

Achieving SCA Conformance Testing with Model-Based Testing

Julien Botella¹ · Jean-Philippe Delahaye² ·
Eddie Jaffuel³ · Bruno Legeard^{1,4} · Fabien Peureux^{1,4}

Received: 31 January 2015 / Revised: 8 June 2015 / Accepted: 16 November 2015 / Published online: 10 December 2015
© Springer Science+Business Media New York 2015

Abstract The Software Communications Architecture (SCA) is a software architecture provided and published by the Joint Tactical Networking Center (JTNC). 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 Software Defined Radio (SDR) platforms. In this paper, we present an approach using Model-Based Testing (MBT) to ensure the conformance of a software radio platform with 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 French 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 (MBT) · Dynamic testing

1 Introduction

The Software Communications Architecture (SCA) is an open architecture that provides designers information on how hardware and software artefacts are to interoperate within a Software Defined Radio (SDR). The SCA specifications, provided by the Joint Tactical Networking Center (JTNC) [11], set out requirements for behavioral specifications, interface specifications, application program interfaces (APIs), and rules. This architecture, made up of three main components (SCA Core Framework, CORBA middleware, and POSIX-based operating system) thus provides a framework in which the interoperability of products developed under this architecture is enhanced and assured. However, due to the multiplicity and diversity of the platform and form factors, the conformity of a given implementation with SCA requirements still remains a challenging and complex activity. Indeed, the increase of variable expectations regarding norms coupled with the growing technology heterogeneity lead to deploy a mess of standards, which can be supported by numerous devices and implemented with various softwares that can include previously developed code. Moreover, a lot of waveforms can be implemented

✉ Fabien Peureux
fpeureux@femto-st.fr; peureux@smartesting.com

Julien Botella
botella@smartesting.com

Jean-Philippe Delahaye
jean-philippe.delahaye@intradef.gouv.fr

Eddie Jaffuel
eddie.jaffuel@econsult.fr

Bruno Legeard
legeard@smartesting.com; blegeard@femto-st.fr

¹ Smartesting R&D Center, 25000 Besançon, France

² DGA/CELAR, French MoD, 35170 Bruz, France

³ eConsult, 25870 Cussey-sur-l'Ognon, France

⁴ Institut FEMTO-ST, UMR CNRS 6174, 25030 Besançon, France

in radio software, which have to communicate with many different radios with only a small change in software parameters. To build radios that are able to support operations in a wide variety of domains without losing the ability to communicate with each other, the SCA specifications provide a way to ensure the interoperability of compliant products.

To assess and validate such a compliance, conformance testing is today a widely-used approach. 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 test suites, and leads to the following questions:

- How the right tests can be designed?
- 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 provide first results obtained by using Model-Based Testing (MBT) [18] from UML and OCL models [16] to evaluate the functional conformance of a software implementation with respect to the Software Communications Architecture. MBT refers to a particular type of software testing processes and techniques consisting to automatically derive abstract test cases from high-level abstract models, to generate concrete tests from abstract tests, and to manually or automatically execute the resulting concrete test cases to obtain the verdict of each test.

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 [20]. Initially proposed to address functional testing [7], MBT has also been used for a few years to perform conformance testing in several areas of industry. We can mention for example projects about the ETSI conformance testing process [8] and about the GlobalPlatform compliance program [10].

Besides, we have conducted this last project about the deployment of a Model-Based Testing approach to produce compliance test suites for GlobalPlatform specifications [1]. The MBT integration has been a concrete success story, and has motivated the experimentation of this same MBT approach for SCA conformance testing. In this context, our

main goal was to explore and evaluate the technical feasibility and the relevance of applying such a Model-Based Testing process for SCA conformance testing. This proof of concept approach, developed and experimented within a French research project called OSeP¹ in partnership with DGA MI (French DoD), aimed to generate test cases using an MBT solution to validate the conformance of a software radio platform with SCA requirements. Indeed, while some works address SCA conformance (e.g., design method [17], design framework [15] as well as static analysis for compliance testing [9]), using an MBT solution to derive conformance testing in this context defines a novel and complementary approach.

The rest of the paper is organized as follows. Section 2 briefly describes the principles of the MBT approach and introduces the MBT solution used to conduct the experiments. Section 3 provides a short description of the MBT conformance testing process applied to GlobalPlatform and summarizes the lessons learnt in this domain. Section 4 gives a detailed description of the application of the MBT solution and process to a subset of SCA 2.2.2 specifications and discusses the obtained results. Finally, Section 5 concludes the paper and proposes some perspectives to this work.

2 Model-Based Testing Principles and Motivation

The MBT approach, which is used in this paper to compute conformance test cases, is depicted in Fig. 1, in which solid arrows define automated tasks whereas dotted arrows indicate tasks requiring manual design.

This approach takes as input a behavioural UML [16] and OCL [19] model (1), allowing the test generation engine both to determine relevant contexts of execution, and to predict the expected system behaviour (2). Each abstract generated test case (abstract because they are defined at the level of the input model) is typically an abstract sequence of high-level actions (operations) specified in the UML test model. Moreover, test generation algorithms make it possible to produce and maintain a bidirectional traceability matrix between generated test cases and initial system requirements. The generated abstract test cases are next concretized (3) to be automatically executed on the testing platform composed of the wrapper code generated from the model and the executable code of the components. Finally, a test report, including verdict assignments computed by comparing expected and obtained results, is automatically produced by the testing framework (4).

¹<http://osep.univ-fcomte.fr> (last access June 2015).

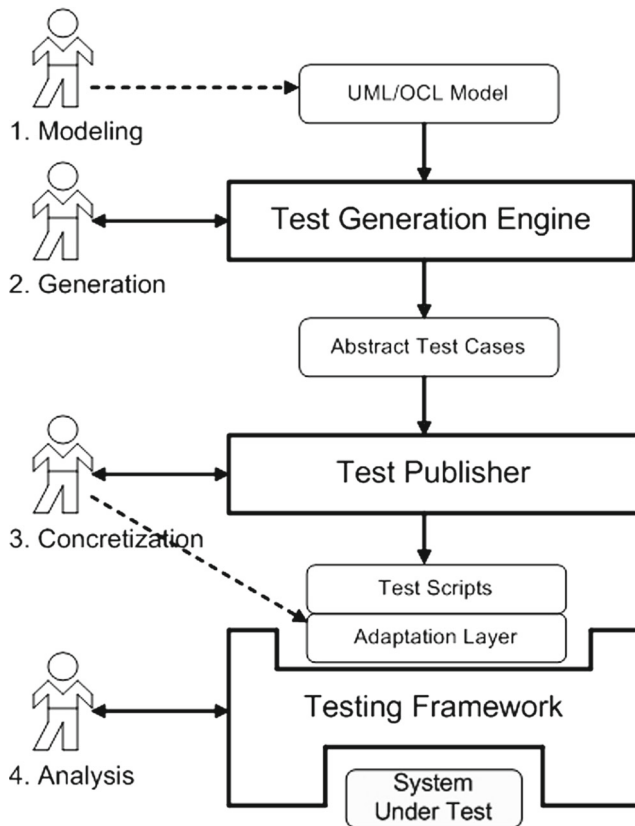


Figure 1 Model-Based Testing Process.

In the present paper, the tool solution experimented to generate conformance test cases is mainly based on the *CertifyIt* MBT tool [3], provided by the company Smartesting,² which was originally developed to generate and manage functional test cases as shown in [4]. This software is a test generator that takes as input a test model written with a subset of the UML notations called UML4MBT [5], which captures the behavior of the System Under Test (SUT). Concretely, a UML4MBT test model consists of UML class diagrams to describe the static view of the system (with classes, associations, class attributes and operations), UML Object diagrams to list the concrete objects used to compute test cases and to define the initial state of the SUT (and possibly state machines to specify behavioral aspects). Moreover, operations of the class diagrams are annotated with OCL constraints to specify the dynamic aspects of the SUT. OCL expressions provide the expected level of formalization necessary for Model-Based Testing modeling since an operational interpretation of the OCL postconditions makes it possible to determine its effect (this specific interpretation of OCL, called OCL4MBT [6], basically consists to interpret the OCL equality as an assignment). That is why

²<http://www.smartesting.com> (last access June 2015).

such UML4MBT test models have a precise and unambiguous meaning, so that these models can be understood and processed by the *CertifyIt* technology. This precise meaning makes it possible to simulate the execution of the test models and to generate test cases in an automated manner by applying predefined model coverage strategies. In this way, the generated test cases contain the sequence of stimuli to be executed, but also the expected results (to perform the verdict assignment) obtained by resolving the associated OCL constraints. Compared with a manual (traditional) test design approach, such an MBT approach is known to bring the following benefits:

- MBT modeling is a process that 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 make it easier to master trade-off 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 requirements.

The next section describes the GlobalPlatform compliance program for which this test generation process, based on the Smartesting solution, has been firstly experimented to produce conformance test suites. The obtained results and the lessons learned from this experience, which have motivated the use of this process for SCA conformance testing, are also introduced.

3 GlobalPlatform Compliance Program

GlobalPlatform is a cross industry and not-for-profit association, whose 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 Fig. 2, GlobalPlatform identifies, develops and publishes specifications facilitating secure and interoperable deployment and management of multiple embedded applications on secure chip technology.

The proven technical GlobalPlatform specifications are regarded as the international industry standard for building a trusted end-to-end solution serving multiple actors and supporting several business models. These freely available specifications provide the foundation for market

Figure 2 GlobalPlatform Standard Presentation.



convergence or innovative new cross-sector partnerships. The technology has been adopted globally across finance, telecom, mobile, healthcare, retail and transit sectors. GlobalPlatform also supports an open compliance program ecosystem to ensure the long-term interoperability of secure chip technology. Recent 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. Since 2007, the GP Compliance Program is managed using a unified process [2], which is described in Fig. 3. The Specifications Working Group is in charge to define the specifications

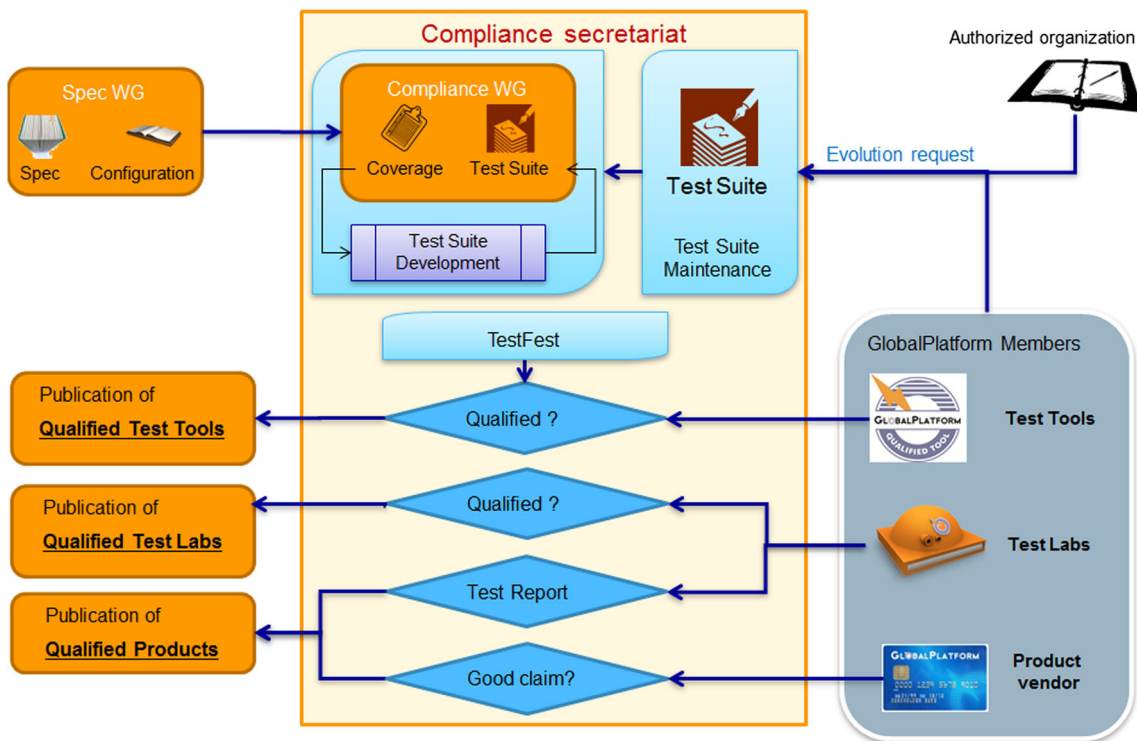


Figure 3 Process of the GlobalPlatform Compliance Program.

and the configurations. The Compliance Working Group reviews the coverage of the compliance test suite, and arbitrates interpretation of the specifications when necessary. Finally, the Compliance Secretariat manages the test suite creation and maintenance, and also organizes the TestFests. A TestFests consists of 3 or 4 days face-to-face meeting involving the GlobalPlatform Secretariat, the testing tool providers (usually 3 to 5 companies) and the product vendors (usually 2 to 4 companies). The ultimate goal of a TestFest is to qualify the testing tools (softwares as well as test harnesses), regarding a given compliance test suite.

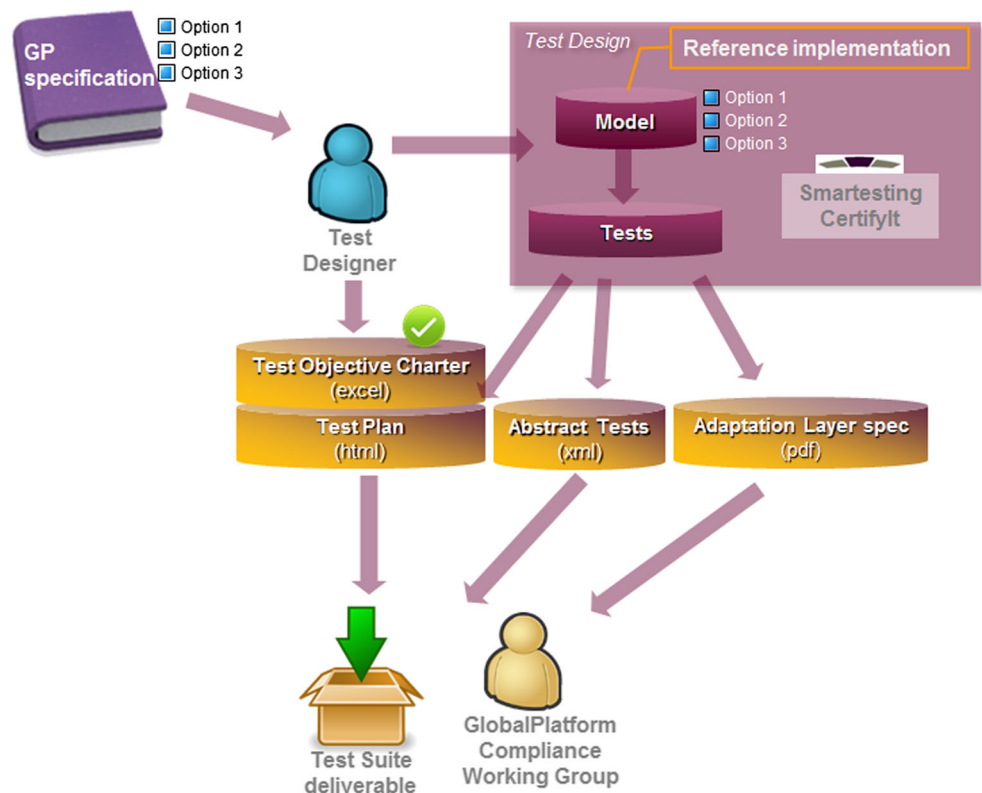
During a TestFest, all tests of the test suite are executed on every testing tools. The tests remain unchanged all along the TestFest, even if some test cases, considered as not relevant by the involved participants, may be excluded from. The product under test remains also unchanged during the TestFest: only the testing tool providers may correct their software or test harness during the TestFest when expected results are marked as wrong. At the end of the TestFest, each testing tool must give the same result as the expected result for each product, otherwise the testing tool will not be qualified. This process, required by the GlobalPlatform Secretariat for all testing tool providers and product vendors, thus allows the compliance ecosystem to be equipped with qualified testing tools and so qualified test laboratories.

Hence, the development of a well-formatted and correct compliance test suite (from the standard specifications point of view) is crucial to ensure the success of these events, which constitute today a keystone to make efficient the GP Compliance Program. To achieve this goal, GlobalPlatform group has been using Model-Based Testing, using the tool *CertifyIt* provided by the company Smartesting, 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. It enables the automatic derivation of abstract tests from UML4MBT models describing the expected GlobalPlatform requirements, the generation of the corresponding concrete tests, and finally their manual or automated execution on the different testing tools.

Figure 4 gives an overview of the Model-Based Testing process instantiated to specifically address the GlobalPlatform conformance issues.

The process starts on the left at the textual requirements, from which a test designer team derives the Test Objective Charter and a UML test model. This model, based on the UML4MBT notation, represents the expected behavior of the Application Protocol Data Unit (APDU) specified in the GlobalPlatform standard. It includes UML class diagrams, state machines and OCL constraints to formalize the control points and observation points, the expected dynamic behavior described in the standard, the business entities

Figure 4 MBT Process for GlobalPlatform Compliance Testing.



SCA 2.2.2 Application Requirements List version 2.2 July 8, 2010						
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 [12].

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. Such models are precise and complete enough to allow automated derivation of tests. This derivation is a fully automated process supported by the Smartesting *CertifyIt* testing tool, which generates abstract test cases to cover the items of the Test

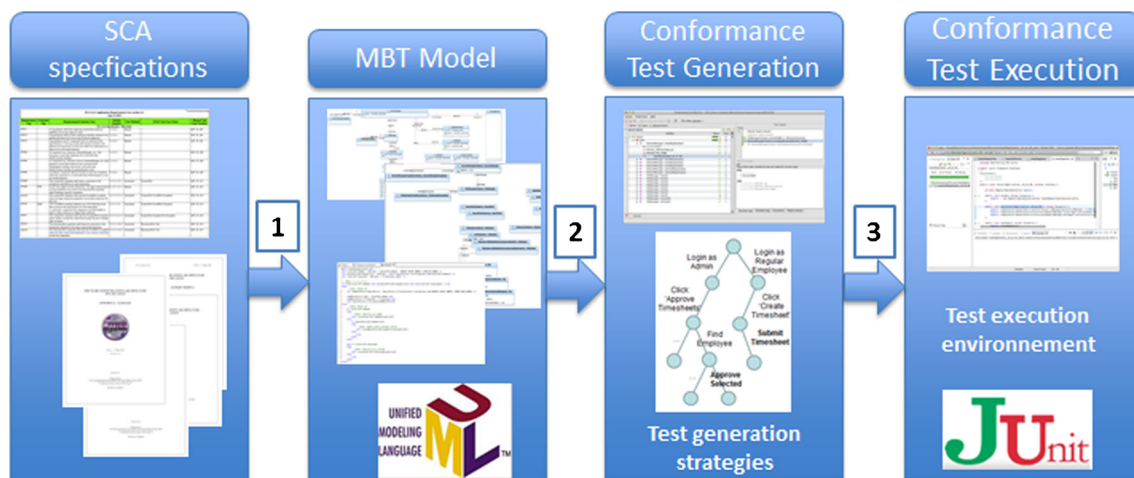


Figure 6 MBT Process for SCA Conformance Testing.

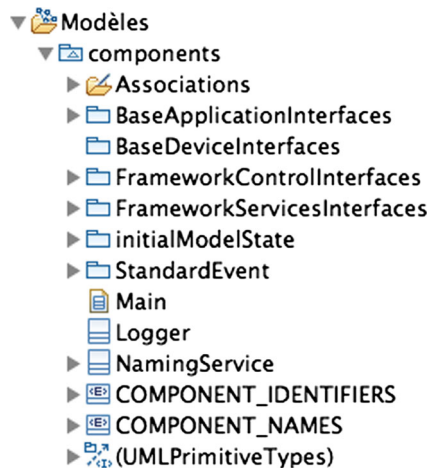


Figure 7 Model Structure for SCA Packages.

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 [14]. Therefore they are easily understood by humans, e.g., GlobalPlatform Compliance Testing Group, and complete enough to be delivered and directly executed on a targeted system by a manual tester.

In this context, the major added values of the MBT process have been the following:

- Test case generation is an automated process and so more predictive and less error-prone than manual processes. Moreover, it gives to the generated test cases a clear functional coverage metrics from the viewpoint of the Test Objective Charter.
- It provides test suite for integration to the Product vendors in-house systems and remains open to any testing tools suppliers (let the market decide the best tools).
- All generated assets (test suites, adaptation layer specifications) are kept in sync because they are derived from one common asset: the test model is used as the unique reference implementation.
- It supports product variants or options (enabling to reuse all or some parts of the test model).

From 2007 till present, the GP Compliance Program has been using this Model-Based Testing approach to produce its compliance test suite. 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 6000 tests have been generated for 15 active compliance test suites. For a further description of this MBT process and related feedback, a more detailed presentation can be found in [1]. On the basis of this success story, we have decided to apply this MBT approach for SCA specifications conformance issues. The next section describes this work and introduces the obtained results.

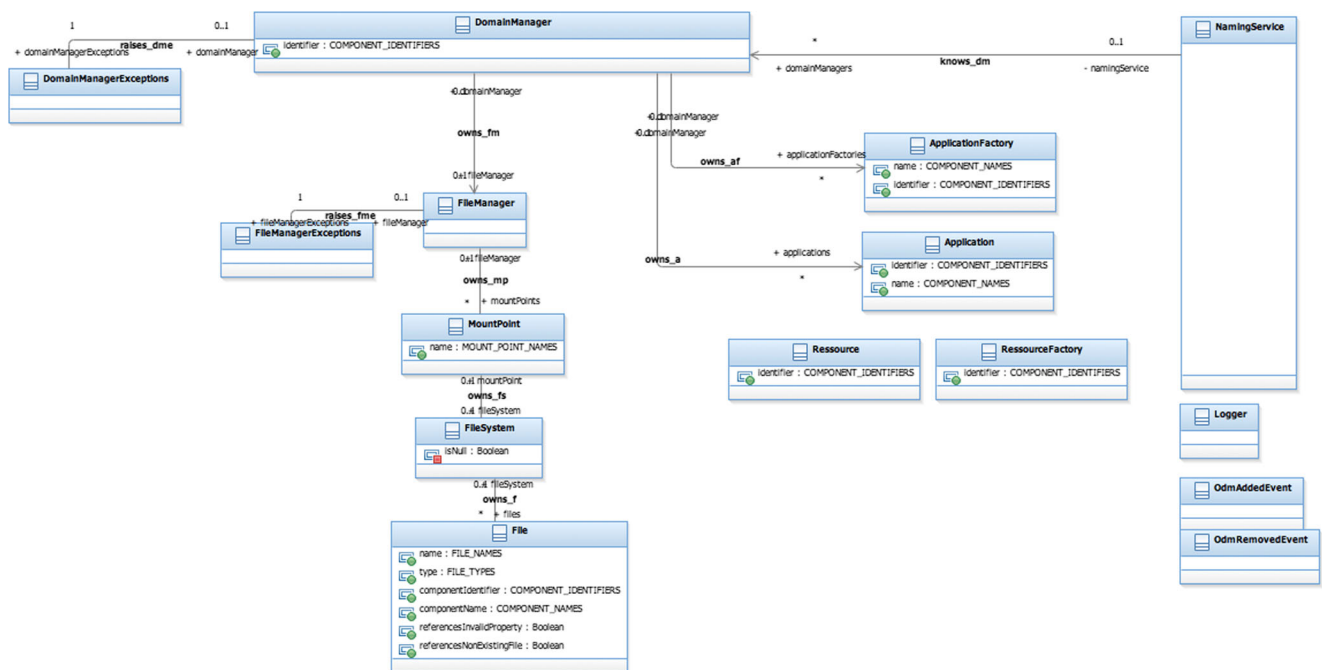


Figure 8 Class Diagram of the SCA Model.

4 Experiments on SCA 2.2.2 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 [11]. To test SCA compliance and interoperability between SDR platforms, we have studied the adaptation of the MBT approach introduced in the previous section. In the context of the SCA 2.2.2 functional specifications conformance testing, the conducted experimentation is mainly focused on the “Domain manager” function to “install / uninstall Application” nominally, including exception management [12].

In the rest of the paper, the overall presentation will thus target this part of the specifications, which takes the form of a matrix, as shown in Fig. 5. Regarding the identification of the requirements to be tested, the more important columns are the first that displays the SCA requirements identifier, and the third that expresses the requirements/criterion statements. The data in the other columns (test method, test case name and test case number) indeed relate to the test suite, provided by the Joint Tactical Radio System and Evaluation Laboratory (JTRS / JTEL), which we plan to automatically generate using the MBT process.

4.1 MBT Process for SCA Conformance Testing

Figure 6 describes the overall MBT process that has been deployed on the aforesaid subpart of SCA 2.2.2 specifications (functional requirements at the level of the SCA core framework) introduced in Fig. 5. This MBT process, directly inspired by the MBT process defined to address the GlobalPlatform compliance program, has been adapted for the SCA conformance testing context (available specifications, testing conformance goals, available technologies), and organised in three steps:

1. **Modeling for Test Generation from SCA specifications.** From functional SCA 2.2.2 requirements, 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 specifications. It should be noted that the UML4MBT test model does not contain UML state diagrams since they are not necessary to capture the behavioral aspects of the SDR platform (OCL constraints are indeed sufficient).
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 UML4MBT model makes it possible to simulate the execution of the model, to use it as an oracle by predicting the expected outputs of the system under test, and to provide traceability matrix that gives clear functional coverage metrics.

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 using a test execution environment that provides facilities to automatically compute the test cases and record test verdicts. In our context, the tests are generated in Java language and automatically executed using the JUnit framework.

The next subsections detail each of these steps, and exemplify the approach using test model, test case generation and test execution results and illustrations.

4.2 From SCA Specifications to MBT Model

The UML test model is specified on the basis of the UML4MBT modeling language introduced before. More precisely, the test model 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 dynamic behaviors.

The global structure of the UML4MBT test model conforms with the architecture proposed in the SCA specifications. Hence, one dedicated UML package has been created to model each specified SCA interface, as shown in Fig. 7.

The different artefacts of the UML4MBT test model, i.e. the class diagram, the OCL constraints (including ad-hoc OCL comment tagging to allow requirements traceability) and the object diagram, are now described in the next three subsections, respectively.

4.2.1 Class Diagram

The different entities of the system have been specified using a UML class diagram as shown in Fig. 8, which depicts 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.

4.2.2 OCL Constraints

The expected behavior of each specified operation is described by OCL constraints to determine its effects. It allows the Smartesting *CertifyIt* tool to predict them in an automated manner. For instance, Fig. 9 introduces the constraints of the operation `mount()`.

To manage conformance requirements traceability, the OCL effects are tagged to associate the requirements identifiers with the current OCL statement. As illustration, in Fig. 9, the green expressions (specific line comment) starting with the keywords *REQ* (for high level requirements) and *AIM* (for sublevel requirements) associate the OCL code with the requirements identifiers that the OCL statement precisely covers. When a test case executes this statement, it is referenced as covering the requirements defined by the annotated identifier.

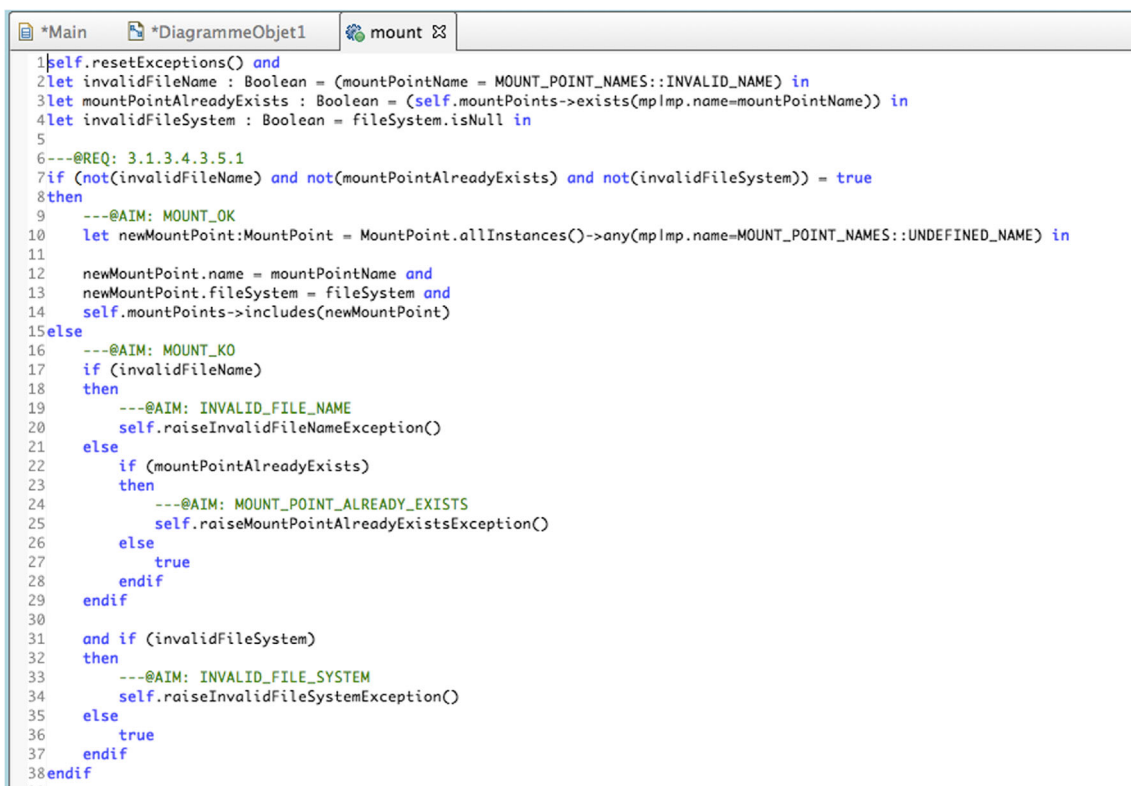
This tagging mechanism makes it very easy to link initial conformance requirements with the corresponding model behavior. It enables to automatically produce the requirements traceability matrix at the same time as the generated test cases: when a test case executes the annotated statement, this test case is indeed automatically referenced as covering the annotated requirements.

4.2.3 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. It also enables to take into account numerous customizations and specific valuation of parameters regarding the implementation under test. Figure 10 depicts an excerpt of such a model, by providing an object diagram that instantiates the class diagram previously given in Fig. 8.

4.3 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



```

1 self.resetExceptions() and
2 let invalidFileName : Boolean = (mountPointName = MOUNT_POINT_NAMES::INVALID_NAME) in
3 let mountPointAlreadyExists : Boolean = (self.mountPoints->exists(mplmp.name=mountPointName)) in
4 let invalidFileSystem : Boolean = fileSystem.isNull in
5
6 ---@REQ: 3.1.3.4.3.5.1
7 if (not(invalidFileName) and not(mountPointAlreadyExists) and not(invalidFileSystem)) = true
8 then
9   ---@AIM: MOUNT_OK
10  let newMountPoint:MountPoint = MountPoint.allInstances()->any(mplmp.name=MOUNT_POINT_NAMES::UNDEFINED_NAME) in
11
12  newMountPoint.name = mountPointName and
13  newMountPoint.fileSystem = fileSystem and
14  self.mountPoints->includes(newMountPoint)
15 else
16   ---@AIM: MOUNT_KO
17   if (invalidFileName)
18   then
19     ---@AIM: INVALID_FILE_NAME
20     self.raiseInvalidFileNameException()
21   else
22     if (mountPointAlreadyExists)
23     then
24       ---@AIM: MOUNT_POINT_ALREADY_EXISTS
25       self.raiseMountPointAlreadyExistsException()
26     else
27       true
28     endif
29   endif
30
31   and if (invalidFileSystem)
32   then
33     ---@AIM: INVALID_FILE_SYSTEM
34     self.raiseInvalidFileSystemException()
35   else
36     true
37   endif
38 endif

```

Figure 9 OCL Constraints of the Operation `mount()`.

the conformance 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 requirements 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 requirements, 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 computation 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 precise and relevant.

Once test generation procedure is computed, a window of the test generator, see Fig. 11, 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 requirements identifiers (at the bottom right).

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

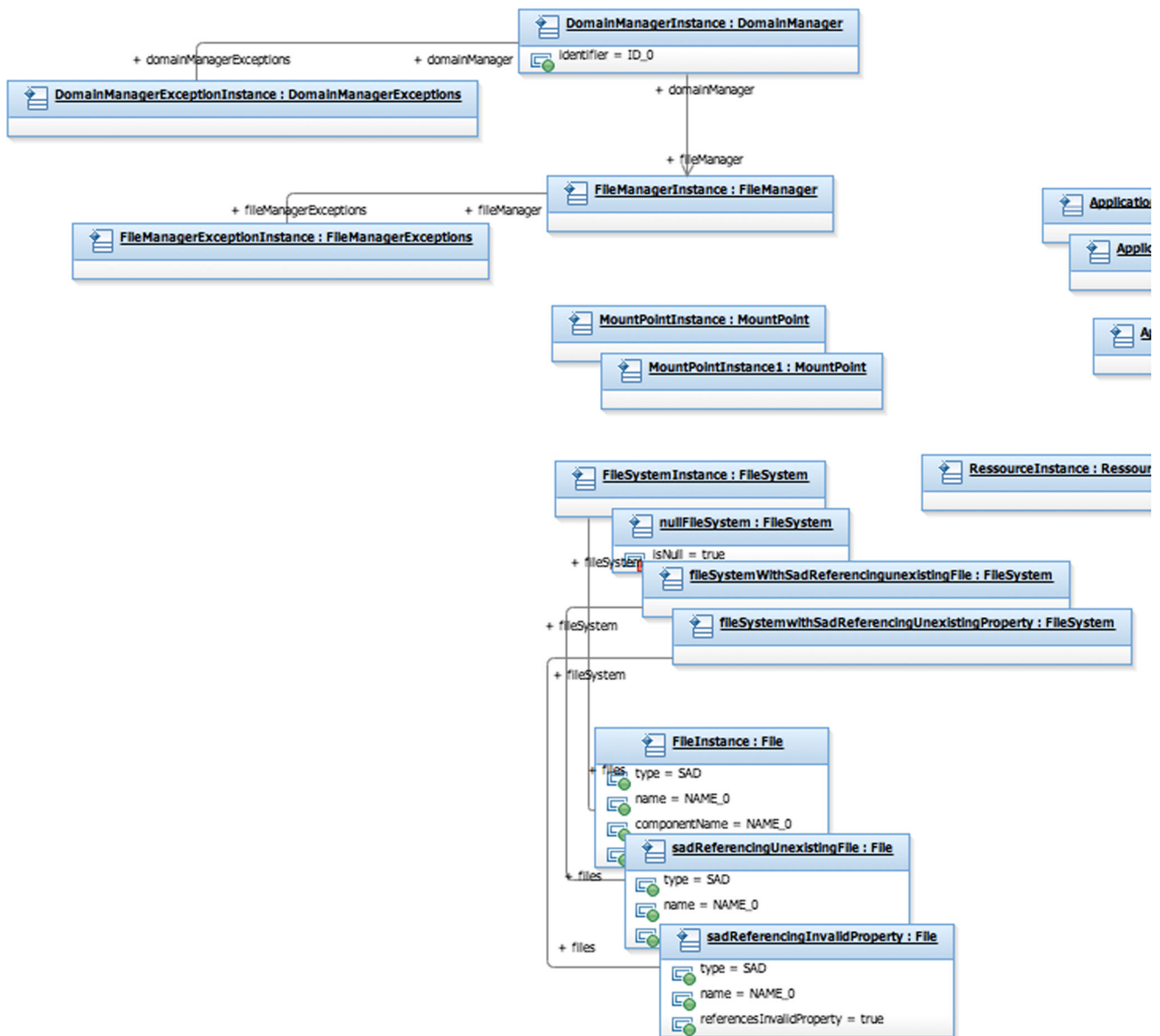


Figure 10 Excerpt of one Object Diagram.

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 defined 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 test model to the concrete ones of the system under test. Thus, test publisher and adaptation layer make it possible to automatically derive executable test cases and offer 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 an 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.

Figures 12 and 13 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. However, even if the Java keyword library file is automatically generated from the UML test model, it has to be manually documented. This file has indeed to be manually designed since it directly depends on the implementation to be tested. To achieve that, for each keyword (representing the UML operations of the test model), the implementation activity consists to complete its definition by manually implementing the stub (identified by the *TODO* comments) with the corresponding concrete code instructions.

To perform this manual task, 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 (an example is given in Fig. 14), the generated test cases can be exported and executed using a JUnit engine and the related test execution report can be computed and delivered.

Indeed, the last phase consists of exporting, in a JUnit test execution environment, the abstract test cases and the completed interfaces, which define the prototype of each operation and link the abstract structures and data of the test cases to the concrete ones. Within the SCA compliance testing context, each abstract test case is exported as a JUnit test case, and all the test cases belong to a unique JUnit

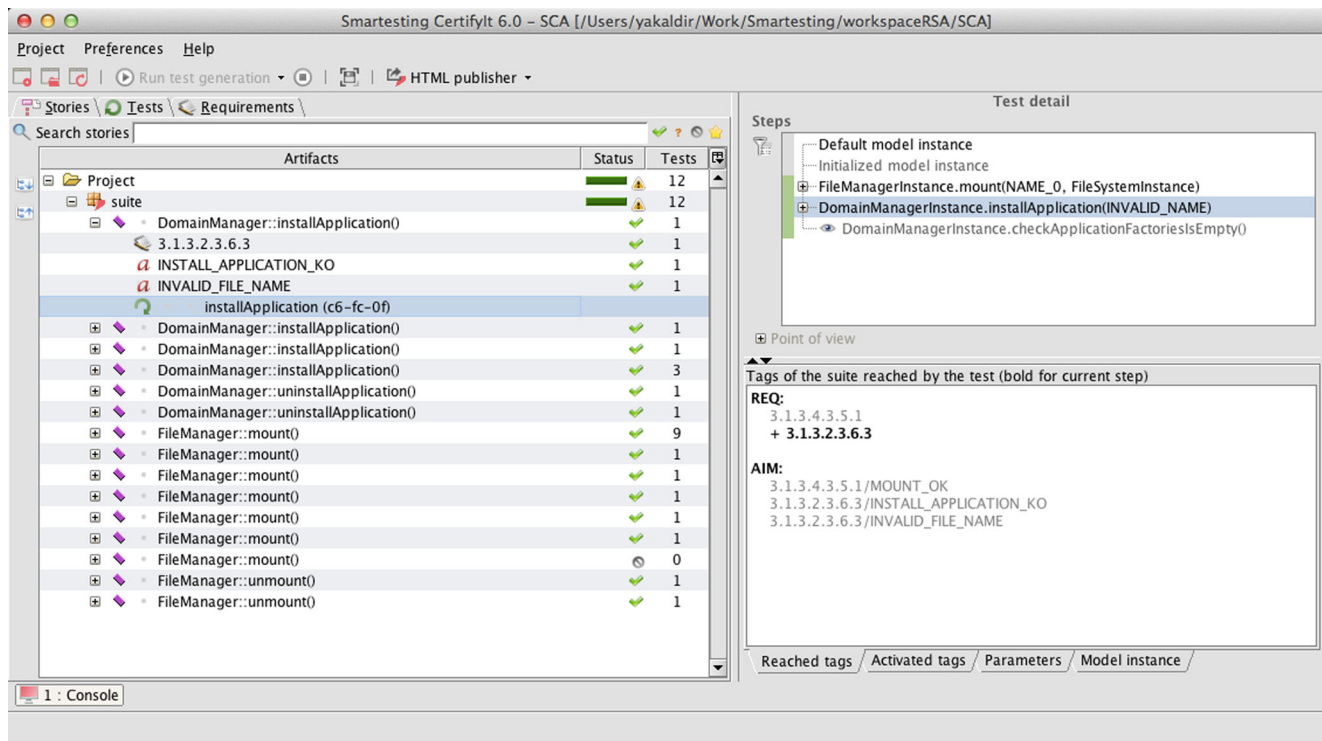


Figure 11 Smartesting *CertifyIt* GUI with Generated Test Cases.

```

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 12 Example of Generated JUnit Test File.

test suite. Figure 15 shows the interface of the JUnit Eclipse environment supporting the management of the executable test suite and its computation to assign the test verdict. In Fig. 15, each line in the left window frame corresponds to a test case and gives its verdict. The green mark means that all the test cases are in success, i.e. the execution gives the expected results with respect to the specifications, and so it demonstrates that the tested scenario is implemented correctly with respect to the SCA specifications. Other frames of the Eclipse interface can be used to display implementation details about the test suite or a selected test case (for

example, in Fig. 15, it shows the code to compute a given test).

4.4 Lessons Learnt from our Experiments

This Model-Based Testing approach has been successfully applied on a subpart of the SCA 2.2.2 specifications, and this project has enabled to implement a fully automated and suitable conformance testing approach for SCA standard. Moreover, the results obtained using this Model-Based Testing process give some valuable benefits regarding the

```

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 13 Example of Generated Java Keyword Library.

conformance testing challenges introduced in Section 1. The main lessons learnt from these experiments are the following:

1. The UML modeling style (the UML4MBT language) is adequate to design such SCA MBT models. The SCA specifications are also precise enough to specify the OCL constraints related both to the functional behaviours of the APIs and to the conformance requirements they have to ensure. The interpretation of the specifications was easy, and no specific problem appeared during the modeling phase.
2. The annotation of the MBT model by SCA requirements (at the level of OCL constraints) is a good way to ensure an appropriate and relevant coverage of the SCA specifications during automated test generation. The requirements identifiers appear in the test scripts in order to ensure the traceability link from the requirements to the automated test. Therefore, this enables

to easily and precisely retrieve which conformance requirements are concerned when a test execution fails.

3. Finally, due to a natural mapping between the modeled operations and the SCA APIs of the SDR platform, automated test execution was straightforwardly managed. More concretely, the (nearly always) one-to-one mapping between the abstract UML operations (resp. attributes and data) and concrete system APIs (resp. variables and values) has made the concretization step very simple.

In the same way as for GlobalPlatform experiments, some other benefits, directly inherited from well-known advantages of the MBT approaches [7] (and already demonstrated on software radio protocol during a previous experiment [13]) 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

Figure 14 Java Implementation of the Keyword Library.

```
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());
        }
    }
}
```

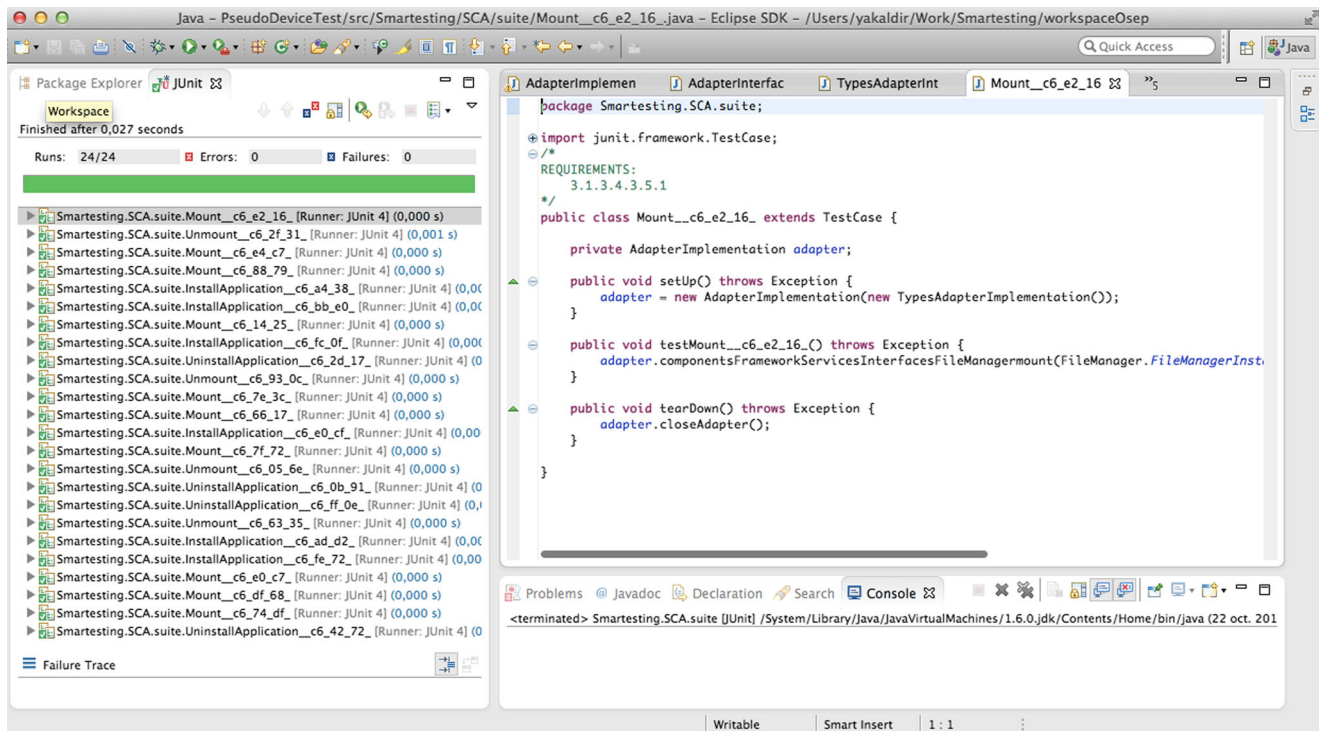



Figure 15 Example of Eclipse Test Execution Report.

based on the same model, they are generated for various implementations, releases, and versions of a single application. This unique reference ensures efficient regression testing and makes easier all maintenance and upgrade activities.

Nevertheless, contrary to the GlobalPlatform experience, this case-study has been conducted as a first experimental evaluation to investigate the technical feasibility and the relevance of applying a Model-Based Testing process to address SCA conformance testing. In this way, we have developed a proof of concept approach, which still requires to be validated in a large-scale context, in particular to assess the effective capacity of the proposed MBT approach to manage possible alternative interpretations and/or implementations of the SCA specifications.

Moreover, even if the experiments on a fragment of the SCA specifications have been successfully conducted and we have obtained results that are undeniably very promising, additional extensive experiments are needed to ensure the approach is effectively suitable for all the features defined by the SCA specifications, and not only for the studied fragment.

Finally, to propose an industrial deployment of this MBT solution, we also need to increase the readiness level of its supporting tooling, especially by offering an enhanced integration of the *CertifyIt* MBT tool with the SDR execution platforms and test beds. To ease modeling, we could also

implement a library of model patterns dedicated to SDR units. It could help engineers to reduce the time spent during the modeling phase, and could allow them to write a test model devoid of noisy specified data from a compliance point of view.

5 Conclusion and Further Work

Since last 10 years, Model-Based Testing (MBT) has seen an increasing interest in different industrial areas 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 softwares and systems. In the context of conformance testing, several deployments of MBT led by industrial consortium (GlobalPlatform for instance) have shown that these techniques may help a standardization organization to better conduct and master their compliance program.

In this paper, within a French research project called OSeP, we investigate and report about a small but successful technical proof of concept of applying Model-Based Testing to validate interoperability between SDR platforms and to assess their conformance with the SCA specifications. To achieve and automate this process, we have developed and extended an existing Model-Based Testing

toolchain in order to manage automated SCA conformance testing generation. Basically, we make use of the Smartest-*CertifyIt* test generator to generate abstract test cases from a UML model specifying the SCA requirements. These test cases are next exported as JUnit test scripts to be executed on the SDR platform under test in order to validate its compliance with the modeled SCA requirements.

Experiments have been conducted on a fragment of the SCA 2.2.2 functional specifications focusing on the “Domain manager” function to “install / uninstall Application” nominally, including exception management. Modeling this functional fragment of the SCA specifications was finally quite simple, and automated test generation techniques have provided the corresponding tests to be run on the SDR platform. Moreover, the use of dedicated annotations in the UML model, to support requirements traceability from the test model to the executable test scripts, makes the solution suitable to precisely validate or invalidate conformance requirements. Therefore, experimentation feedback using this automated Model-Based Testing generation process are very encouraging for SCA conformance testing.

The perspective of this project is to continue to extend the coverage of the functional SCA specifications by the UML model, and so to improve the generated conformance test suite. Indeed, this first empirical evaluation needs to be further investigated by addressing a larger part of the SCA specifications to definitively demonstrate its relevance and effectiveness to achieve SCA compliance testing within large-scale and various SDR deployments. As future work, we also plan to propose a deeper integration of the tools (in particular regarding the link between test generation tool and the execution platform) to increase the readiness level of the solution, and to foster its use by engineers.

Acknowledgments This work has been supported by the ANR ASTRID project OSeP (On-line and Off-line Model-Based Testing of Security Properties, ANR 11 ASTR 002). See <http://osep.univ-fcomte.fr> (last access June 2015).

References

- Bernabeu, G., Jaffuel, E., Legeard, B., & Peureux, F. (2014). MBT for GlobalPlatform compliance testing: Experience report and lessons learned. In *Proceedings of the 25th International Symposium on Software Reliability Engineering (ISSRE'14)* (pp. 66–70). Naples: IEEE Computer Society Press.
- Bernabeu, G., & Lavabre, N. (2013). Model-based testing for a world-wide compliance program. In *1st User Conference on Advanced Automated Testing (UCAAT'13)*, Paris France. http://ucaat.etsi.org/2013/presentations/Keynote_MBT%20for%20a%20Compliance%20Program-GlobalPlatform-GilBernabeu.pdf. (last accessed January 2015).
- Bernard, E., Bouquet, F., Charbonnier, A., Legeard, B., Peureux, F., Utting, M., & Torrebore, E. (2006). Model-based testing from UML models. In *Proceedings of the International Workshop on Model-Based Testing (MBT'06)*, LNI, vol. 94 (pp. 223–230). Dresden: GI.
- Bernard, E., Legeard, B., Luck, X., & Peureux, F. (2004). Generation of test sequences from formal specifications: GSM 11-11 standard case study. *International Journal of Software Practice and Experience*, 34(10), 915–948.
- Bouquet, F., Grandpierre, C., Legeard, B., & Peureux, F. (2008). A test generation solution to automate software testing. In *Proceedings of the 3rd Int. Workshop on Automation of Software Test (AST'08)* (pp. 45–48). Leipzig: ACM Press.
- Bouquet, F., Grandpierre, C., Legeard, B., Peureux, F., Vacelet, N., & Utting, M. (2007). A subset of precise UML for model-based testing. In *Proceedings of the 3rd International Workshop on Advances in Model-Based Testing (AMOST'07)* (pp. 95–104). London: ACM Press.
- Dias-Neto, A., & Travassos, G. (2010). A Picture from the Model-Based Testing Area: Concepts, Techniques, and Challenges. *Advances in Computers*, 80, 45–120. ISSN 0065-2458.
- ETSI: Conformance Testing. <http://www.etsi.org/technologies-clusters/technologies/testing>. (last accessed January 2015).
- Ezick, J., & Springer, J. (2011). The benefits of static compliance testing for sca next. In *Wireless innovation forum conference on communication technologies and software defined radio (SDR-WInnComm'11)*.
- GlobalPlatform (2001). GlobalPlatform Card Specification Version 2.2.1. <http://www.globalplatform.org/specificationscard.asp>. (last accessed January 2015).
- JTNC Standards, Joint Tactical Networking Center, Final/15 V.2.2.2: JTRS/JPEO Software Communications Architecture Specification (2006). <http://jtncc.mil/sca/Pages/default.aspx>. (last access January 2015).
- JTRS Test and Evaluation Laboratory (2010). SCA 2.2.2 Application Requirements List version 2.2 Release Notes. https://jtel.spawar.navy.mil/docs/sca_2.2.2_application_requirements_list.v2.2.pdf. (last access January 2015).
- Li, S., Bourdellès, M., Acebedo, A., Botella, J., & Peureux, F. (2012). Experiment on using model-based testing for automatic tests generation on a software radio protocol. In *Proceedings of the 9th Int. Workshop on Systems Testing and Validation (STV'12)*, pp. 79–84. Paris, France.
- Nguyen, H., Hackett, M., & Whitlock, B. (2006). Global Software Test Automation: A Discussion of Software Testing for Executives. Happy About books. ISBN 1-6000-5011-5.
- Riccobene, E., & Scandurra, P. (2014). A formal framework for service modeling and prototyping. *Formal Aspects of Computing*, 26(6), 1077–1113.
- Rumbaugh, J., Jacobson, I., & Booch, G. (2005). *The Unified Modeling Language Reference Manual*, 2nd. Addison-Wesley. ISBN 0-3212-4562-8.
- Seignole, V., Hachet, O., Council, B., & Balp, H. (2010). Method and system for encapsulating a plurality of software components compatible with the CCM standard into a software standard compatible with the SCA standard. WO Patent App. PCT/EP2009/065,831. *Google Patents*.
- Utting, M., & Legeard, B. (2006). Practical Model-Based Testing - A tools approach. Morgan Kaufmann, San Francisco, CA, USA. ISBN 0-1237-2501-1.
- Warmer, J., & Kleppe, A. (1999). *The Object Constraint Language: Precise Modeling with UML*, 2nd. Addison-Wesley. ISBN 0-2013-7940-6.
- Zhu, H., & Belli, F. (2009). Advancing test automation technology to meet the challenges of model-based software testing. *Information and Software Technology*, 51, 1485–1486. ISSN 0950-5849.



Julien Botella received his Master of Science degree in Computer Science in 2006 from the University of Franche-Comté. From 2006 to 2008, he was R&D Engineer at the Inria (French Research Institute for Computer Science and Applied Mathematics) in Besançon. From 2008, he is technical leader and senior consultant at Smartesting. He is an expert in automated test generation and execution approaches with a large experience of developing and

automating Model-Based Testing techniques for both functional and security testing.



Jean-Philippe Delahaye received his MS in Telecommunications from Telecom ParisSud and his Master Degree in Electronics from the University of Paris 11, both in 2003. He received in 2007 his Ph.D. Degree in Electronics from the University of Rennes 1. In 2007, he joined the French armament procurement agency in Information Warfare Technology Center as a technical system engineer in software radio (SDR). He is currently the expert for the

French *Programme CONTACT* working in SDR standardization. His interests include middleware and software architectures for embedded systems, component-based software engineering and testing.



Eddie Jaffuel received his engineering degree in Software Engineering from ENSIMAG (Mathematics and Computer Sciences) in 1998. He worked as a consultant at Smartesting during 8 years. In 2012, he founded the independent service company *eConsult*, specialist in test production with Model Based Testing solution. Eddie Jaffuel is an expert in modeling and in Model-Based Testing solution to various domains (automotive, smartcards, telecom,

multi-component systems, etc.).



Bruno Legard is professor of Software Engineering at the University of Franche-Comté and Scientific advisor for Smartesting Solutions & Services. He is co-author of the book “*Practical Model-Based Testing - A tools approach*” (Elsevier) published in 2006. He started working on Model-Based Testing in the mid 1990’s and has extensive experience in applying Model-Based Testing to large information systems, e-transaction applications and

embedded softwares. He is member of the French Software Testing Committee Board, and co-leads the Model-Based Testing certification within the International Software Testing Qualification Board (ISTQB).



Fabien Peureux received his Ph.D. degree in Computer Science from the University of Franche-Comté in 2002, where he works since 2003 as assistant professor and does his research activities with the FEMTO-ST Institute. Since 2005, he is also senior scientific consultant for the Smartesting company. His main expertise is focused on the automation of validation process in the domains of smartcard applications, information systems

and embedded softwares, with a particular interest in Model-Based Testing techniques and agile approaches.