation process are very encouraging.

The perspective of this project is to continue to extend the coverage of SCA functional specifications by the MBT model, and therefore to extend the generated conformance test suite.

Acknowledgment.

This work is sponsored by the ANR ASTRID OSeP project (On-line and Off-line Model-based Testing of Security Properties, ANR 11 ASTR 002) - http://osep.univ-fcomte.fr

5. REFERENCES

[1] M. Utting, B. Legeard. “Practical Model-Based Testing – A Tools Approach”, Morgan & Kauffmann, 2007

[2] J. Rumbaugh, I. Jacobson and G. Booch. “The Unified Modeling Language Reference Manual”, Second Edition. Addison-Wesley, 2004. ISBN 0 321 24562 8

[3] JTNC Standards, Joint Tactical Networking Center, “JTRS/JPEO Software Communications Architecture Specification”, Final/15 May 2006 V.2.2.2, http://jtnc.mil/sca/Pages/default.aspx

[4] E. Bernard, F. Bouquet, A. Charbonnier, B. Legeard, F. Peureux, M. Utting, and E. Torreborre. “Model-based testing from UML models”. *Proceedings of the international workshop on Model-based Testing (MBT’2006)*, LNCS, vol. 94. pages 223–230. Dresden, Germany. October 2006.

[5] A. Dias-Neto and G. Travassos. “A Picture from the Model-Based Testing Area: Concepts, Techniques, and Challenges”. *Advances in Computers*, vol. 80, pp. 45–120, July 2010, iSSN:0065-2458.

[6] H. Zhu and F. Belli. “Advancing test automation technology to meet the challenges of model-based software testing,” *Journal of Information and Software Technology*, vol. 51, no. 11, pp. 1485–1486, 2009.

[7] “ETSI Conformance Testing Process – An introduction”, available on-line (last access June 2014) http://www.etsi.org/images/files/ETSITechnologyLeaflets/MethodsforTestingandSpecification.pdf

[8] G. Bernabeu, N. Lavabre. “Model-Based Testing for a world-wide Compliance Program”, *User Conference on Advanced Automated Testing (UCAAT 2013)*, Paris, October 2013. http://ucaat.etsi.org/2013/presentations/Keynote_MBT%20for%20a%20Compliance%20Program-GlobalPlatform-GilBernabeu.pdf

[9] J. Rumbaugh, I. Jacobson, G. Booch.: “The Unified Modeling Language Reference Manual”. 2nd edition. Addison-Wesley (2004) ISBN 0321245628.

[10] F. Bouquet, C. Grandpierre, B. Legeard, F. Peureux, N. Vacelet, and M. Utting. “A subset of precise UML for model-based testing”. *Proceedings of the 3rd international workshop on Advances in model-based testing (A-MOST'07)*, pages 95--104. ACM, July 2007.

[11] J. Warmer and A. Kleppe. “The Object Constraint Language: Precise Modeling with UML”. Addison-Wesley, 1996. ISBN 0 201 37940.

[12] F. Bouquet, C. Grandpierre, B. Legeard, and F. Peureux, “A test generation solution to automate software testing,”. *Proceedings of the 3rd international workshop on Automation of Software Test (AST’08)*. ACM Press, May 2008.

[13] H.Q. Nguyen, M. Hackett, and B.K. Whitlock. “Global Software Test Automation: A Discussion of Software Testing for Executives”, Happy About, 2006.

[14] G. Bernabeu, E. Jaffuel, B. Legeard, and F. Peureux. “MBT for GlobalPlatform Compliance Testing: Experience Report and Lessons Learned”. *Proceedings of the 25th International Symposium on Software Reliability Engineering (ISSRE’14)*, IEEE Computer Society Press, November 2014.

[15] S. Li, M. Bourdellès, A. Acebedo, J. Botella, and F. Peureux. “Experiment on Using Model-Based Testing for Automatic Tests Generation on a Software Radio Protocol”. *Proceedings of the 9th international workshop on Systems Testing and Validation (STV'12)*, October 2012.