IoT Interoperabity Model-Based Testing

Case study – FIWARE Generic Enabler

Authors: Abbas AHMAD, Frank Le Gall, Bruno Legeard

Confidentiality level: Public, All Access

Date: July 2015

Version: 2.0

Table of contents

INTRODUCTION 3
1 The MBT process we used 7
1.1 The MBT Approach 7
1.2 Tool Chain and Test environment 8
2 Espr4FastData Test Objectives 10
2.1 Espr4FastData Specification 10
2.2 Test objectives 10
3 MBT modeling for Espr4FastData 11
4 Test Generation 13
5 Test Adaption and Test Execution 16
6 Project Metrics and Discussion 18
7 Conclusion and way foward 19
8 Bibliography 20

INTRODUCTION

Models are described in many ways, depending on the discipline. They can be described by use of diagrams, tables, text, or other kinds of notations. They might be expressed in a mathematical formalism or informally, where the meaning is derived by convention.

Although computing equipment and software plays an important role for the application of modeling in science and engineering, modeling is not as ubiquitous in software engineering as it is in other engineering disciplines. However, formal methods is one discipline where modeling has been applied in the areas of safety- and security-critical software systems.

Test phases are the last stages of system development. They are therefore strongly constrained by delivery periods. In general, they are subject to all of the delays accumulated during the overall project. Currently, test phases still correspond to 30% or even to 50% of the total cost of system development and thus represent an important point for possible improvements.

Currently, the verification & validation process having no systematic or automatic process is often underestimated and outsourced to reduce costs. All these factors affect the quality of current products.

Testing is a difficult activity. Indeed, it is not possible to test everything because the quantity of test cases to be applied seems infinite for the majority of modern systems. The difficulty of testing lies in the selection of relevant test cases for each phase of validation. It also lies in the absence of a true reflection of test optimization guided by the maintenance of the final quality while minimizing the costs.

We present hereafter a technology which is more and more used in industry to avoid bugs, to improve quality and reduce costs: Model-Based Testing.

Model-based testing (MBT) is the automatic generation of software test procedures, using models of system requirements and behavior. Although this type of testing requires more up-front effort in building the model, it offers substantial advantages over traditional software testing methods:

- Rules are specified once.
- Project maintenance is lower. There is no need to write new tests for each new feature. Once we have a model it is easier to generate and re-generate test cases than it is with hand-coded test cases.
- Design is fluid. When a new feature is added, a new action is added to the model to run in combination with existing actions. A simple change can automatically ripple through the entire suite of test cases.
- Design more and code less.
- High coverage. Tests continue to find bugs, not just regressions due to changes in the code path or dependencies.
- Model authoring is independent of implementation and actual testing so that these activities can be carried out by different members of a team concurrently.

Model-Based Testing - Importance:

- Unit testing won't be sufficient to check the functionalities
- To ensure that the system is behaving in the same sequence of actions.
- Model-based testing technique has been adopted as an integrated part of the testing process.
- Commercial tools are developed to support model-based testing.

In this work we are interested in the Internet of Things (IoT) Services Enablement Generic Enablers (GEs) of FIWARE, and more specifically in verifying their compliance to the specifications. The FIWARE platform [1] provides a rather simple yet powerful set of APIs (Application Programming Interfaces) that ease the development of Smart Applications in multiple vertical sectors. The specifications of these APIs are public and royalty-free. Besides, an open source reference implementation (Generic Enabler implementation) of each of the FIWARE components is publicly available so that multiple FIWARE providers can emerge faster in the market with a low-cost proposition.

Generic Enablers offer a number of general-purpose functions and provide high-level APIs in:

- Cloud Hosting
- Data/Context Management
- Internet of Things (IoT) Services Enablement
- Applications, Services and Data Delivery
- Security of data
- Interface to Networks and Devices (I2ND)
- Advanced Web-based User Interface

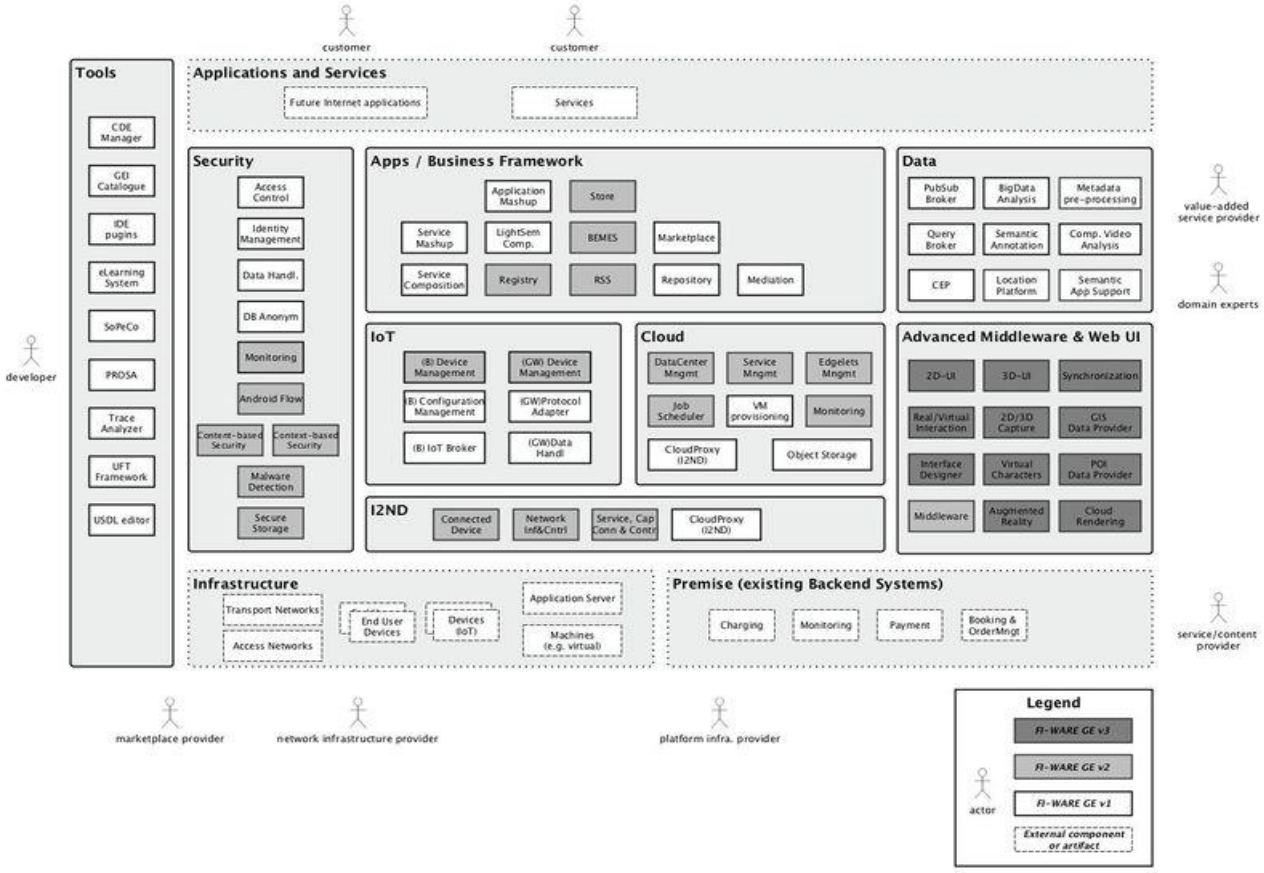

Figure 1 Schematic depiction of FIWARE platform with all major chapters

The IoTs are spread in two different domains: Gateway and Backend. While IoT Gateway GEs provide inter-networking and protocol conversion functionalities between devices and the IoT Backend GEs, the IoT Backend GEs provide management functionalities for the devices and IoT domain-specific support for the applications.

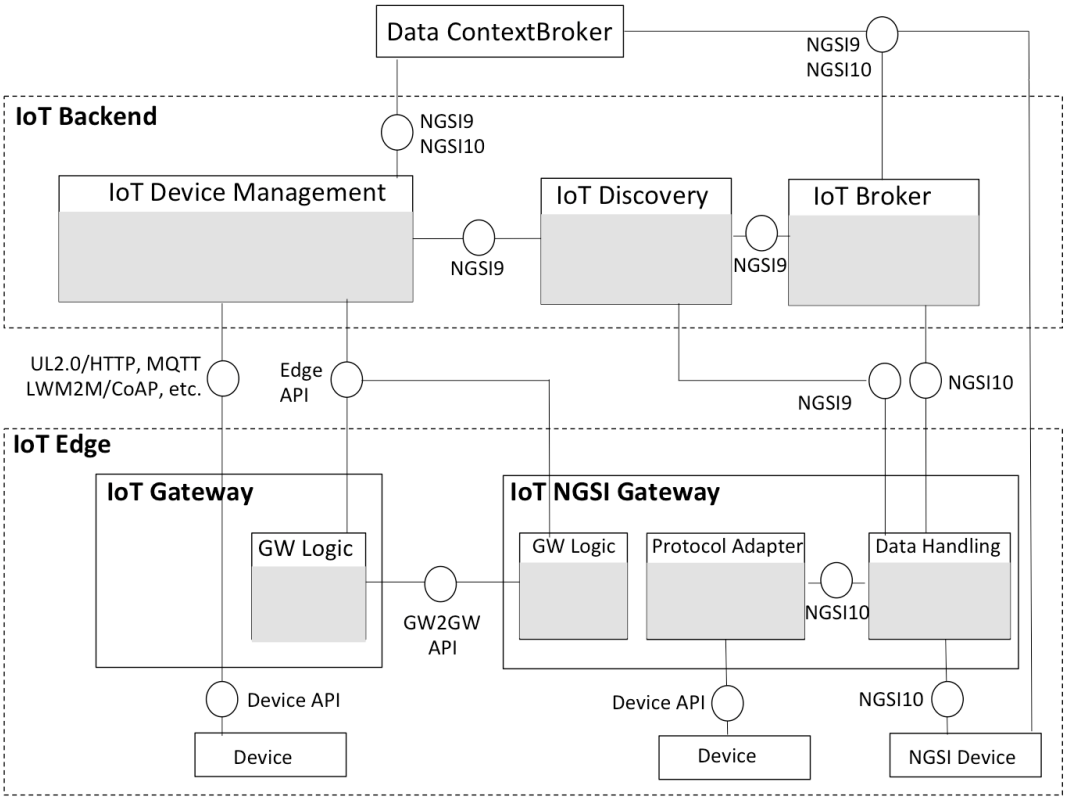

Figure 2 Complete IoT Architecture

Espr4FastData is a data handling GE.

The FIWARE version of the OMA NGSI-9/10 interfaces [2] are a RESTful APIs. RESTful systems communicate over the Hypertext Transfer Protocol with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) which web browsers use to retrieve web pages and to send data to remote servers. Their purpose is to exchange information about the availability of context information.

The three main interaction types for the NGSI-9 are:

- one-time queries for discovering hosts where certain context information is available
- subscriptions for context availability information updates (and the corresponding notifications)
- registration of context information, i.e. announcements that certain context information is available (invoked by context providers)

The three main interaction types for the NGSI-10 are:

- one-time queries for context information
- subscriptions for context information updates (and the corresponding notifications)
- unsolicited updates (invoked by context providers)

The data handling GE is NGSI-9/10 compliant. Moreover, although it is often assumed that the GE are correctly implemented and are used or are part of services provided by FIWARE, non-compliance to FIWARE specifications may lead to a number of dysfunctional interoperability issues. To ensure the interoperability, GEs should be thoroughly tested to assess their compliance to these specifications.

Model-Based Testing (MBT) is considered to be a lightweight formal method to validate software systems. It's formal because it works out of a formal (that is, machine-readable) specification (or model) of the software system one intends to test (usually called the implementation or System Under Test, SUT). It's lightweight because, contrary to other formal methods, MBT does not aim at mathematically proving that the implementation matches the specifications under all possible circumstances. What MBT does is to systematically generate from the model a collection of tests (a "test suite") that, when run against the SUT, will provide sufficient confidence that it behaves as the model predicted it would.

So the difference between lightweight and heavyweight formal methods is basically about sufficient confidence vs. complete certainty. Now, the price to pay for absolute certainty is not low, which results in heavyweight formal methods being very hard (sometimes prohibitively hard) to apply to real-life projects. MBT on the other hand scales much better and has been used to test life-size systems in huge projects [3].

In this paper, we give an overview of our industrial MBT approach applied for compliance testing of Generic Enablers. We organize this paper in seven sections. Section 1 introduces the MBT approach we used and the created test environment. Section 2 defines the scope of our study: the Espr4FastData specification and its scope under study, the system under test (SUT) and as well the test objectives identified from the specification. Sections 3, 4 and 5 discuss the modeling for testing, the test generation and the test adaptation and execution, respectively illustrated through the case study. Section 6 discusses the study's metrics and test generation and execution results, before concluding in Section 7.

1 THE MBT PROCESS WE USED

The fundamental MBT process [4] includes activities such as test planning and controls, test analysis and design (which includes MBT modeling, choosing suitable test selection criteria), test generation, test implementation and execution.

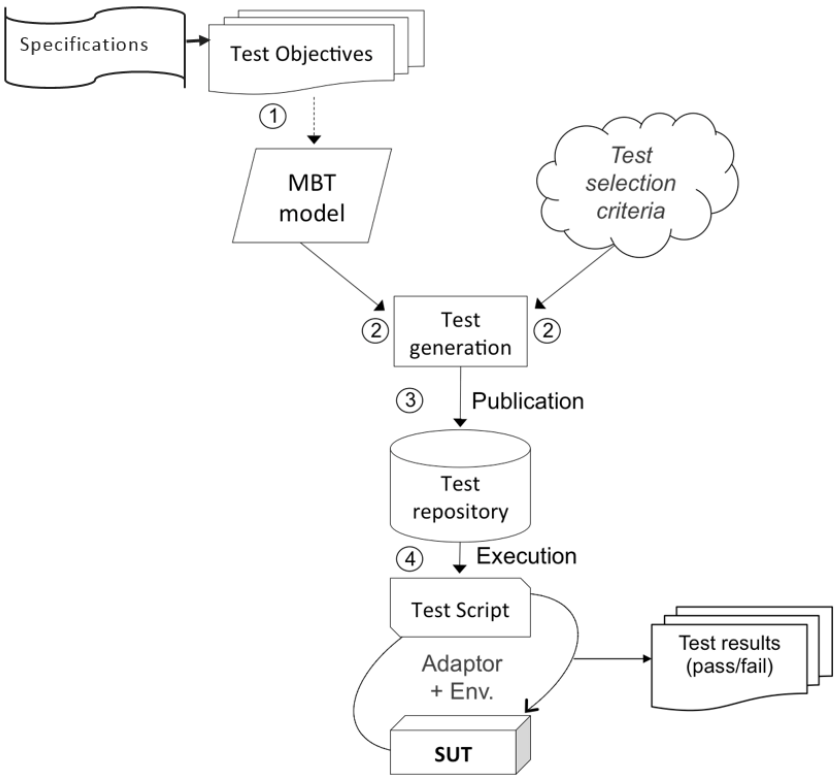

Figure 3. MBT Process

In this section we present the MBT process we used for testing a FIWARE generic enabler implementation: Espr4FastData. As given in Figure 3, in a classical MBT process, test analyst takes requirements and defines the test objectives as input to model the System Under Test (SUT) (step 1). This MBT model contains static and dynamic views of the system. Hence, to benefit as much as possible from the MBT technology, we consider an automated process, where the test model is sent as an input to the test generator that automatically produces abstract test cases and a coverage matrix, relating the tests to the covered model elements or according to another test selection criteria (step 2). These tests are further exported, automatically (step 3), in a test repository to which test scripts can be associated. The automated test scripts in combination with an adaptation layer link each step from the abstract test to a concrete command of the SUT and automate the test execution (step 4). In addition, after the test execution, tests results and metrics are collected and feedback is sent to the user.

1.1 THE MBT APPROACH

From one MBT model various test selection criteria can be applied, as shown in Figure 3. For our study we used the Smartesting MBT approach and its test generation tool - CertifyIt, which is based on coverage-based test selection criteria.

This approach considers a subset of UML to develop MBT models, composed from two types of diagrams for the structural modeling:

- class diagrams
- object diagrams

Each type has a separate role in the test generation process. The class diagram describes the system's structure, namely the set of classes that represents the static view of the system:

- Its entities, with their attributes
- Operations that model the API of the SUT
- Observations that serve as oracles (for instance an observation returns the current state of the user's connection to a web site)

Next, this static view, given by the class diagram, is instantiated by an object diagram. The object diagrams provides the initial stat of the system and also all objects that will be use in the test input data as parameters for the operations in the generated tests. Finally, the dynamic view of the system or its behaviors are described by OCL constraints written as pre/postcondition in operations in a class of a class diagram. The test generation engine sees these behavior objects as test targets. The operations can have several behaviors, identified by the presence of the conditional operator if-then-else. The precondition is the union of the operation's precondition and the conditions of a path that is necessary to traverse for reaching the behavior's postcondition. The postcondition corresponds to the behavior described by the action in the "then" or "else" clause of the conditional operator. Finally, each behavior is identified by a set of tags (as initially defined in the test objective charter - TOC), which refers to a requirement covered by the behavior. For each requirement, two types of tags exists:

- @REQ - a high-level requirement
- @AIM- its refinement

Both followed by an identifier. Finally, one major advantage of the CertifyIt approach is the possibility to automatically deduce the test oracle[1]. A specific type of operations, called observations, define the test oracle. The tester with these special operations can define the system points or variables to observe, for instance a function return code. Thus, based on these observations, the test oracle is automatically generated for each test step.

1.2 Tool Chain and Test environment

We depict the tool chain used for interoperability model-based testing of FIWARE GE in the figure below.

SoapUI [5] is a free and open source cross-platform Functional Testing solution. With an easy-to-use graphical interface, and enterprise-class features, SoapUI allows to easily and rapidly create and execute automated functional, regression, compliance, and load tests. The test environment as given at the figure contains: the SoapUI software where we will import the generated tests. After the test execution, the

[1] Test oracle is a source to determine expected results to compare with the actual result of the software under test.

pass/fail test results are determined by matching the test plan doc observation and the SoapUI output response.

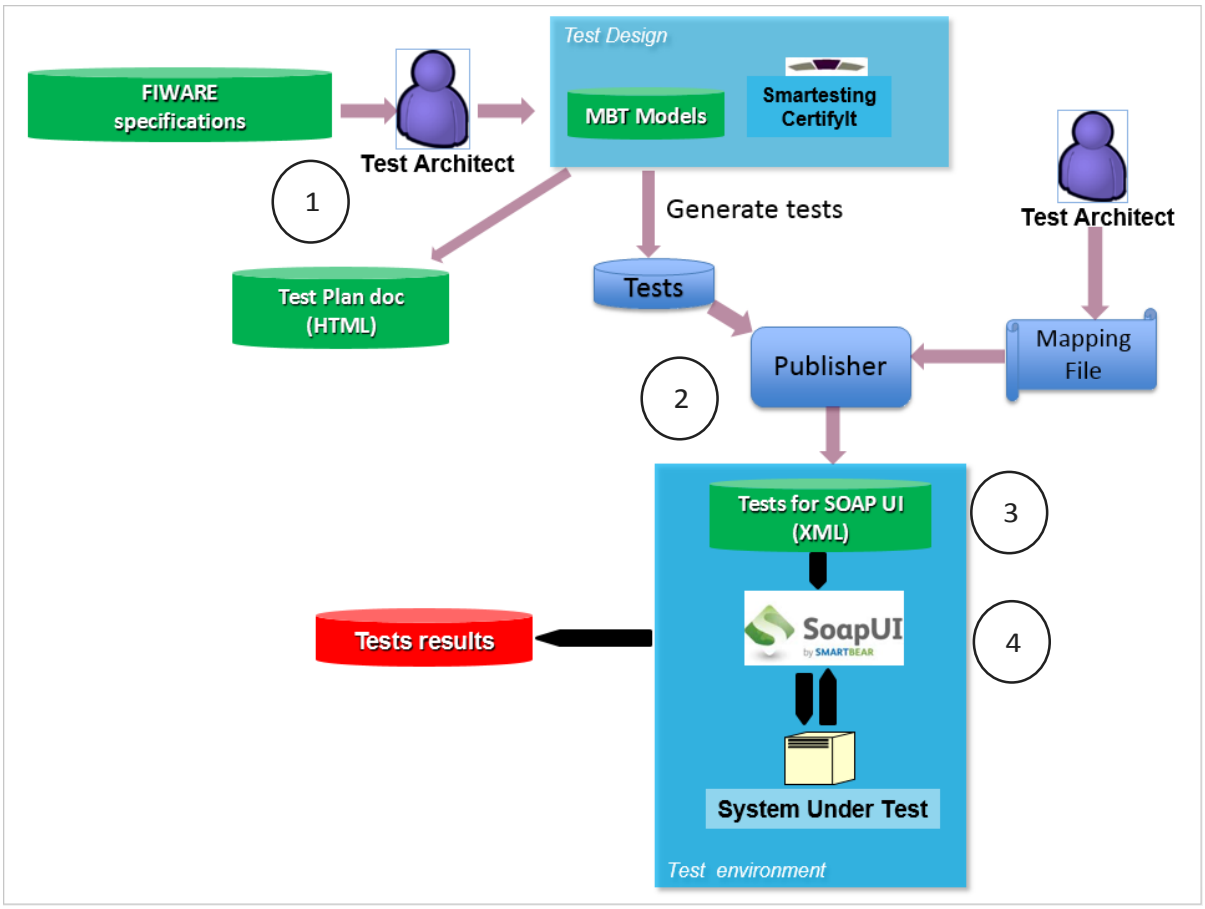

Figure 4. Espr4FastData Test Environment

The system under test for our project is installed on a Linux mint virtual machine. But note that as we are testing FIWARE Generic Enabler implementation in the IoT chapter, they should be NGSI9/10 compliant and therefore, the SUT can be installed anywhere as long as we have access to it through a reachable IP address (RESTful API). SoapUI is installed on a windows 7 OS.

2 ESPR4FASTDATA TEST OBJECTIVES

2.1 ESPR4FASTDATA SPECIFICATION

The Data Handling GE addresses the need to process data in real time. Frequently implemented features include filtering, aggregating and merging real-time data from different sources. Thanks to Complex Event Processing (CEP), it is easy for applications to only subscribe to value-added data, which is relevant to them. CEP technology is sometimes also referred to as event stream analysis, or real time event correlation. As seen mentioned in the introduction Espr4FastData is NGSI9/10 compliant, this means that the GEi should implement all their functions.

2.2 TEST OBJECTIVES

In an MBT approach, it is a good practice to define the perimeter and from it, identify the test objectives to be covered by tests. In this way, the test objectives delimit the model subject and focus.

As given in Table 1, for each function we define a set of test objectives. Each test objective is composed of a high level requirement, which is the function name denoted as *@REQ*, and its refinement, which is an expected behavior of the function, denoted as *@AIM*. The total number of the test objectives for the considered sub-set of Espr4FastData is 18.

In addition, the MBT model will be based on these test objectives, by linking each requirement/behavior in the function with the corresponding *@REQ* and *@AIM*, thus ensuring the bi-directional traceability between the requirements and the generated tests.

@REQ	Requirement description	#AIM		
RESET_GE	This operation completely destroy the CEP threads, the CEP data, and all the application level data. It can be used whether the application is running or not. It leaves the application clean and ready to start. EXAMPLE : http://localhost/EspR4FastData-3.3.3/admin/espr4fastdata (queried with HTTP Delete method)	success	Error Invalid URL	success and is Empty
REGISTER_CONTEXT	This operation registers a NGSI context within EspR4FastData. This makes an input event type and related entities to be known from an EspR4FastData perspective. An event is characterized by the involved entities (eg: Van1, Van2, Plane), its type, and properties that related to the type. For an event to be processed, it is mandatory to be "known" by the application. Otherwise it would be rejected.	success	Context already exist, Error rejected	Error Invalid URL

Table 1 Test Objective Charter example for Espr4FastData

3 MBT MODELING FOR ESPR4FASTDATA

The MBT model for Espr4FastData focuses on the test objectives we have defined in the TOC (see Section 1). A class diagram represents the static view of the Espr4FastData API. In the following figure we give a simplified view of MBT model which contains four classes: *Sut (Espr4FastData)*, *CepStatment*, *RegisteredThing*, *Subscription*. We represent the API *Espr4FastData* that offers RESTful interfaces for communicating with entities, modeled by the class *RegisteredThing*. The user can register Complex Event Processing (CEP) statements which are identified by an entity id. Finally the user can subscribe to the updates of regular entities and CEP entities which results in the creation of a subscription id.

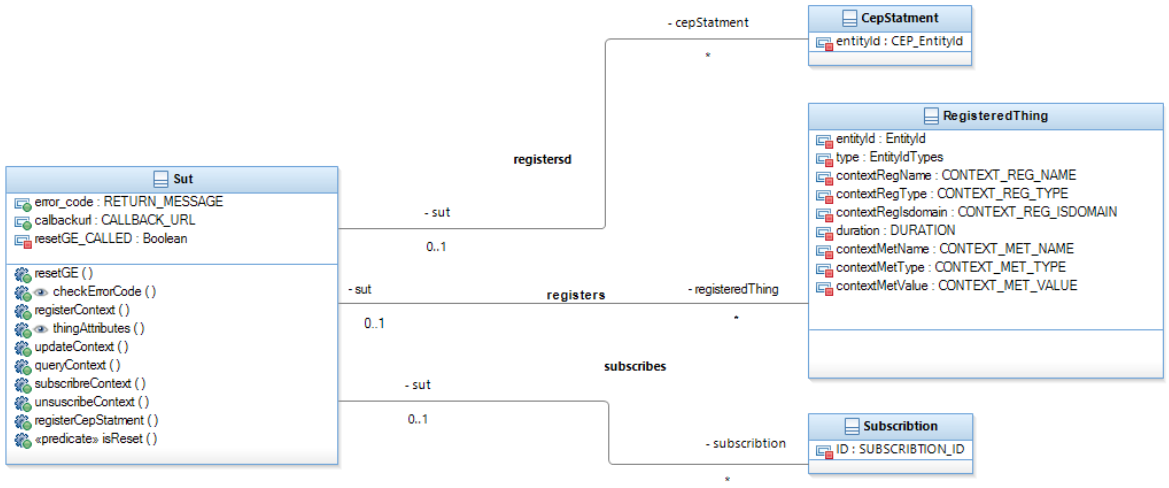

Figure 5. Espr4FastData simplified class diagram

The dynamic view of the Espr4FastData API is modeled using Object Constraint Language (OCL) postcondition in the Espr4FastData functions. An OCL postcondition captures the test objectives for each function given in the TOC. Further, using the tagging system with @REQ and @AIM, as discussed previously in Section 1, we annotate each behavior in the postcondition. For example, the reset function "*resetGE*" has three test objectives, as given in Table 1: successful execution of the signing reset function, successful and is empty, finally error in the URL given as no payload is required thus error can only come from the url. In the following figure we represent them respectively as an OCL postcondition. As shown in the figure below, each @AIM is identically tagged in the postcondition, thus after test generation the CertifyIt tool is able to export a report documenting the coverage of each test objective by the generated tests.

EspR4FastData_GETestSuite | resetGE

```
1---@REQ: resetEspR4FastData
2if IN_URL = CALLBACK_URL::INVALIDURL
3then
4    ---@AIM: Invalid url
5    self.error_code = RETURN_MESSAGE::URL_ERROR
6else
7    if not self.registeredThing->isEmpty()
8    then
9        ---@AIM: success
10       self.registeredThing->isEmpty() and
11       self.error_code = RETURN_MESSAGE::CODE_200
12   else
13       ---@AIM: success and isEmpty
14       self.error_code = RETURN_MESSAGE::CODE_200
15   endif
16endif and
17result = self.error_code and
18self.resetGE_CALLED = true and
19checkErrorCode()
```

Figure 6. OCL postcondition for resetGE

Furthermore, to ease the test concretization we fulfil a documentation for each function, which consists in giving as much as precise instructions for the test concretization activity.

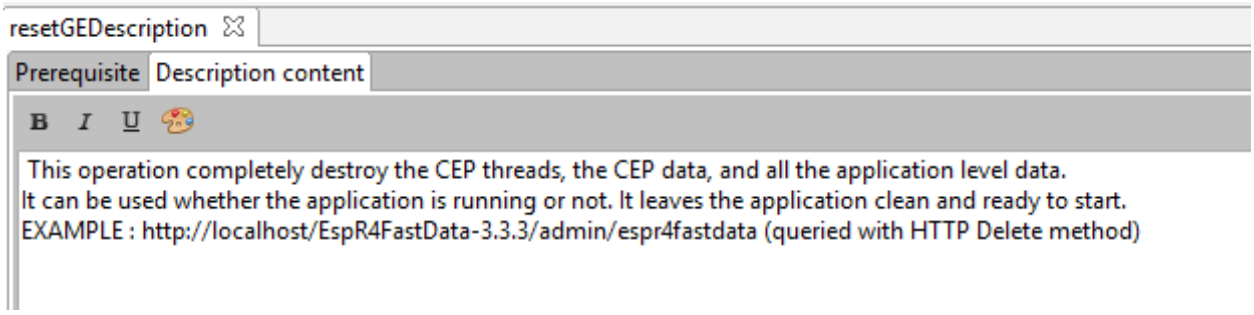

Figure 7. Example of function description, resetGE

As depicted in Figure 7, we consider a documentation for the function "*resetGE* ", which will guide the test engineer work on the adaptation layer, when concretizing the tests. We will discuss further the test adaptation and execution in Section 5.

4 TEST GENERATION

The chosen behavioral test selection criteria drive the CertifyIt test generation engine. The test generation activities, using the CertifyIt tool, consist to create an object diagram, and then a SmartSuite.

The object diagram represents the initial state of the system under tests and contains the input data based on partition equivalence principle. The input data are instance of the classes and they are represented in the form of instances (objects), as introduced previously in Section 2 Each input data is identified in general by an enumeration, representing an equivalent partition of a data from which test could be derived. For instance, as shown in Figure 8, to test the Espr4FastData API, we may consider an initial state of the system with a *Sut* having a nothing connected to it and some entities to be used for registering, creating CEP statements, subscription, etc...

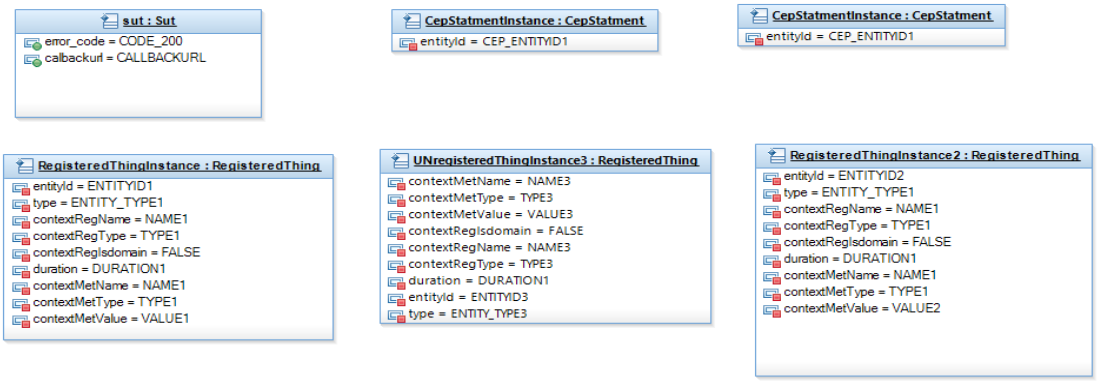

Figure 8. Excerpt of the Espr4FastData Object Diagram

The next step is to create a SmartSuite. In our study we have created two test suites named "EspR4FastData_GETestSuite" and "UserTestScenarios". The EspR4FastData_GETestSuite indicates that the generation should be done for all defined test objectives and as the UserTestScenarios SmartSuite is named, we have user scenarios tests in order to test some very specific cases.

Based on the object diagram and the OCL postcondition the CertifyIt tool extracts automatically a set of test targets, which is the tool comprehensible form of a test. The test targets are used to drive the test generation.

As discussed previously each test has a set of tags associated for which it will ensure the coverage. Figure 9, gives a snapshot of a test in the CertifyIt tool. On the left side, the tool lists all generated tests clustered per covered requirement. On the right side, we can visualize the test case and for each step a test oracle is generated. As discussed, the tester with the observation manually defines the system point to observe when calling any function. As we can see on the figure, "*checkResult*" observes the return code of each Espr4FastData function with respect to the activated requirement. In addition, on the right-bottom of the figure for each test case and test step it is possible to observe the test targets (set of tags).

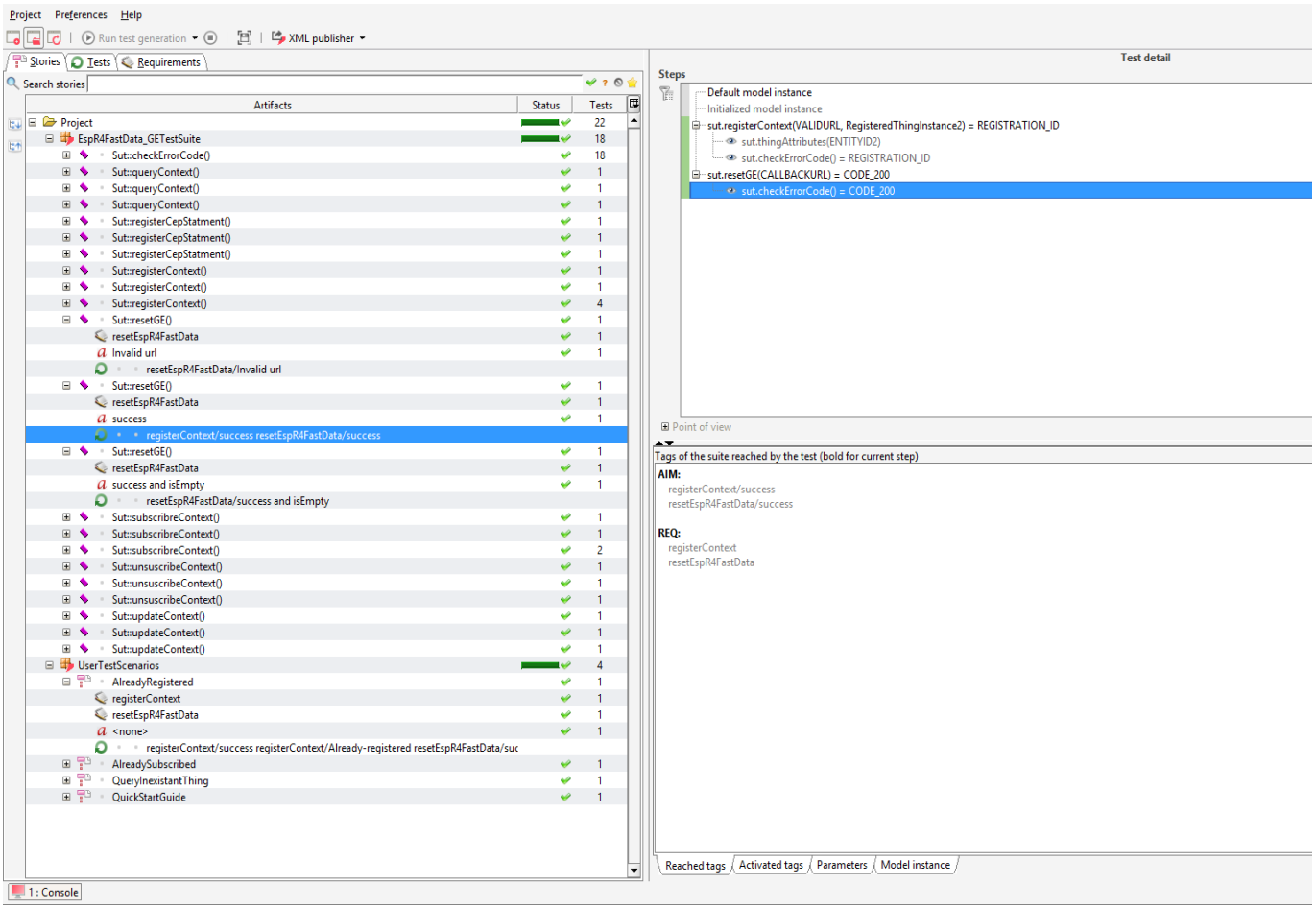

Figure 9. CertifyIt test case view

In addition, Figure 10 illustrates the generated tests with CertifyIt, in an HTML export. More specifically, this test covers the behavior of resetting the GEi with an URL error using the "*resetGE*" function. The export, as shown in the middle column, contains guidelines for test adaptation, further detailed in Section 5.

As discussed previously each test has a set of associated tags for which it will ensure the coverage. The export of this test, as illustrated in Figure 10, in the most right column, maps to each step the covered requirements or more precisely the test objective tagged by @REQ and @AIM, initially defined in the test objective charter (for more details refer to Section 1).

Steps	Actions	Requirements, aims and custom tags
Step 1 *(sut)*	*resetGE* This operation completely destroy the CEP threads, the CEP data, and all the application level data. EXAMPLE : http://localhost/EspR4FastData-3.3.3/admin/espr4fastdata (queried with HTTP Delete method)	REQ resetEspR4FastData AIM resetEspR4FastData/Invalid url
1.1	*@synthesis@* *Check that the error code is URL_ERROR* *@/synthesis@* *Check if an error code URL_ERROR is returned*	

Figure 10. Espr4FastData Abstract Test Case (HTML export)

Within the considered perimeter of Espr4fastData, the tool extracted 26 "*test targets*" and generated 22 tests in around 10 seconds to cover the test targets. Each story in the CertifyIt tool corresponds to one test target. However, one test covers one or more test targets. Moreover, the tool's engine generates fewer tests then test targets, because it uses the "light merge" of tests method, which considers that one test is covering one or more test objectives (all test objectives that have been triggered by the test steps). For instance, the

test shown in Figure 9 covers two test objectives. The "light merge" of tests shortly and on a high level means that the generator will not produce separate tests for the previously reached test targets(test steps 1, 2, 3 and 4), but will consider the test targets from steps 1, 2, 3 and 4 as covered by this test.

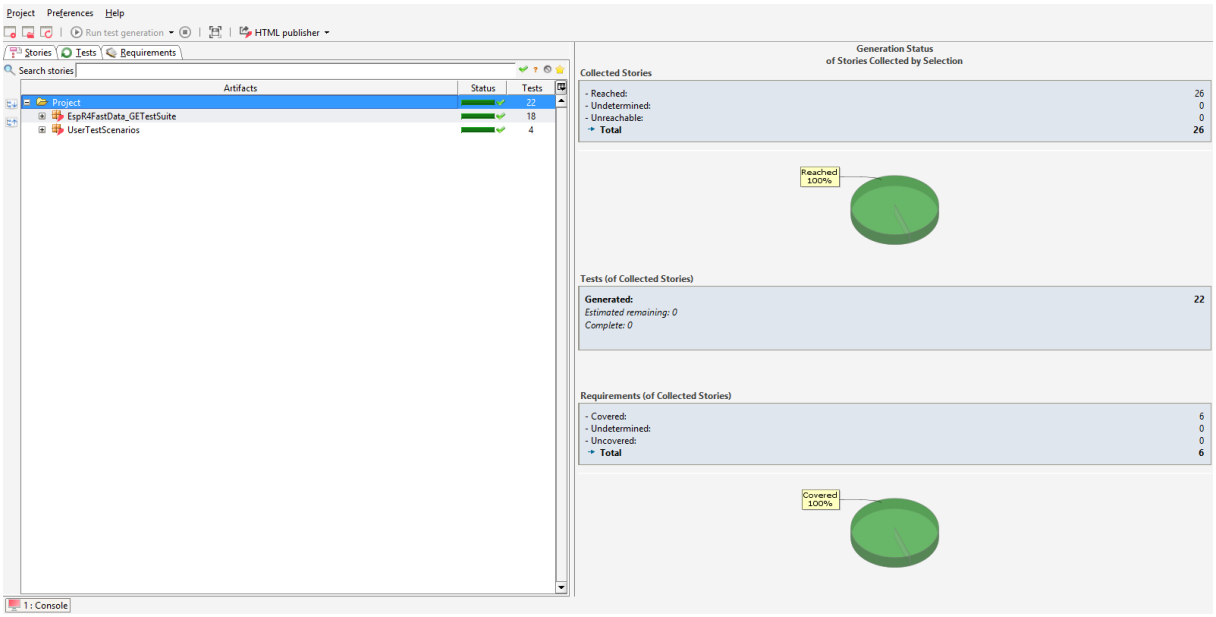

Figure 11. CertifyIt - generated tests for Espr4FastData

The generated tests are abstract and to execute them on the system they should be further *adapted*. We discuss the test adaptation and execution activities in the following section.

For our study we have created a SoapUI exporter, which publishes tests into SoapUI XML projects using the SoapUI library.

5 TEST ADAPTION AND TEST EXECUTION

In the used MBT approach, to execute tests on the system under test, we perform first the activity of test adaptation, which consists to create an adaptation layer to fulfill the gap between the abstract test case, generated from the MBT model, and the concrete interface and initial set of test data of the system under test.

As classically required for the test adaption activities, we first exported the abstract test cases into executable script, in our case tests in XML that can be imported and executed on SoapUI.

To illustrate the test adaptation activities, let us consider the following simple test case given in Figure 11:

$$sut.resetGE(INVALIDURL)$$

$$\hookrightarrow sut.checkErrorCode\ () = URL\_ERROR$$

Figure 12. Simple abstract test case for Espr4FastData

This test has one step that calls the function to reset the data with a URL "INVALIDURL". CertifyIt generates automatically the test oracle, by using an observation operation defined in the model, in our example called "*checkErrorCode* ". For instance, the expected result of the test is the error –*URL_ERROR*.

Further, we created a SoapUI exporter, which from the CertifyIt tool publishes the abstract test cases into concrete SoapUI project executable on SoapUI. In Figure 13, we present an excerpt of the Espr4FastData exporter, illustrating the export of the previous simple test case. The exporter is generic for FIWARE NGSI9/10 implementations, thus could be reused for other compliant enablers.

```
private void createSoapUITestStep(Collection<Argument> argList, Collection<Argument> thingAttributes, WsdlTestCase tc, String opName) {
    String endpoint_url = "";
    String result = "";
    for (Iterator<Argument> iterator = argList.iterator(); iterator.hasNext();) {
        Argument arg = (Argument) iterator.next();
        //          System.out.println(arg.getParameter().getName());
        if(arg.getParameter().getName().equals(Mapping.getString("IN_URL_MODEL"))){
            endpoint_url = arg.getValue().getName();
        }
        if(arg.getParameter().getName().equals(Mapping.getString("result_MODEL"))){
            result = arg.getValue().getName();
        }
    }

    if(opName.equals("teardown")){
        endpoint_url = "CALLBACKURL";
        result="ResetStateOfGE";
    }
    tc.addTestStep(HTTP_RQST, opName+"_"+result, ipPortVersionURL+Mapping.getString(opName+"_"+endpoint_url), Mapping.getString(opName+"_method"));

}
```

Figure 13. Excerpt of the Espr4FastData exporter

A mapping file exist in order to change the values from literal to real values.

To execute the tests on the test harness, we import and lunch the produced XML file in the SoapUI software.

The figure 14 below shows an example of pass/fail result after importing the SoapUI project in the software and executing it. We have a clear view of the passed tests as they are in green and the fail tests in red.

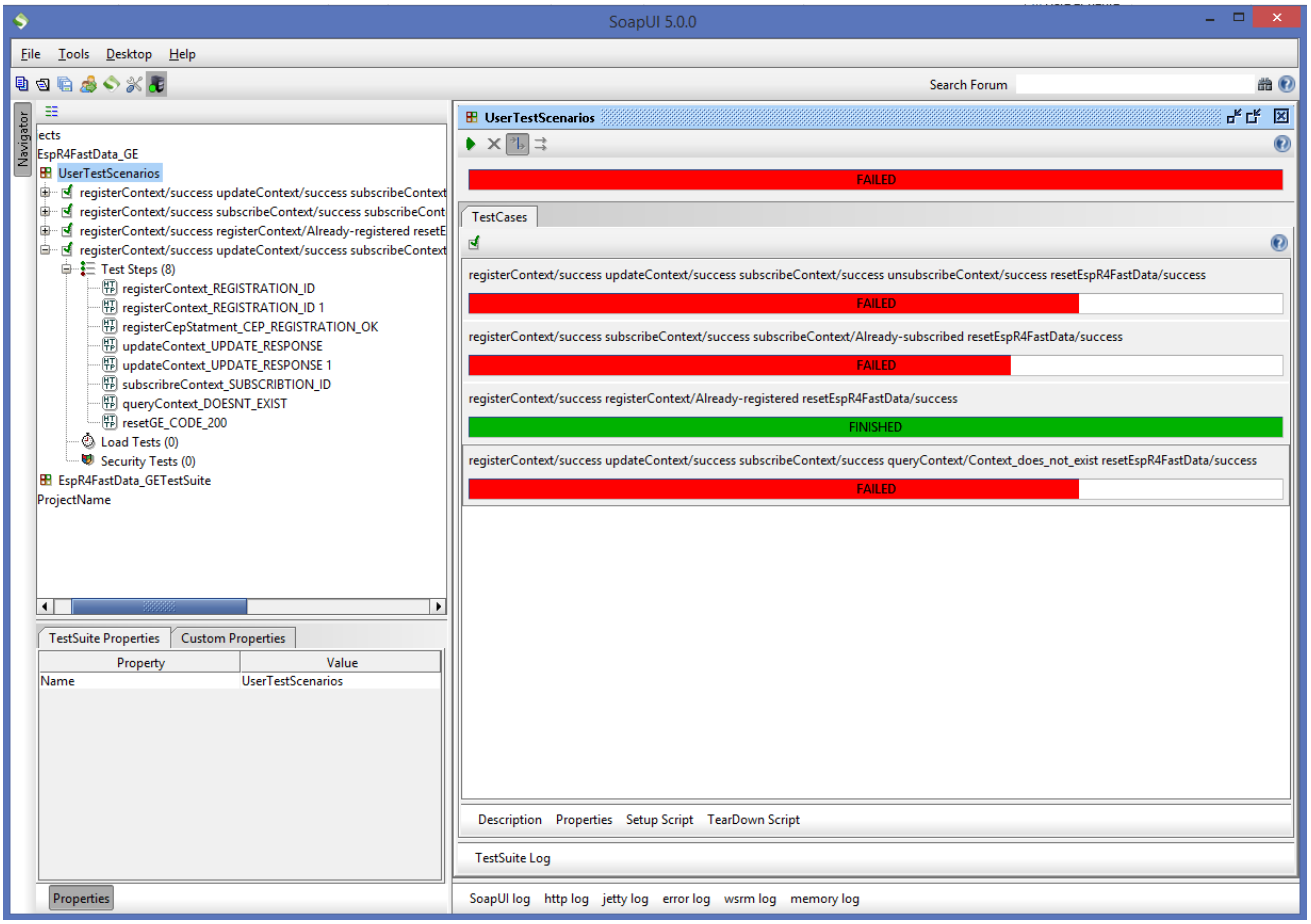

Figure 14 SoapUI Test case Launch example with result status

6 Project Metrics and Discussion

It was a great opportunity to apply an MBT approach on an IoT service enablement API. We were thus able to see the benefits of applying an MBT approach in terms of improving the API's interoperability thus ensuring the respect of the specifications.

In terms of project planning, it took us 16 person/hours to create the MBT model. More specifically, it took 4 person/hours to model the static view of the system (the class diagram) suitable for testing and 12 person/hours to model the dynamic view of the system (to write the OCL postconditions).

These metrics abstract away the domain knowledge on FIWARE standards and the Espr4FastData specification itself. If the MBT approach is integrated within the project, the testing teams have already this knowledge.

In addition, the time spent to create the adaptation layer is not measurable, since we didn't have the knowledge of the SoapUI API. This is linked to the developers/testers experience and we consider it as negligible.

One major advantage we saw in applying the Smartesting MBT approach on EspR4FastData is that the test repository remains stable, while the project requirements and specification may evolve within the time.

Another advantage we saw concerns the testing exhaustiveness. It is indeed impossible to test any system exhaustively. Hence, being able to generate tests automatically and in a systematic way, as we did, makes possible to constitute a test suite covering the defined test objectives. The CertifyIt tool further allows generating reports to justify the test objective coverage, which can be easily used for auditing for instance.

These couple of examples show the usefulness of an automated and systematic use of an MBT approach on applications that should comply to specific standards.

7 CONCLUSION AND WAY FOWARD

This chapter presented a successful application of an MBT approach using Smartesting CertifyIt tool. We believe that this approach can be generally applied on a wide range of specifications defining APIs for FIWARE.

Within the FIWARE context, the created MBT model is compliant for NGSI9/10 specification and can be reused for testing huge range of enablers. The only changes to be made concern the test environment and adaptation layer, in order to make it compatible with the component under test.

At last, one of our concern is to provide the tests to the community in the easiest way possible. Our solution is called "Generic Enabler i Testing Tool (GETT)". It provides to all (not only testers) the capacity to test their GE installation remotely from an online webpage. We call the approach "Plug and Test".

Figure 15. GETT Home page insight

In more details, the webpage we provide enables the user to not worry about the SoapUI installation and all the technical challenges that may vary from system incompatibilities to networking access problems. The user interface allows after pressing the "Launch Tests!" button (Figure 16) to point at the GE installation (for instance an URL) and it will launch the tests and give back an HTML test plan document with the result of each test for the user's installation.

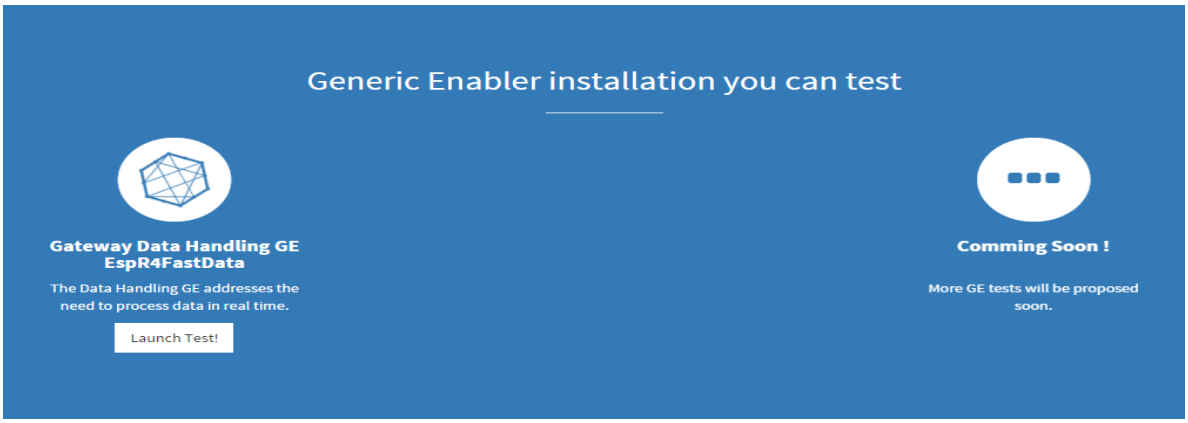

Figure 16. GETT Tests launch page

Our approach looks forward to make testing a cheap, easy and essential part of IoT expansion.

8 BIBLIOGRAPHY

[1] FIWARE, "FIWARE," [Online]. Available: https://www.fiware.org/.

[2] O. M. Alliance, "NGSI Context Management," May 2012. [Online]. Available: http://technical.openmobilealliance.org/Technical/release_program/docs/NGSI/V1_0-20120529-A/OMA-TS-NGSI_Context_Management-V1_0-20120529-A.pdf.

[3] B. Legeard and M. Utting, Practical Model-Based Testing: A Tools Approach, Morgan Kaufmann, 2007.

[4] M. Utting, B. Legeard et A. Pretschner, «A taxonomy of model-based testing approaches,» *Wiley InterScience,* 2010.

[5] [En ligne]. Available: http://www.soapui.org/.