

**Private Public Partnership Project (PPP)**

Large-scale Integrated Project (IP)



**D.11.8.2: Report on FIWARE QA testing**

**Project acronym:** FI-Core

**Project full title:** Future Internet - Core

**Contract No.:** 632893

**Strategic Objective:** FI.ICT-2011.1.7 Technology foundation: Future Internet Core Platform

**Project Document Number:** ICT-2013-FI-632893-WP11-D.11.8.2

**Project Document Date:** 2016-10-19

**Deliverable Type and Security:** PU

**Author:** Clara Pezuela(ATOS), Miguel A. Ramirez(ATOS), Carlos Lucena (ATOS); Riccardo Zanetti(ENG), Luigi Buglione(ENG), Annamaria Cappa(ENG), Andrea La Porta (ENG); Peter Muryshkin (Fraunhofer); Franck LeGall(EGM), Abbas Ahmad (EGM); Carmen MacWilliams (GAR)

**Contributors:** ATOS, ENG, Fraunhofer, EGM, GAR

Table of Contents

[1 Executive Summary 3](#_Toc465006780)

[2 Functional Testing 7](#_Toc465006781)

[**2.1 Motivation and approach** 7](#_Toc465006782)

[**2.2 Methodology** 7](#_Toc465006783)

[**2.3 Performed tests** 17](#_Toc465006784)

[***2.3.1 Documentation testing*** 22](#_Toc465006785)

[***2.3.2 APIs testing*** 37](#_Toc465006786)

[***2.3.3 Integration tests*** 45](#_Toc465006787)

[***2.3.4 Academy courses testing*** 52](#_Toc465006788)

[**2.4 NGSI APIs testing** 57](#_Toc465006789)

[***2.4.1 Introducing MBT in FIWARE (NGSI Model)*** 57](#_Toc465006790)

[***2.4.2 Test Adaptation and Execution*** 61](#_Toc465006791)

[***2.4.3 Test results discussion*** 63](#_Toc465006792)

[***2.4.4 Continuous Integration (CI) Testing*** 73](#_Toc465006793)

[**2.5 Automatic online documentation testing** 74](#_Toc465006794)

[**2.6 Summary of obtained results** 75](#_Toc465006795)

[3 Non-functional (stress) testing 77](#_Toc465006796)

[**3.1 Motivation and approach** 77](#_Toc465006797)

[**3.2 Methodology** 78](#_Toc465006798)

[**3.3 Performed tests** 83](#_Toc465006799)

[**3.3.1 Security – Identity Management - KeyRock** 84](#_Toc465006800)

[**3.3.2 Security - PEP Proxy - Wilma** 85](#_Toc465006801)

[**3.3.3 Security – Authorization PDP - AuthZForce** 86](#_Toc465006802)

[**3.3.4 IoT - IoTBroker - Aeron** 87](#_Toc465006803)

[**3.3.4 Data- Context Broker - Orion** 89](#_Toc465006804)

[**3.3.5 IoT - Data Edge Consolidation - Cepheus** 91](#_Toc465006805)

[**3.3.6 Cloud - Policy Manager - Bosun** 92](#_Toc465006806)

[**3.3.7 Data - Stream Oriented - Kurento** 94](#_Toc465006807)

[**3.3.6 IoT – Backend Device Management – IDAS** 94](#_Toc465006808)

[**3.3.6 Data - Complex Event Processing – Proton** 95](#_Toc465006809)

[**3.5 External assessment** 97](#_Toc465006810)

[4 Labelling model and application 99](#_Toc465006811)

[**4.1 Motivation and approach** 99](#_Toc465006812)

[**4.2 Labelling model representation** 102](#_Toc465006813)

[**4.3 Governance model** 104](#_Toc465006814)

[**4.4 Labelling functional aspects** 106](#_Toc465006815)

[**4.5 Labelling non-functional aspects** 112](#_Toc465006816)

[**4.5.1. Benchmarking for adjusting the non-functional results** 118](#_Toc465006817)

[**4.6 Labelling FIWARE GEs** 119](#_Toc465006818)

[5 Conclusions 121](#_Toc465006819)

[ANNEX 1: Non-functional (stress) testing report 122](#_Toc465006820)

[**A1.1 Non-functional testing reports (for version 5.1)** 122](#_Toc465006821)

[**A1.2 Non-functional testing reports (for version 5.2)** 122](#_Toc465006822)

[**A1.3 Non-functional testing reports (for version 5.3)** 122](#_Toc465006823)

[ANNEX 2: Functional testing report 123](#_Toc465006824)

[ANNEX 3: Non-functional labelling for some GEs 124](#_Toc465006825)

[**A3.1 IoTBroker non functional labelling** 124](#_Toc465006826)

[**A3.2 KeyRock IdM non functional labelling** 128](#_Toc465006827)

[**A3.3 Wilma PEP Proxy non functional labelling** 133](#_Toc465006828)

[**A3.4 AuthZForce PDP non functional labelling** 136](#_Toc465006829)

[**A3.5 Orion Context Broker non functional labelling** 139](#_Toc465006830)

[**A3.6 Proton non functional labelling (results from 5.1)** 140](#_Toc465006831)

[**A3.7 Kurento non functional labelling** 140](#_Toc465006832)

[**A3.8 IDAS non functional labelling (results from 5.3)** 140](#_Toc465006833)

[**A3.9 Cepheus non functional labelling** 141](#_Toc465006834)

[**A3.10 Bosun non functional labelling** 141](#_Toc465006835)

# [1](#_Toc460326232) Executive Summary

This deliverable is the second and final report of the work done and obtained results in the Quality Assurance task. The goal of this task has been analyzing and assessing the quality of the most used FIWARE GEs considering functional and non-functional aspects.

FIWARE is rapidly moving from experimental to production environments in which the platform must scale up in reliable and real workload conditions. This fact implies that all FIWARE GEris must work at an adequate quality, reliability and at performance level appropriate for these conditions. The reported task in this deliverable was launched in the framework of the initiative to analyze and assess the level of quality of each GE, providing diverse kind of **reports, labels for GEs and an assessment dashboard**.

The quality is evaluated from different points of view:

* **Curation of GEs documentation** (documentation testing), both inspecting the code and the accompanying documentation (installation manuals, user guidelines, academy courses and similar). The goal of this assessment is to support FIWARE users with high-quality support for installation, configuration and operation of FIWARE technology, thereby improving the FIWARE user experience in general.
* **Verification of the GE specification** (functional testing), developing the appropriate test cases to assess if the GEs implementation corresponds to what is defined in the specification.
* **Assessment of performance, stability and scalability** of GEs in operational environments, like under excessive workload (stress testing). Test scenarios are defined and executed such that limits of a GE under test are identified, and can be compared with reference levels. The goal of this assessment is to favor the applicability of FIWARE in purely commercial scenarios.

The **testing of the documentation** **and verification** has been done for all GE not deprecated in FIWARE Catalogue (**28 in total**). Three phases have been required to complete the QA functional test process. The first phase verifies for each GE the completeness of documentation, the consistency of artefacts and the soundness of information. The usability of documentation, by example, in case of installation manual is checked installing step by step the GE. In the second phase specific method calls verified the single APIs and the response correctness of each GEs. The last phase consisted of functional verifications based on reference architectures integrating some GEs. As result a live dashboard collects and maintains the assessment information and GE owners are punctually requested to correct the encountered deficiencies. At the end of the task, near 95% of the high priority GEs has passed successfully the documentation and verification tests. The medium and low priority GEs are around 80% of success but they are working on solving the issues.

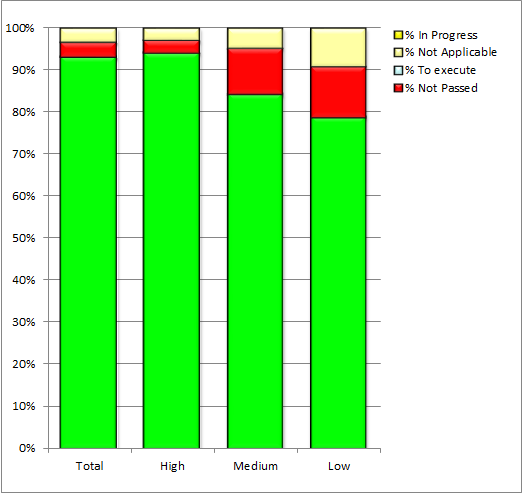


Figure 1: Overall functional testing results

On the other hand, the **stress testing** has been performed only for those GEs most critical in terms of performance in the overall architecture. An iterative process and operative methodology have been put in place, obtaining after each iteration, a complete report with the measures obtained after stress test and analysis of the data. The reports were sent to the GE owners for considering improvements about performance and stability for next release. Three iterations have been achieved until September this year: one took place in February testing 9 GEs (Orion, Proton, IoT Broker, IDAS, Kurento, Wilma, KeyRock, Cepheus, Bosun); the second one in May testing new versions of these GEs; and final one has tested again a new updated version of some of these GEs (Orion, Proton, Bosun, Wilma, KeyRock) plus two more identified (AuthZForce and Cygnus) and more frequent combination of GEs (IDAS+Orion+Cygnus and Wilma+KeyRock+AuthZForce). In summary, **10 GEs and 2 bundles** were tested in stress conditions.

Once the first iteration of stress testing was conducted, a quality assurance expert was consulted for carrying out an **independent assessment** of the followed process and executed tests to produce an assessment of the achieved work. The main conclusions of his assessment were:

* Important performance borders were identified
* Robustness of use within bounds was shown
* Documentation needs to be improved

According to this assessment, FIWARE GEs are fit for being released in a commercial operational environment with some adjustments. A new external independent assessment has been requested at the end of the task. The report was not available already at the time of this report.

As part of the overall testing process and based on the obtained results in the three aspects (documentation, verification and stress) above mentioned, an overall label of quality is granted to each GE. This global label represents the degree of quality of the GE by adopting the energy labelling system used by EU for devices. Specific labels for each analyzed category (usability, reliability, efficiency, scalability, performance and stability) are also granted. Thus, in the Catalogue each GE is labelled with a global label expanded by clicking of detailed labels map.

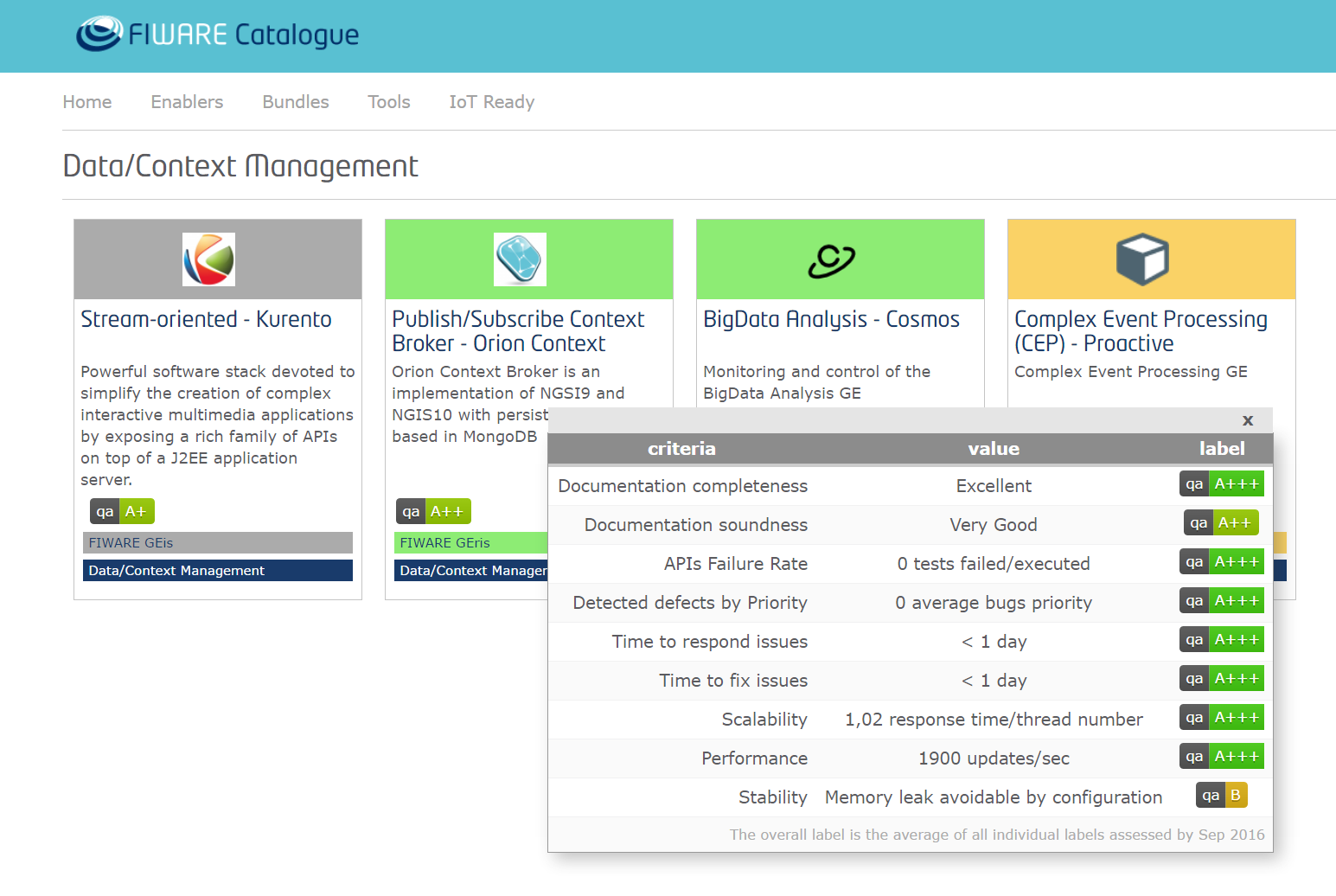


Figure 2: Visualization of quality labels in the FIWARE Catalogue

The work in the task, overall led by Atos, has been split into two working groups:

* **Functional test working group**, mostly focused on testing the documentation and specifications led by ENG. **ENG** has made all tests reported in the assessment dashboard (link) and the evaluation of the FIWARE Academy courses for all GEs. Additionally to this work, and also in this working group, two research lines have been conducted by EGM and Fraunhofer/Grassroot respectively. **EGM** has been doing research on NGSI APIs testing (see section 2.4) and **Fraunhofer/Grassroots** have worked on the automation of Catalogue testing documentation (see section 2.5). See section 2 for the overall details.
* **Non-functional working group,** focused on the stress testing, led by Atos and supported by ENG. This working group has been responsible for testing the scalability, stability and performance of the most used GEs and bundles (see section 3 for more details).

Commonly to the two working groups and in jointly manner, both of them have been working on the labeling model and establishment of GEs labels. The label model and concluded labels are detailed in section 4.

The stress testing working group has had weekly follow up calls and the overall task has set up as many calls and meetings as was needed to agree on the overall task strategy and agree on decisions.

In the annexes of this document, it can be found the links to all the reports and files that have been generated along the task execution.

The task has also promoted publicly the performed work by publishing a blog post in FIWARE blog, producing a brochure for events, giving talks in events and providing all the reports and used code in FIWARE forge and Github respectively.

Here the list of online resources those are available:

* Wiki page recording all the tests and results: <http://wiki.fiware.org/FIWARE_QA_Activities>
* Code and guidelines for executing the functional tests: <https://github.com/Fiware/test.Functional>
* Code and guidelines for executing the stress tests: <https://github.com/Fiware/test.NonFunctional>
* All the reports and other related generated assets at Docman in FIWARE forge under FIWARE Quality Assurance folder.
* Blog post at FIWARE blog: <https://www.fiware.org/2016/09/20/assessing-fiware-ges-quality/>
* Public document about Quality Assurance in FIWARE: <http://www.fiware.org/wp-content/uploads/2016/10/QA_public_document.pdf>

# 2 Functional Testing

The functional test activity verifies the consistency of the documentation and of the software package of each Generic Enabler and its usability both in standalone mode and combined as integrated platform.

## **2.1 Motivation and approach**

The main motivation to perform the functional test activity is to verify the consistency of GEri's release and their usability both in standalone mode and combined as integrated platform.

* The first scope is to check the completeness of documentation, specification, implementation and installation of the GEris.

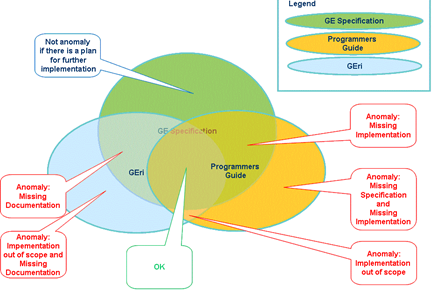


Figure 3: Artefacts completeness & consistency

* Another scope is to validate the APIs of the most used GE, the same ones already object of the not-functional test (performance test).
* The last scope is to perform the functional tests on the main functionalities of a bundle of GEs that identify the reference architecture selected as case study.

## **2.2 Methodology**

The description of the three planned test phases and the approach followed for each of them is detailed below.

The functional tests are structured into three phases:

1. Documentation Testing

2. APIs Testing

3. Bundle Integration Testing

After these three phases, there is the activity of **test and bug management** to make sure that each test activity performed is traced on the FIWARE Backlog and linked to Jira platform as a work-item ticket.

Each bug is reported on the Jira project corresponding to the Technical Chapter/Component under verification and linked to the work-item (test activity). When the test phase finishes, the work-item is closed.

The functional test activity includes also the evaluation of the **training efficiency** of the courses published and available on the FIWARE Academy (edu.fiware.org).

The training material created and published to the FIWARE Academy was evaluated on the basis of identified criteria corresponding to specific recommendations to be implemented for granting an efficient training offering. These recommendations (listed in the FIWARE wiki page “Working with the FIWARE Academy”) are both formal (minimum set of requirements to make the course comprehensible to users) and QA (set of requirements to address in order to comply with a good quality). The verification of formal requirements has been always an on-going activity within the task T3.2.2, in charge of the training content development and organization.

Thus, the evaluation activity included two different tasks performed by different teams that cooperated together and worked in parallel both in the definition of evaluation criteria and in the review phase:

* the evaluation of the QA requirements, resulting in the consequent QA label;
* the verification of formal requirements related to the structure of the course and to its efficient output.

***First Phase: Documentation Testing***

In the first phase it will be verified for each GEris, the completeness of documentation, the consistency of artifacts and the soundness of information, based on a specific user profile.

Moreover it will be checked the usability of installation manual by installing step by step the GEri, performing the sanity check operations and invoking the main APIs.

Here below the entities to test, included into this first phase, broken down by types and chapters.

|  |  |
| --- | --- |
| Applications/Services Ecosystem and Delivery Framework | |
| DataVisualization | SpagoBI |
| Application Mashup | Wirecloud |
| Marketplace | Marketplace |
| Repository | Repository-RI |
| Revenue Settlement and Sharing System | RSS-RI |
| Store | WStore |
| Security | |
| Identity Management | KeyRock |
| PEP Proxy | Wilma |
| Authorization PDP | AuthZForce |
| Data/Context Management | |
| Context Broker | Orion Context Broker |
| Big Data | Cosmos |
| Complex Event Processing (CEP) | Proactive |
| Stream Oriented | Kurento |
| Short Term Historic Open Data Repository | CKAN |
| BigData Analysis - Cosmos | Cygnus |
| Interface to Networks and Devices (I2ND) | |
| Network Information And Control | OFNIC |
| Advanced Middleware | Kiara |
| Internet of Things (IoT) Services Enablement | |
| Backend Device Management | IDAS |
| IoT Broker | Aeron |
| IoT Data Edge Consolidation | Cepheus |
| IoT Discovery | IoT Discovery |
| Protocol Adapter | MRCoAP |
| Advanced WebUI | |
| 3D-UI-XML3D | 3D-UI-XML3D |
| Augmented Reality | Augmented Reality |
| Cloud Rendering | Cloud Rendering |
| GIS Data Provider | Geoserver/3D |
| Interface Designer | Interface Designer |
| POI Data Provider | POI Data Provider |
| Real Virtual Interaction | Real Virtual Interaction |
| Synchronization | Synchronization |
| Virtual Characters | Virtual Characters |
| Cloud Hosting | |
| Policy Manager | Bosun |

Table 1: First Phase GE

**Documentation Testing Validation criteria**

The designed tests aim to checks the completeness, consistency, soundness and usability of GEs.

Some checks are subjective because they are based on the evaluator profile. Therefore two evaluation levels (decision makers and developers) have been identified.

Checking completeness means to verify that each released artefact is complete in all its parts.

The consistency check intends to verify that the release contains all the expected artefacts and that they are consistent between them.

The soundness’ verification ensures that each artefact has proper contents to its purpose and suitable to the profile of those who uses them. In fact, the content of a document might be enough to a manager who must decide whether to adopt a solution based on FIWARE but not enough to the developer that must implement and vice versa.

Finally, the usability check intends to verify that a document or a package is easily usable, for example, that an installation manual allows to properly installing a released package.

Some verifications of "Completeness" are made on the web catalogue and they aim to ensure that information are complete, updated and the linked contents are really accessible. A detailed checklist, containing all verifications to execute, is linked to these tests. This information is most useful to high-level user profiles that need an overview to determine if an application is useful to their goals.

Another verification of completeness, useful to both profiles, is to check the Programmer's Guide and Open Specification in order to verify the Programmer's Guide covers fully the Open Specification.

Regarding Usability:

* for a decision maker it is required:

- all basic information is available and easily attainable

- the catalogue is easily navigable

- training / online courses are available

* for a developer it is required:

- get simple and fast installation methods such as docker, script

- make a step by step GEri installation using the installation manual

- execute the Sanity Check Procedures

- invoke easily the exposed APIs.

***Second Phase: APIs Testing***

In the second phase, it will be verified all APIs of the GEs, through the specific method call and verifying the response correctness.

Each Application Programming Interface has been verified as atomic procedure, without combining business logic. Each procedure has been tested only in the positive case in order to verify the correct execution of the functionality. The purpose is to verify that the software package actually contains all the APIs declared in the programmer guides without paying special attention to the negative case or wrong input of each specific interface.

During this phase, priority will be given to the following GEs, that are the most used ones and also under performance test.

|  |  |
| --- | --- |
| Security | |
| Identity Management | KeyRock |
| PEP Proxy | Wilma |
| Authorization PDP | AuthZForce |
| Data/Context Management | |
| Context Broker | Orion Context Broker |
| Complex Event Processing (CEP) | Proactive |
| Stream Oriented | Kurento |
| Short Term Historic Open Data Repository(CKAN) | CKAN |
| BigData Analysis - Cosmos | Cygnus |
| Internet of Things (IoT) Services Enablement | |
| Backend Device Management | IDAS |
| IoT Broker | IoT Broker |
| IoT Data Edge Consolidation | Cepheus |
| IoT Discovery | IoT Discovery |
| Interface to Networks and Devices (I2ND) | |
| Advanced Middleware (Kiara) | KIARA |
| Advanced WebUI | |
| Augmented Reality | Augmented Reality |
| GIS Data Provider | Geoserver/3D |
| POI Data Provider | POI Data Provider |
| Synchronization | Synchronization |
| XML 3D | 3D-UI-XML3D |
| Interface Designer | Interface Designer |
| Virtual Characters | Virtual Characters |
| Cloud Rendering | Cloud Rendering |
| Applications/Services Ecosystem and Delivery Framework | |
| Data Visualization | SpagoBI |
| Marketplace | WMarket |
| Repository | RepositoryRI |
| Application Mashup | Wirecloud |
| Revenue Settlement and Sharing System | RSS-RI |
| Store | WStore |
| Cloud Hosting | |
| Policy Manager | Bosun |

Table 2: Second Phase GE

***Third Phase: Bundle Integration Testing***

The functional bundle integration testing activity performs verifications based on functional scenarios combining some of the main GE’s interfaces to highlight the interaction among the GEs composing the bundle. The functional bundle integration tests simulate a real use case.

The proposed functional scenario is a simple Parking management, involving sensors to detect when a new car enter into the parking, another sensor when a car exits out of the parking and the last sensor to detect the CO2 level in the parking. The users can manage and verify the parking information according to specific policy of access control.

The GEs that compose the integrated FIWARE bundle where the functional integration tests are performed are:

|  |  |
| --- | --- |
| Chapter | GE |
| Security | Identity Management - KeyRock |
| PEP Proxy – Wilma |
| Authorization PDP – AuthZForce |
| Internet of Things Services Enablement | Backend Device Management – IDAS  IoT Agent Ultraligth2.0/HTTP |
| Data/Context Management | Publish/Subscribe Context Broker - Orion |
| Complex Event Processing (CEP) - Proactive Technology Online (Proton) |
| Applications/Services and Data Delivery | Data Visualization - SpagoBI |

The functional scenarios use some of the main GE interfaces to highlight the interaction among the GEs of the platform. The integration tests have been performed using a Client Application in order to simulate a real application. Here follows the list of the scenarios grouped by functional area contents.

|  |  |  |
| --- | --- | --- |
| **Test Suite** | **ID**  **TestSuite\_TestCase** | **Test case** |
| Security setup | ‎TS01\_TC01 | Register User |
| ‎TS01\_TC02 | Register Application |
| ‎TS01\_TC03 | Register new PEP-PROXY |
| Access Control | ‎TS02\_TC01 | Login as Guest |
| ‎TS02\_TC02 | Login as Operator |
| ‎TS02\_TC03 | Login as Administrator |
| Entity Management | ‎TS03\_TC01 | Create Entity Parking |
| ‎TS03\_TC02 | Remove Entity Parking |
| ‎TS03\_TC03 | Modify Entity Parking - Access Permit |
| ‎TS03\_TC04 | Modify Entity Parking - Access Denied |
| ‎TS03\_TC05 | Get Entity Parking |
| Service Registration | ‎TS04\_TC01 | Create service Parking |
| Device Registration | ‎TS05\_TC01 | Create device: Sensor sCarEntry |
| ‎TS05\_TC02 | Create device: Sensor sCarExit |
| ‎TS05\_TC03 | Create device: Sensor sCO2 |
| Observation Measurement | ‎TS06\_TC01 | Sensor sCarEntry detects a car entry (parking not full) |
| ‎TS06\_TC02 | Sensor sCarEntry detects a car entry (parking full) |
| ‎TS06\_TC03 | Sensor sCarExit detects a car exit |
| ‎TS06\_TC04 | Sensor sCO2 measures the CO2 level |
| Data visualization | ‎TS07\_TC01 | Dynamic Report - Parking data |
| ‎TS07\_TC02 | Static Report – Parking data |
| ‎TS07\_TC03 | Static Report – Parking Statistics |

Table 3: Functional scenarios of integration tests

The integration tests have been performed imagining to simulate a real application with a simple business logic. The test suite is a group of test cases or scenarios related to the same functional content.

Each Integration Test Scenario is a test case containing a list of steps explaining the flow of the communication through the GEs and for each test case is reported the log of the main process steps.

The detailed results are reported in the ANNEX 2 of the current document

**Test and bug management**

Each test activity is traced on FIWARE Backlog and linked on Jira platform as work-item.

Each bug is reported on the Jira project corresponding to Technical Chapter/Component under verification and linked to the work-item. When the test activity finishes, the work-item is closed.

The attached image describes spreadsheets collecting the designed test suite.

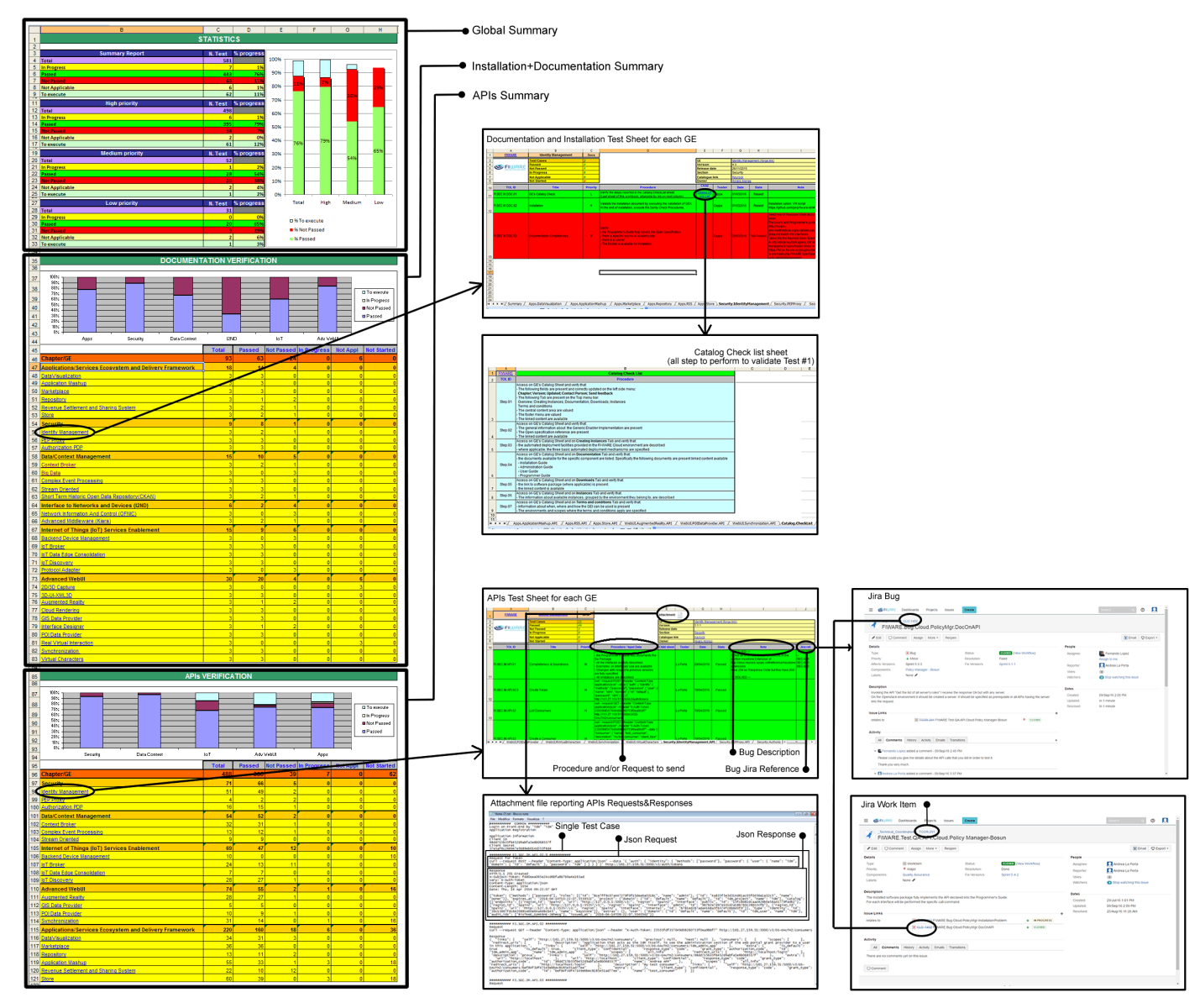


Figure 4: Spreadsheets collection with functional tests results

It has a summary sheet that reports the global overview on activity progress and detailed sheets for each GE.

These sheets show all planned tests, permit to trace the execution information and the Jira reference in case of failed result.

Each sheet also reports the general information (owner, version, etc.) about GE or Platform to test and the global pre-requirements in order to perform the tests.

For some tests, in addition to the execution procedure and the validation criteria, a “child” sheet that represents a checklist is linked.

It reports, based on the test scenario, the items list to verify, the steps to execute or the APIs to invoke.

Here below the spreadsheet information for GEri.

|  |  |
| --- | --- |
| Overview | |
| Field | Description |
| GE | Generic Enabler and Reference implementation names |
| Section | Technical chapter which the GEri belongs to |
| Catalogue link | Link to GEri FIWARE web catalogue |
| Owner | The reference person for the GEri |
| Total Test Cases | Total Number of Test Cases to execute |
| Passed | Test Cases passed |
| Not Passed | Test Cases Not Passed |
| In Progress | Test Cases still in Progress |
| Not Applicable | Test Case Not Applicable |
| Not Started | Test Case Not Started |
| Test Cases Information | |
| Field | Description |
| TOL ID | Unique identification of a Test Case |
| Title | Type of test or scenario to perform |
| Version | Version of the last GEri release |
| Priority | Execution Priority level (High, Medium, Low) |
| Procedure | Steps to execute and validation criteria |
| Child sheet | Link to check list sheet (item list/step to perform/APIs) |
| Tester | Executor name |
| Date | Execution date |
| State | Test Result (Passed/Not Passed/Not Applicable/In Progress) |
| Note | Execution annotation |
| Jira reference | Reference to JIIRA Platform in case of failure |

Table 4: Test Object List Information

## **2.3 Performed tests**

This section describes the functional test activity performed on GEs.

As said before, the tests are split into three phases:

1. Verification of completeness, consistency and soundness of the documentation. Moreover it is checked the usability of installation manual by installing the software step by step.
2. Verification of the GE’s APIs, through the specific method calls and checks of the responses correctness according to the reference documentation.
3. Functional verification of a reference architecture that integrates and combines some of the tested GEs. The functional integration tests highlight the communication among the components using some of the main interfaces (API) that composes the functional scenarios.

Documentation testing and APIs testing activities have been performed for all the generic enablers listed in the table below. The whole results are reported in the ANNEX 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **GE** | **GEri** | **Type of App** | **Type of Test** | **HW and SW** |
| **Data Visualization** | SpagoBI | RESTful Web services  JavaScript API  Web GUI | Documentation  Installation  APIs | CPU: 2 VCPU  RAM: 4GB  Hard Disk: 40GB  Operating System: Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86\_64) |
| **Application Mashup** | Wirecloud | RESTful Web services  JavaScript API  Web GUI | Documentation  Installation  APIs | CPU: 2 VCPU  RAM: 4GB  Hard Disk: 40GB  Operating System: Ubuntu  14.04.1 LTS (GNU/Linux  3.13.0-37-generic x86\_64) |
| **Marketplace** | Wmarket | RESTful Web services | Documentation  Installation  APIs | CPU: 2 VCPU  RAM: 4GB  Hard Disk: 40GB  Operating System: Ubuntu  14.04.1 LTS (GNU/Linux  3.13.0-37-generic x86\_64) |
| **Repository** | Repository-RI | RESTful Web services | Documentation  Installation  APIs | CPU: 2 VCPU  RAM: 4GB  Hard Disk: 40GB  Operating System: Ubuntu  14.04.1 LTS (GNU/Linux  3.13.0-37-generic x86\_64) |
| **Revenue Settlement and Sharing System** | RSS-RI | RESTful Web services | Documentation  Installation  APIs | CPU: 1 VCPU  RAM: 2GB  Hard Disk: 20GB  Operating System: Ubuntu  14.04.1 LTS (GNU/Linux  3.13.0-37-genericx86\_64) |
| **Store** | Wstore | RESTful Web services | Documentation  Installation  APIs | CPU: 1 VCPU  RAM: 2GB  Hard Disk: 20GB  Operating System: Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86\_64)2GB |
| **Identity Management** | KeyRock | RESTful Web services  Web GUI | Documentation  Installation  APIs | RAM: 2GB  Disk: 20GB  VCPUs: 1 VCPU  Operating System: Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86\_64)  GE version: 5.1.1 |
| **PEP Proxy** | Wilma | RESTful Web services | Documentation  Installation  APIs | RAM: 4GB  Disk: 40GB  VCPUs: 2 VCPU  Operating System: Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86\_64)  GE version: 4.3 |
| **Authorization PDP** | AuthZForce | RESTful Web services | Documentation  Installation  APIs | RAM: 4GB  Disk: 40GB  VCPUs: 2 VCPU  Operating System: Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86\_64)  GE version: v4.3.0 |
| **Context Broker** | Orion Context Broker | RESTful Web services | Documentation  Installation  APIs | RAM: 2GB  Disk: 20GB  VCPUs: 1 VCPU  Operating System:  CentOS 7.2 -  Linux 3.10.0-327.13.1.el7.x86\_64  GE version: 023.0 |
| **Big Data** | Cosmos | RESTful Web services | Documentation  Installation  APIs | RAM: 8GB  Disk: 25GB  VCPUs: 2 VCPU  Operating System: Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-37-generic x86\_64)  GE version: missing |
| **Complex Event Processing (CEP)** | Proactive | RESTful Web services | Documentation  Installation  APIs | RAM: 2GB  Disk: 20GB  VCPUs: 1 VCPU  Operating System: Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-37-generic x86\_64)  GE version: 4.4.1 |
| **Stream Oriented** | Kurento | RESTful Web services | Documentation  Installation  APIs | RAM: 4GB  Disk: 40GB  VCPUs: 2 VCPU  Operating System: Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-37-generic x86\_64)  GE version: 6.0 |
| **Short Term Historic Open Data Repository** | CKAN | RESTful Web services | Documentation  Installation  APIs | RAM: 4GB  Disk: 40GB  VCPUs: 2 VCPU  Operating System: Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-37-generic x86\_64)  GE version: 2.5 |
| **Network Information And Control** | OFNIC | RESTful Web services | Documentation  Installation | RAM: 4GB  Disk: 40GB  VCPUs: 2 VCPU  Operating System: Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86\_64)  GE version: v4.3.0 |
| **Advanced Middleware** | Kiara | RPC APIs | Documentation  Installation | RAM: 2GB  Disk: 20GB  VCPUs: 1 VCPU  Operating System: Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86\_64)  GE version: v4.3.0 |
| **Backend Device Management** | IDAS | RESTful Web services | Documentation  Installation  APIs | RAM: 2GB  Disk: 20GB  VCPUs: 1 VCPU  Operating System: Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-37-generic x86\_64)  GE version: Release 5 |
| **IoT Broker** | IoTBroker | RESTful Web services | Documentation  Installation  APIs | RAM: 8GB  Disk: 25GB  VCPUs: 2 VCPU  Operating System: Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-37-generic x86\_64)  GE version: 3.2.3. |
| **IoT Data Edge Consolidation** | Cepheus | RESTful Web services | Documentation  Installation  APIs | RAM: 8GB  Disk: 25GB  VCPUs: 2 VCPU  Operating System: Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-37-generic x86\_64)  GE version: 0.1.7-SNAPSHOT |
| **IoT Discovery** | IoT Discovery | RESTful Web services  Web User Interface | Documentation  Installation  APIs | RAM: 8GB  Disk: 25GB  VCPUs: 2 VCPU  Operating System: Ubuntu 14.04.4 LTS (GNU/Linux 3.13.0-37-generic x86\_64)  GE version: 4.2  (Sense2Web Platform:5.2.3.SNAPSHOT - NGSI-9:R4.2) |
| **3D-UI-XML3D** | 3D-UI-XML3D | JavaScript API | Documentation  Installation  APIs | CPU: 1 VCPU  RAM: 2GB  Hard Disk: 20GB  Operating System: Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86\_64) |
| **Augmented Reality** | Augmented Reality | JavaScript API | Documentation  Installation  APIs | CPU: 1 VCPU  RAM: 2MB  Hard Disk: 20GB  Operating System: Ubuntu 12.04.5 LTS (GNU/Linux 3.2.0-67-virtual x86\_64) |
| **Cloud Rendering** | Cloud Rendering | JavaScript API | Documentation  Installation | CPU: 2 VCPU  RAM: 4GB  Hard Disk: 40GB  Operating System: Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86\_64) |
| **GIS Data Provider** | Geoserver/3D | RESTful Web services | Documentation  Installation  APIs | CPU: 1 VCPU  RAM: 2GB  Hard Disk: 20GB  Operating System: Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86\_64) |
| **Interface Designer** | Interface Designer | JavaScript API | Documentation  Installation | CPU: 1 VCPU  RAM: 2GB  Hard Disk: 20GB  Operating System: Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86\_64) |
| **POI Data Provider** | POI Data Provider | RESTful Web services | Documentation  Installation  APIs | CPU: 2 VCPU  RAM: 4GB  Hard Disk: 40GB  Operating System: Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86\_64) |
| **Synchronization** | Synchronization | RESTful Web services  JavaScript API | Documentation  Installation  APIs | CPU: 2 VCPU  RAM: 4GB  Hard Disk: 40GB  Operating System: Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86\_64) |
| **Virtual Characters** | Virtual Characters | JavaScript API | Documentation  Installation  APIs | CPU: 1 VCPU  RAM: 2GB  Hard Disk: 20GB  Operating System: Ubuntu 14.04.1 LTS (GNU/Linux 3.13.0-37-generic x86\_64) |
| **Cloud Hosting - Policy Manager - Bosum** | Bosum | RESTful Web services | Documentation  Installation  APIs | CPU: 1 VCPU  RAM: 2GB  Hard Disk: 20GB  Operating System: Ubuntu 14.04.2 LTS (Linux 3.13.0-92-generic x86\_64) |
| **BigData Analysis – Cosmos – Cygnus** | Cygnus | RESTful Web services | Documentation  Installation  APIs | CPU: 4 VCPU  RAM: 8GB  Hard Disk: 80GB  Operating System: Ubuntu 14.04.2 LTS (Linux 3.13.0-92-generic x86\_64) |

Table 5: List of functional tested GEs

The following section, Documentation testing and APIs testing, report the functional test results of all the tested components, not only those for which have been performed the non-functional tests and the labeling model has been applied. The list of those crossed GEs is the following:

1. Identity Management – KeyRock
2. Authorization PDP – AuthZForce
3. PEP Proxy – Wilma
4. Context Broker - Orion
5. Complex Event Processing (CEP) – Proton
6. Stream Oriented – Kurento
7. Backend Device Management – IDAS (IoT Agents)
8. IoT Broker – Aeron
9. IoT Data Edge Consolidation – Cepheus
10. Policy Manager - Bosun

### ***2.3.1 Documentation testing***

The Documentation testing activity consists in two types of verifications:

**Documentation - Completeness & Soundness Test**

o The Programmer’s Guide fully covers the Open Specification.

o There is specific course in academy site.

o There is a tutorial.

o The Docker is available for installation.

**Documentation - Installation Test**

o The GE manual allows installing easily the component.

o The hardware and software pre-requirements are described.

The current section reports the main conclusions for those GEs analyzed by non-functional tests and by label model.

**1.** **Security - Identity Management - KeyRock**

The Identity Manager is the central component that provides a bridge between IdM systems at connectivity-level and application-level. Furthermore, Identity Management is used for authorizing foreign services to access personal data stored in a secure environment.

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/identity-management-keyrock>

**Documentation - Completeness & Soundness Test**

* There was some mismatch (solved) between Programmer’s Guide and Open Specification
* A course in academy site is available.
* The Docker is available for installation.
* The Programmer’s Guide not always describes completely and with examples the interfaces. Some examples were wrong and some request parameters were missing (solved)

**Documentation - Installation Test**

o The GE manual allows installing easily the component.

o The hardware and software pre-requirements are described.

**2.** **Security - Authorization PDP – AuthZForce**

Wilma is the reference implementation of PEP Proxy Generic Enabler. It is completely integrated with the FIWARE ecosystem and specifically with FIWARE account. It is thought to work with OAuth2 and XACML protocols, the standards for authentication and authorization chosen in FIWARE.

Thanks to this component and together with Identity Management and Authorization PDP GEs, is possible to add authentication and authorization security to backend applications.

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/pep-proxy-wilma>

**Documentation - Completeness & Soundness Test**

o The Programmer’s Guide covers the Open Specification

o A course in academy site is available.

o The Docker is available for installation.

o The Programmer’s Guide describes with examples the interfaces.

**Documentation - Installation Test**

o The GE manual allows to install the component.

o Not all hardware and software pre-requirements were described (solved).

**3.** **Security - PEP Proxy – Wilma**

Wilma is the reference implementation of PEP Proxy Generic Enabler. It is completely integrated with the FIWARE ecosystem and specifically with FIWARE account. It is thought to work with OAuth2 and XACML protocols, the standards for authentication and authorization chosen in FIWARE.

Thanks to this component and together with Identity Management and Authorization PDP GEs, is possible to add authentication and authorization security to backend applications.

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/pep-proxy-wilma>

**Documentation - Completeness & Soundness Test**

o The Programmer’s Guide covers the Open Specification

o A course in academy site is available.

o The Docker is available for installation.

o The Programmer’s Guide describes with examples the interfaces.

**Documentation - Installation Test**

o The GE manual allows installing the component.

o Not all hardware and software pre-requirements were described (solved)

**4.** **Data/Context Management - Context Broker – Orion**

The Orion Context Broker is an implementation of the Publish/Subscribe Context Broker GE, providing the NGSI9 and NGSI10 interfaces to perform the following operations:

o Register context producer applications, e.g. a temperature sensor within a room

o Update context information, e.g. send updates of temperature

o Being notified when changes on context information take place (e.g. the temperature has changed) or with a given frequency (e.g. get the temperature each minute)

o Query context information. The Orion Context Broker stores context information updated from applications, so queries are resolved based on that information.

Orion mediates between consumer producers (e.g. sensors) and the context consumer applications (e.g. a smartphone applications taking advantage of the context information provided by the sensors).

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/publishsubscribe-context-broker-orion-context-broker>

**Documentation - Completeness & Soundness Test**

o There are some mismatch between Programmer’s Guide and Open Specification

o Courses in the academy site are available.

o The Docker is available for installation.

o The hardware and software pre-requirements are described.

o The Programmer’s Guide describes completely and with examples the interfaces.

**Documentation - Installation manual Test**

o The GE installation manuals allow to install easily the component, technical knowledge are necessary to install.

o The Docker is available for installation.

**5.** **Data/Context Management - BigData Analysis – Cosmos – Cygnus**

Cosmos and its ecosystem is the BigData Analysis Generic Enabler reference implementation (GEri). Cosmos is mainly addressed to those service providers aiming to expose a BigData Analysis GE-like services. For those service providers, the data analysis is not a goal itself but providing ways others can perform such data analysis.

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/bigdata-analysis-cosmos>

Cygnus has been the GE tested, Cygnus is part of the Cosmos.

**Documentation - Completeness & Soundness Test**

o There were some mismatch between Programmer’s Guide and Open Specification

o Courses in the academy site are available.

o The Docker is not available for installation because Cosmos is not intended to have docker images since it is not a single-machine software, but cluster based.

o The hardware and software pre-requirements are described.

**Documentation - Installation manual Test**

o The GE installations manuals do not allow to install easily the component, because it’s a cosmos of components, technical knowledge is necessary to install.

**6.** **Data/Context Management - Complex Event Processing (CEP) – Proton**

The CEP GE analyses event data in real-time, generates immediate insight and enables instant response to changing conditions. While standard reactive applications are based on reactions to single events, the CEP GE reacts to situations rather than to single events. A situation is a condition that is based on a series of events that have occurred within a dynamic time window called processing context. Situations include composite events (e.g., sequence), counting operators on events (e.g., aggregation) and absence operators. The Proactive Technology Online is an implementation of the FIWARE CEP (Complex Event Processing) GE.

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/complex-event-processing-cep-proactive-technology-online>

**Documentation - Completeness & Soundness Test**

o There were some mismatch (solved) between Programmer’s Guide and Open Specification that have been solved.

o Courses in the academy site are available.

o The hardware and software pre-requirements are described.

o The Programmer’s Guide describes completely and with examples the interfaces.

**Documentation - Installation manual Test**

o The GE installations manuals allow to install easily the component, technical knowledge is necessary to install.

**7.** **Data/Context Management - Stream Oriented – Kurento**

The Stream Oriented GE is a powerful software stack devoted to simplify the creation of complex interactive multimedia applications by exposing a rich family of APIs on top of a J2EE application server. Stream Oriented is a development framework that provides an abstraction layer for multimedia capabilities, allowing non-expert developers to include interactive media components to their applications. At the heart of this enabler there is the Open API. A REST-like API based on JSON RPC 2.0, exposing a toolbox of Media Elements that can be chained to create complex media processing pipelines. The Stream Oriented GE provides several client implementations of the Open API. The Java client allows developers to include media capabilities to Java or JEE applications.

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/stream-oriented-kurento>

**Documentation - Completeness & Soundness Test**

o There were some mismatch (solved) between Programmer’s Guide and Open Specification

o Courses in the academy site are available.

o The hardware and software pre-requirements are described.

o The Programmer’s Guide describes completely and with examples the interfaces.

**Documentation - Installation manual Test**

o The GE installation manuals allow to install easily the component, technical knowledge is necessary to install.

**8.** **Internet of Things Services Enablement - Backend Device Management – IDAS (IoT Agents)**

The IDAS component is an implementation of the [Backend Device Management GE](https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.ArchitectureDescription.IoT.Backend.DeviceManagement).

In May 2016, IDAS has been totally restructured the new IDAS (Release 5).

IDAS is composed by the following components

o IoT Agent for UL2.0 (HTTP or MQTT transport)

o IoT Agent for JSON (MQTT transport)

o IoT Agent for OMA-LWM2M (CoAP transport)

o IoT Agent for SIGFOX-Cloud Devices

The Northbound API of IDAS is the OMA NGSI protocol, as long as IoT Agents are normally expected to be connected to an NGSI Broker, such as the [Orion ContextBroker](http://catalogue.fiware.org/enablers/configuration-manager-orion-context-broker).

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/backend-device-management-idas>

**Documentation - Completeness & Soundness Test**

o There were some mismatch (solved) between Programmer’s Guide and Open Specification

o Courses in the academy site are available.

o The Docker is not available for installation.

o The hardware and software pre-requirements are described.

o The Programmer’s Guide describes completely and with examples the interfaces.

**Documentation - Installation manual Test**

o The GE installations manuals allow to install easily the component, technical knowledge is necessary to install.

o The Docker is not available for installation, but it’s easy installing using Git.

**9.** **Internet of Things Services Enablement - IoT Broker – Aeron**

The IoTBroker by NEC component is the reference implementation of the Generic Enabler  [IoTBroker](https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.ArchitectureDescription.IoT.Backend.DeviceManagement). The IoT Broker GE is a component for retrieving and aggregating information from the Internet of Things.

The main features of this Enabler are:

o Offering a single point of contact to the user, hiding the complexity of the multi-provider nature of the Internet of Things.

o Collecting and aggregating information about thousands of real-world objects on behalf of the user.

o Providing means to assemble lower-level device information (device-centric access) into higher-level Thing information (information-centric access).

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/iot-broker>

**Documentation - Completeness & Soundness Test**

o There were some mismatch (solved) between Programmer’s Guide and Open Specification

o Courses in the academy site are available.

o The Docker is available for installation.

o The hardware and software pre-requirements are described.

o The Programmer’s Guide describes completely and with examples the interfaces.

**Documentation - Installation manual Test**

o The GE installation manuals allow to install easily the component, technical knowledge is necessary to install.

o The Docker is available for installation, but it’s easy installing using Git.

o The configuration settings in order to integrate the component with the others are not always described in a clear manner.

**10.** **Internet of Things Services Enablement - IoT Data Edge Consolidation – Cepheus**

IoT Data Edge Consolidation GE is designed to provide a common access in real time to all data, for any kind of sensors and "Things". Using a lightweight storage system, this enabler can save and locally store relevant processed data, as close as possible to the processed entities.

Cepheus is divided in two simple deployable applications:

o NGSI broker: it is a light NGSI broker which features an implementation of [the standardized NGSI API](http://technical.openmobilealliance.org/Technical/release_program/docs/NGSI/V1_0-20120529-A/OMA-TS-NGSI_Context_Management-V1_0-20120529-A.pdf).

o CEP: it is dedicated to REST management API to configure real time analysis on NGSI events.

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/iot-data-edge-consolidation-ge-cepheus>

**Documentation - Completeness & Soundness Test**

o There are no mismatch between Programmer’s Guide and Open Specification

o Courses in the academy site are available.

o The hardware and software pre-requirements are described.

o The Programmer’s Guide describes completely and with examples the interfaces.

**Documentation - Installation manual Test**

o The GE installation manuals allow to install easily the component, technical knowledge is necessary to install.

**11.** **Cloud Hosting - Policy Manager – Bosun**

Policy Manager is an easy rule engine designed to be used in the OpenStack ecosystem and of course inside the FIWARE Cloud.

The Policy Manager GE provides the basic management of cloud resources based on rules, as well as management of the corresponding resources within the FIWARE Cloud Instance like actions based on physical monitoring or infrastructure, security monitoring of resources and services or whatever that could be defined by a facts, actions and rules.

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/policy-manager-bosun>

**Documentation - Completeness & Soundness Test**

o Basically there are no mismatches, but the Programmer's Guide doesn't contain examples for all APIs.

o Courses are not available on the academy but it is available a tutorial on Youtube

o The HW (minimum requirements) is not specified. The SW required and all dependencies are described. The Operating System is supposed is a Linux (Ubuntu or Centos) but not specified.

o Not all interfaces are described into the Programmer's Guide. They are described only on the OPEN API.

**Documentation - Installation manual Test**

o The installation manual allows installing the component but some steps are not very easy. A technical knowledge is necessary to proceed with the installation.

1. **Internet of Things Services Enablement – IoT Discovery**

IoTDiscovery is the reference implementation for the IoT Discovery GE open specification.

The primary purpose is to allow context producers to register their IoT Objects in linked-data format, and in turn allow context consumers to discover them using a set of search techniques.

The implementation provides two modules:

* NGSI-9 Server: it provides a repository for the storage of NGSI entities and allows NGSI-9 clients to:
  + Register context information about Sensors and Things.
  + Discover context information using ID, attribute, attribute domain, and entity type
* Sense2Web Platform: a Web User interface is provided for users in order to register and discover the IoT elements using:
  + Retrieve descriptions in RDF
  + Semantic querying via SPARQL

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/iot-discovery>

**Documentation - Completeness & Soundness Test**

* There were some mismatch (solved) between Programmer’s Guide and Open Specification
* Courses in the academy site are available.
* The Docker is available for installation.
* The hardware and software pre-requirements are described.
* The Programmer’s Guide describes completely and with examples the interfaces.

**Documentation - Installation manual Test**

o The GE installation manuals allow to install easily the component, technical knowledge is necessary to install.

o The configuration settings in order to integrate the component with the others are not always described in a clear manner.

1. **DataContext - CKAN**

CKAN is a powerful data management system that makes data accessible, it allows to streamline publishing, sharing, finding and using data. CKAN is aimed at data publishers (national and regional governments, companies and organizations) wanting to make their data open and available.

Features include:

* Complete catalog system with easy to use web interface and a powerful API.
* Strong integration with third-party CMS’s like Drupal and WordPress.
* Data visualization and analytics.
* Workflow support lets departments or groups manage their own data publishing.
* Fine-grained access control.
* Integrated data storage and full data API.
* Federated structure: easily set up new instances with common search.

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/ckan>

**Documentation - Completeness & Soundness Test**

* There were some mismatch between Programmer’s Guide and Open Specification.
* Courses in the academy site are available.
* The hardware and software pre-requirements are described.
* The Programmer’s Guide describes completely and with examples the interfaces.

**Documentation - Installation manual Test**

* The GE installation manuals allow to install the component, technical knowledge is necessary to install.

1. **Data Visualization - SpagoBI**

SpagoBI is an open source suite for Business Intelligence and Big Data Analytics. It provides full capabilities to get insights on data and turn them into actionable knowledge for effective decision-making processes. It includes reporting and charting tools, innovative solutions for emerging domains, such as location intelligence, KPIs, real-time, mobile, big data, data mining, what-if and social listening.

The GE is present in the FIWARE catalogue at the following link:

<http://catalogue.fiware.org/enablers/data-visualization-spagobi>

**Documentation - Completeness & Soundness Test**

* The Programmer’s Guide covers the Open Specification
* Many specific courses with tutorials and manuals are available. A course in FIWARE academy site is also present
* The Docker is available for installation.
* The Programmer’s Guide describes with examples the APIs. It lacks some information needed to invoke interfaces

**Documentation - Installation Test**

* The hardware and software pre-requirements are fully described.
* The GE manual allows to install the component easily

1. **Application Mashup - Wire****cloud**

Wirecloud builds on cutting-edge end-user development, RIA and semantic technologies to offer a next-generation end-user centred web application mashup platform aimed at leveraging the long tail of the Internet of Services.

The GE is present in the FIWARE catalogue at the following link:

<http://catalogue.fiware.org/enablers/application-mashup-wirecloud>

**Documentation - Completeness & Soundness Test**

* The Programmer’s Guide covers the Open Specification.
* A course in FIWARE academy site and tutorials are available.
* The Docker is available for installation.
* The Programmer’s Guide doesn’t fully document some APIs.

**Documentation - Installation Test**

* The hardware and software pre-requirements are fully described.
* Some problems were encountered installing the GE by following the installation manual.

1. **Marketplace – Wmarket**

WMarket provides functionality necessary for bringing together offering and demand for making business. These functions include basic services for registering business entities, publishing and retrieving offerings and demands, search and discover offerings according to specific consumer requirements as well as lateral functions like review, rating and recommendation.

The GE is present in the FIWARE catalogue at the following link:

<http://catalogue.fiware.org/enablers/marketplace-wmarket>

**Documentation - Completeness & Soundness Test**

* The Programmer’s Guide covers the Open Specification
* A course in FIWARE academy site is available.
* The Docker is available for installation.
* The Programmer’s Guide fully describes the APIs.

**Documentation - Installation Test**

* The hardware and software pre-requirements are fully described.
* The GE manual allows to install the component easily

1. **Repository - Repository RI**

The repository provides a consistent uniform API to USDL service descriptions and associated media files for applications of the business framework. A service provider can use the Repository to publish the description of various aspects of the service according to a uniform description language. The repository must be able to store resources in arbitrary formats.

The GE is present in the FIWARE catalogue at the following link:

<http://catalogue.fiware.org/enablers/repository-repository-ri>

**Documentation - Completeness & Soundness Test**

* There was some mismatch between Programmer’s Guide and Open Specification
* A course in FIWARE academy site and tutorial are available.
* The Docker was not available for installation.
* The Programmer’s Guide not always describes completely the interfaces.

**Documentation - Installation Test**

* The hardware and software pre-requirements are fully described.
* The GE manual allows to install the component easily

1. **Revenue Settlement and Sharing System - RSS RI**

The Revenue Settlement and Sharing System (RSS) GE is in charge of distributing the revenues originated by the usage of a given service among the involved stakeholders. It focuses on distributing part of the revenue generated by a service between the Store Provider and the Service Provider(s) responsible for the service.

The GE is present in the FIWARE catalogue at the following link:

<http://catalogue.fiware.org/enablers/revenue-settlement-and-sharing-system-rss-ri>

**Documentation - Completeness & Soundness Test**

* There was some mismatch between Programmer’s Guide and Open Specification
* A course in FIWARE academy site is also present
* The Docker is available for installation.
* The Programmer’s Guide not always describes correctly the interfaces. Some information was wrong.

**Documentation - Installation Test**

* The hardware and software pre-requirements are fully described.
* The GE manual allows to install the component easily

1. **Store - WStore**

The Store GE is mainly responsible for managing offerings and sales: it supports the publication of new offerings, manages offering payment, provides access to all purchased services and provides software downloads if the offering is part of a downloadable service.

The GE is present in the FIWARE catalogue at the following link:

<http://catalogue.fiware.org/enablers/store-wstore>

**Documentation - Completeness & Soundness Test**

* The Programmer’s Guide covers the Open Specification
* A specific course in FIWARE academy site and tutorial are present
* The Docker is available for installation.
* The Programmer’s Guide not always describes correctly the interfaces. Some information was wrong.

**Documentation - Installation Test**

* The hardware and software pre-requirements are fully described.
* The GE manual allows to install the component easily

1. **GIS Data Provider - Geoserver/3D**

This GE is able to host geographical data and serve it in 3D form (where applicable) to both mobile and web clients.

The GE is present in the FIWARE catalogue at the following link:

<http://catalogue.fiware.org/enablers/gis-data-provider-geoserver3d>

**Documentation - Completeness & Soundness Test**

* There was some mismatch between Programmer’s Guide and Open Specification
* A specific course in FIWARE academy site and tutorial are present
* The Docker was not available for installation.
* The Programmer’s Guide describes with examples the APIs. Some information needed to invoke interfaces is wrong.

**Documentation - Installation Test**

* The hardware and software pre-requirements are fully described.
* The GE manual allows to install the component easily

1. **POI Data Provider - POI Data Provider**

POI (Points of interest) Generic Enabler is a web server kit that supports storing information related to locations, serving queries by location and other criteria, can be configured to meet your data needs.

The GE is present in the FIWARE catalogue at the following link:

<http://catalogue.fiware.org/enablers/poi-data-provider>

**Documentation - Completeness & Soundness Test**

* The Programmer’s Guide covers the Open Specification
* A specific course in FIWARE academy site and tutorial are present
* The Docker is available for installation.
* The Programmer’s Guide describes with examples the APIs.

**Documentation - Installation Test**

* The hardware and software pre-requirements are fully described.
* The GE manual allows to install the component easily

1. **Synchronization - Synchronization**

The Synchronization Generic Enabler presents a lightweight and generic network-synchronized dynamic scene data model, and two communication protocols to interact with the model: SceneAPI, a RESTful HTTP API for non-realtime querying and modification of the scene, and a WebSocket-based bidirectional protocol for connected Web clients to receive continuous real-time scene updates, and to post their real-time changes to the scene.

The GE is present in the FIWARE catalogue at the following link:

<http://catalogue.fiware.org/enablers/synchronization>

**Documentation - Completeness & Soundness Test**

* There was some mismatch between Programmer’s Guide and Open Specification
* A specific course in FIWARE academy site and tutorial are present
* The Docker is available for installation.
* The Programmer’s Guide describes with examples the APIs.

**Documentation - Installation Test**

* The hardware and software pre-requirements are fully described.
* The GE manual allows to install the component easily

1. **Virtual Characters - Virtual Characters**

This GE consists of an open standard and reference implementation for virtual characters on the Web. Web applications will be able to create, display and animate virtual characters.

The GE is present in the FIWARE catalogue at the following link:

<http://catalogue.fiware.org/enablers/virtual-characters>

**Documentation - Completeness & Soundness Test**

* The Programmer’s Guide covers the Open Specification
* A specific course in FIWARE academy site and tutorial are present
* The Programmer’s Guide fully describes the APIs.

**Documentation - Installation Test**

* The hardware and software pre-requirements are fully described.
* The GE manual allows to install the component easily

1. **3D-UI-XML3D - 3D-UI-XML3D**

XML3D is an extension to HTML5 for declarative 3D content represented as a scene graph like structure inside the DOM. It provides web developers an easy way to create interactive 3D web applications.

The GE is present in the FIWARE catalogue at the following link:

<http://catalogue.fiware.org/enablers/3d-ui-xml3d>

**Documentation - Completeness & Soundness Test**

* The Programmer’s Guide covers the Open Specification
* A specific course in FIWARE academy site and tutorial are present
* The Programmer’s Guide fully describes the APIs.

**Documentation - Installation Test**

* The hardware and software pre-requirements are fully described.
* The GE manual allows to install the component.

1. **Augmented Reality - Augmented Reality**

The goal of the Augmented Reality Generic Enabler is to provide a high-level application programming interface, which can be used in developing various kinds of HTML5 Augmented Reality applications, which run on web browsers without any specific plug-ins. These applications rely on the functionality of the other GEs, like XML3D Technology, POI Data Provider, etc. Such AR applications will provide additional virtual content on top of the real world surroundings, and enhance the user’s view of the physical world with information that is not directly visible.

The GE is present in the FIWARE catalogue at the following link:

<http://catalogue.fiware.org/enablers/augmented-reality>

**Documentation - Completeness & Soundness Test**

* The Programmer’s Guide covers the Open Specification
* A specific course in FIWARE academy site and tutorial are present
* The Docker is not available for installation.
* The Programmer’s Guide not always describes completely the interfaces

**Documentation - Installation Test**

* The hardware and software pre-requirements are fully described.
* The GE manual allows to install the component easily

1. **Cloud Rendering - Cloud Rendering**

The goal of this GE is to provide a generic way to request, receive and control a video stream of a remote 3D application. The complexity and usual heavy performance requirements for a 3D application can be offloaded to a server, from a low end device that could not handle the rendering otherwise.

The GE is present in the FIWARE catalogue at the following link:

<http://catalogue.fiware.org/enablers/cloud-rendering>

**Documentation - Completeness & Soundness Test**

* The Open API Specifications are not yet available
* A specific course in FIWARE academy site is present
* The Docker is not available for installation.
* The Programmer’s Guide doesn’t describe completely the interfaces

**Documentation - Installation Test**

* The hardware and software pre-requirements are fully described.
* The GE manual allows to install the component easily

1. **Interface Designer - Interface Designer**

The goal of this GE is to provide an easy-to-use full manipulator / editor of 3D objects within a scene.

The GE is present in the FIWARE catalogue at the following link:

<http://catalogue.fiware.org/enablers/interface-designer>

**Documentation - Completeness & Soundness Test**

* The Programmer’s Guide covers the Open Specification
* No tutorial/course are available
* The Docker is available for installation.
* The Programmer’s Guide describes completely the interfaces

**Documentation - Installation Test**

* The hardware and software pre-requirements are fully described.
* The GE manual allows to install the component easily

1. **Network Information and Control - OFNIC**

OFNIC is a reliable and distributed Software Defined Network (SDN) controller for enterprises’ OpenFlow-enabled network. It enables the abstraction and virtualization of network resources and functionalities. It offers a RESTful interface in order to get information about the network topology components and elements either real or virtual. OFNIC also monitors the status of the network and provides near real-time data about network statistics with different levels of granularity (flow, node, port).

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/network-information-and-control-ofnic>

**Documentation - Completeness & Soundness Test**

* There was some mismatch between Programmer’s Guide and Open Specification
* A tutorial in academy site is missing
* The Docker is missing

**Documentation - Installation Test**

* The GE manual not allows to install the component.

1. **Kiara Advanced Middleware**

Kiara Advanced Middleware is a Java based communication middleware for modern, efficient and secure applications.

Kiara Advanced Middleware is a library which is incorporated into the developed applications, the requirements are rather minimal. In particular it requires no service running in the background.

The GE is present in the FIWARE catalogue at the following link

<http://catalogue.fiware.org/enablers/kiara-advanced-middleware>

**Documentation - Completeness & Soundness Test**

* The Programmer’s Guide covers the Open Specification
* A tutorial in academy site is missing.
* The Docker is missing.

**Documentation - Installation Test**

* The GE manual allows to install easily the component.
* The hardware and software pre-requirements are described.

### ***2.3.2 APIs testing***

The APIs testing activity consists of the following verifications:

o The Programmer’s Guide fully documents the Sw Package.

o All the Interfaces are fully described.

o Examples on interfaces' use are available.

o Changes with respect to previous versions are fully specified.

o All the restrictions are described.

o All the API commands are executed.

The current section reports the main conclusions for all GEs analyzed by functional tests.

1. **Security - Identity Management - KeyRock**

o The total number of tested API is 50.

o The main failures concern the missing information on documentation

o The installed software package fully implements the API declared into Open Specification.

o The specific call command for each interface is performed:

- the command to call the interface is not always completed.

- examples to call the interface are always present.

- the expected output or the input payload are always declared.

- the expected error codes are not always right.

o The time to taking charge of Jira issues was a bit slow.

o The time to fix of Jira issues was very fast.

1. **Security - Authorization PDP – AuthZForce**

o The total number of tested API is 15.

o The main failures (solved) concern the missing information on documentation

o The installed software package almost fully implements the API declared into Open Specification. Some interfaces were missing (solved) on Open Specification

o The specific call command for each interface is performed:

- examples to call the interface are present but in some cases they were not very clear

- the expected output or the input payload are always declared.

- the expected error codes are always right.

o The time to taking charge of Jira issues was acceptable.

o The time to fix of Jira issues was acceptable.

1. **Security - PEP Proxy – Wilma**

o The total number of tested API is 3.

o It was discovered an Integration problem with AZF GE (solved).

o The installed software package fully implements what is declared into Open Specification.

o The time to taking charge of Jira issues was very fast.

o The time to fix of Jira issues was acceptable.

1. **Data/Context Management - Context Broker – Orion**

o The total number of tested API is more than 40.

o The failures (solved) are generally due to some mismatch between Programmer’s Guide and Open Specification.

o The installed software package fully implements the API declared into the Open Specification.

o The specific call command for each interface is performed:

- the command to call the interface is clear.

- examples to call the interface are always present.

o The time to taking charge of Jira issues was excellent.

o The time to fix of Jira issues was excellent.

1. **Data/Context Management - BigData Analysis – Cosmos – Cygnus**

o The total number of tested API is 13.

o The installed software package fully implements the API declared into the Open Specification.

o The specific call command for each interface is performed:

- the command to call the interface is clear enough;

- examples to call the interface are always present;

- the expected output or the input payload are always declared. In some case there are little differences between the documentation and the real test, especially for CURL request

- The configurations in order to integrate the component with the others are not always described in a clear manner.

o The time to taking charge of Jira issues was very fast.

o The time to fix of Jira issues was very fast.

1. **Data/Context Management - Complex Event Processing (CEP) – Proton**

o The total number of tested API is 12.

o The failures were generally due to little mismatch in the documentation (solved).

o The installed software package fully implements the API declared into the Open Specification.

o The specific call command for each interface is performed:

- the command to call the interface is clear;

- examples to call the interface are always present;

- the expected output or the input payload are not always declared;

- the expected error codes are not always declared;

- the configurations in order to integrate the component with the others are not always;

- described in a clear manner.

o The time to taking charge of Jira issues was very fast.

o The time to fix of Jira issues was acceptable.

1. **Data/Context Management - Stream Oriented – Kurento**

o The total number of tested API is 8.

o The failures are generally due to mismatch in the documentation (solved).

o The installed software package fully implements the API declared into the Open Specification.

o The specific call command for each interface is performed:

- the command to call the interface is clear;

- examples to call the interface are always present;

- the expected output or the input payload are not always declared;

- the expected error codes are not always declared.

o The time to taking charge of Jira issues was excellent.

o The time to fix of Jira issues was fast.

1. **Internet of Things Services Enablement - Backend Device Management – IDAS (IoT Agents)**

o The total number of tested API is 65.

o They were found some functional or documental errors.

o The installed software package fully implements the API declared into the Open Specification.

o The specific call command for each interface is performed:

- the command to call the interface is clear;

- examples to call the interface are always present;

- the expected output or the input payload are not always declared;

- the expected error codes are not always declared;

- the configurations in order to integrate the component with the others are not always described in a clear manner.

o The time to taking charge of Jira issues was slow.

o The time to fix of Jira issues was slow.

1. **Internet of Things Services Enablement - IoT Broker – Aeron**

o The total number of tested API is 23.

o The failures are generally due to configuration about integration with other external components.

o The installed software package fully implements the API declared into the Open Specification.

o The specific call command for each interface is performed:

- the command to call the interface is clear enough.

- examples to call the interface are always present.

- the expected output or the input payload are not always declared.

- the expected error codes are declared.

o It’s useful to describe in a clear manner the payload parameters of an API in case they refer to configurations about integration with others external components (ex. tag providingApplication)

o The time to taking charge of Jira issues was sufficiently fast.

o The time to fix of Jira issues was a bit slow.

1. **Internet of Things Services Enablement - IoT Data Edge Consolidation – Cepheus**

o The total number of tested API is 6.

o The installed software package fully implements the API declared into the Open Specification.

o The specific call command for each interface is performed:

- the command to call the interface is clear enough;

- examples to call the interface are always present;

- the expected output or the input payload are not always declared.

o The configurations in order to integrate the component with the others are not always described in a clear manner.

o The time to taking charge of Jira issues was excellent.

o The time to fix of Jira issues was excellent.

1. **Cloud Hosting - Policy Manager - Bosun**

o The total number of tested API is 12.

o The installed software package fully implements the API declared into the Open Specification.

o The specific call command for each interface is performed:

- the command to call the interface is clear enough;

- examples to call the interface are always present;

- the expected output or the input payload are always declared. In some case there are little differences between the documentation and the real test.

o The configurations in order to integrate the component with the others are not always described in a clear manner.

o The time to taking charge of Jira issues was excellent.

o The time to fix of Jira issues was fast.

1. **Internet of Things Services Enablement – IoT Discovery**

o The total number of tested APIs is 28.

o The installed software package fully implements the API declared into the Open Specification.

o The specific call command for each interface is performed:

- the command to call the interface is clear enough;

- examples to call the interface are always present;

- the expected output or the input payload are not always declared.

o The configurations in order to integrate the component with the others are not always described in a clear manner.

o The time to taking charge of Jira issues was sufficiently fast.

o The time to fix of Jira issues was slow.

1. **DataContext - CKAN**

* The total number of tested APIs is 150.
* The failures are generally due to mismatch in the documentation.
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
* the command to call the interface is clear;
* examples to call the interface are always present;
* the expected output or the input payload are not always declared;
* the expected error codes are not always declared;
* the configurations in order to integrate the component with the others are not always described in a clear manner.

o The time to taking charge of Jira issues was very fast.

o The time to fix of Jira issues very fast.

1. **Data Visualization - SpagoBI**

* The total number of tested APIs s is 33.
* The main failures concern the missing information on documentation.
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
  + the command to call the interface is clearly described;
  + examples to call the interfaces are always present;
  + the expected output or the input payload are always declared.
* The time to taking charge of Jira issue was sufficiently fast.
* The time to fix of Jira issue was fast.

1. **Application Mashup - Wirecloud**

* The total number of tested APIs is 49
* The main failures concern the missing information on documentation
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
  + the command to call the interface is not always completed
  + examples to call the interfaces are present.
  + the expected output or the input payload are declared.
* The time to taking charge of Jira issue was slow.
* The time to fix of Jira issue was fast.

1. **Marketplace – Wmarket**

* The total number of tested API is 35.
* No relevant bugs found.
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
  + the command to call the interface is clearly described;
  + examples to call the interfaces are always present;
  + the expected output or the input payload are always declared;
  + the expected error codes are declared.
* The time to taking charge of Jira issue was a bit slow.
* The time to fix of Jira issue was fast.

1. **Repository - Repository RI**

* The total number of tested API is 12
* The main failures concern the missing information on documentation.
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
  + the command to call the interface is clearly described;
  + for some requests examples on interfaces' use are missing;
  + the expected output or the input payload are always declared.
* The time to taking charge of Jira issue was very slow.
* The time to fix of Jira issue was very slow.

1. **Revenue Settlement and Sharing System - RSS RI**

* The total number of tested API is 21
* The main failures concern the missing information on documentation
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
  + the command to call the interface is not always correctly described
  + examples to call the interfaces are present.
  + the expected output or the input payload are declared.
* The time to taking charge of Jira issue was very slow.
* The time to fix of Jira issue was very slow.

1. **Store - WStore**

* The total number of tested API is 59
* Some failures on the response of APIs were found
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
  + the command to call the interface is not always correctly described;
  + not always examples on interfaces' use are available;
  + the expected output or the input payload are declared.
* The time to taking charge of Jira issue was very slow.
* The time to fix of Jira issue was very slow.

1. **GIS Data Provider - Geoserver/3D**

* The total number of tested API is 4
* The main failures concern the missing information on documentation.
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
  + the command to call the interface is not always correctly described;
  + examples to call the interfaces are present;
  + the expected output or the input payload are always declared.
* The time to taking charge of Jira issue was a bit slow.
* The time to fix of Jira issue was fast.

1. **POI Data Provider - POI Data Provider**

* The total number of tested APIs is 9.
* The main failures concern some missing information on documentation.
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
  + Some errors were found in the command to call the interfaces;
  + examples to call the interfaces are always present;
  + the expected output or the input payload are always declared.
* The time to taking charge of Jira issue was fast.
* The time to fix of Jira issue was fast.

1. **Synchronization - Synchronization**

* The total number of tested APIs is 32.
* No relevant bugs found.
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
  + the command to call the interface is correctly described;
  + examples to call the interfaces are always present;
  + the expected output or the input payload are always declared.
* The time to taking charge of Jira issue was fast.
* The time to fix of Jira issue was fast.

1. **Virtual Characters - Virtual Characters**

* The total number of tested APIs is 9.
* No relevant bugs found.
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
  + the command to call the interface is correctly described;
  + examples to call the interfaces are always present;
  + the expected output or the input payload are always declared.

1. **3D-UI-XML3D - 3D-UI-XML3D**

* The total number of tested APIs is 12.
* No relevant bugs found.
* The installed software package fully implements the APIs declared into the Open Specification.
* The specific call command for each interface is performed:
  + the command to call the interface is correctly described;
  + examples to call the interfaces are always present;
  + the expected output or the input payload are always declared.
* The time to taking charge of Jira issue was very slow.
* The time to fix of Jira issue was sufficiently fast.

1. **Augmented Reality - Augmented Reality**

* The total number of tested API is 27
* The main failures concern the missing information on documentation.
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
  + some interfaces are not described
  + examples to call the interfaces are present;
  + the expected output or the input payload are declared.
* The time to taking charge of Jira issue was very slow.
* The time to fix of Jira issue was very slow.

1. **Cloud Rendering - Cloud Rendering**

* The total number of tested APIs is 14
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
  + the interfaces are described
  + examples to call the interfaces are present;
  + the expected output or the input payload are declared.
* The time to taking charge of Jira issue was very fast.
* The time to fix of Jira issue was very fast.

1. **Interface Designer - Interface Designer**

* The total number of tested APIs is 13.
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
  + the interfaces are described;
  + examples to call the interfaces are present;
  + the expected output or the input payload are declared.
* The time to taking charge of Jira issue is very fast.
* The time to fix of Jira issue was fast.

1. **Network Information and Control - OFNIC**

* The APIs are not tested because of bugs not solved.
* The time to taking charge of Jira issue was sufficiently fast.
* The time to fix of Jira issue was very slow.

1. **Kiara Advanced Middleware**

* The total number of tested APIs is 41.
* The installed software package fully implements the API declared into the Open Specification.
* The specific call command for each interface is performed:
  + the interfaces are described;
  + examples to call the interfaces are present;
  + the expected output or the input payload are declared.
* The time to taking charge of Jira issue was very slow.
* The time to fix of Jira issue was very slow.

### ***2.3.3 Integration tests***

The functional bundle integration test activity has been divided in three steps:

1. Functional Bundle Analysis

2. Functional Bundle Planning

3. Functional Bundle Execution

**1.** **Functional Bundle Analysis**

The task Functional Bundle Analysisperforms the analysis of the functional integration scenarios among different GEs.

The GEs that compose the integrated FIWARE bundle where the functional integration tests are performed are:

|  |  |
| --- | --- |
| Chapter | GE |
| Security | Identity Management - KeyRock |
| PEP Proxy – Wilma |
| Authorization PDP – AuthZForce |
| Internet of Things Services Enablement | Backend Device Management – IDAS  IoT Agent Ultraligth2.0/HTTP |
| Data/Context Management | Publish/Subscribe Context Broker - Orion |
| Complex Event Processing (CEP) - Proactive Technology Online (Proton) |
| Applications/Services and Data Delivery | Data Visualization - SpagoBI |

The following figure shows the Bundle Architecture:

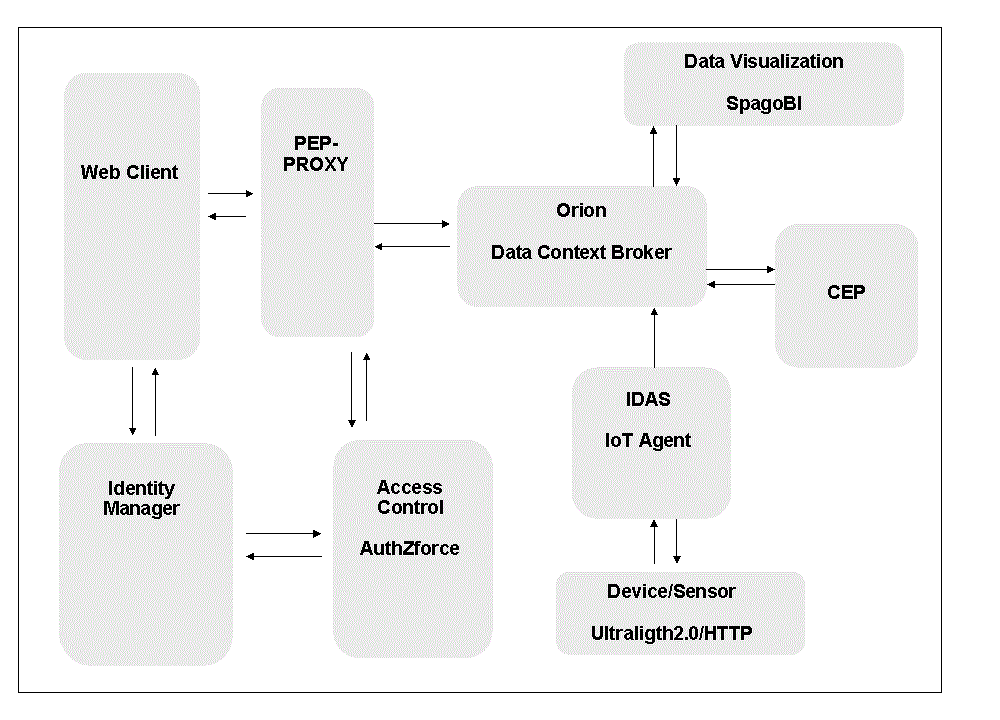


Figure 5: Bundle Architecture used for functional integration testing

**2.** **Functional Bundle Planning**

The task Functional Bundle Planningperforms the activities of planning and writing of the test cases related to the functional integration scenarios.

The functional scenarios combine some of the main GE interfaces to highlight the interaction among the GEs of the bundle platform.

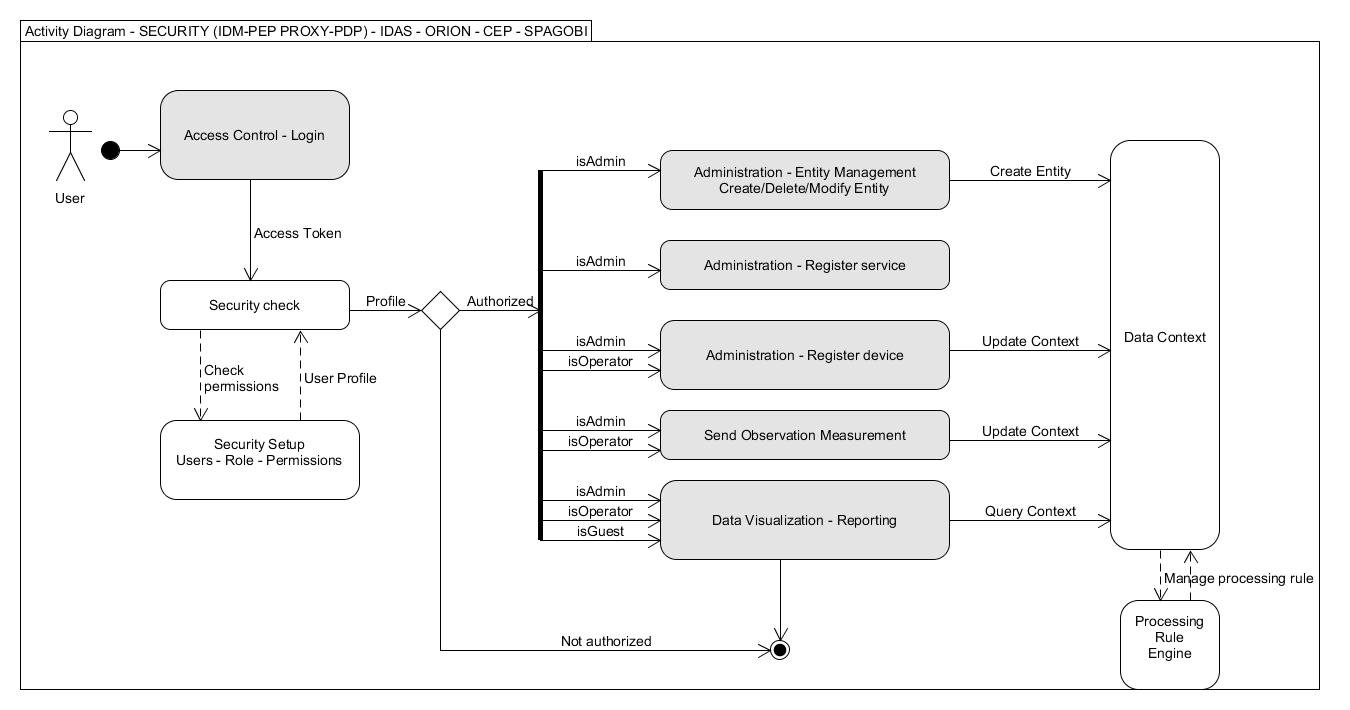
The integration tests simulate a real use case.

Here follows the list of the test cases grouped by functional scenario.

|  |  |
| --- | --- |
| Functional Scenario | Test case Description |
| Security setup | Register User |
| Register Application |
| Register new PEP Proxy |
| Access Control | Login as Guest |
| Login as Operator |
| Login as Administrator |
| Entity Management | Create Entity Parking |
| Remove Entity Parking |
| Modify Entity Parking - Access Permit |
| Modify Entity Parking - Access Denied |
| Get Entity Parking |
| Service Registration | Create service Parking |
| Device Registration | Create device: Sensor sCarEntry |
| Create device: Sensor sCarExit |
| Create device: Sensor sCO2 |
| Observation Measurement | Sensor sCarEntry detects a car entry (parking not full) |
| Sensor sCarEntry detects a car entry (parking full) |
| Sensor sCarExit detects a car exit |
| Sensor sCO2 measures the CO2 level |
| Data visualization | Dynamic Report - Parking data |
| Static Report – Parking data |
| Static Report – Parking Statistics |

The scenarios are organized following a simple business logic imagining the functional actions applied in a parking management application.

Here follows the diagram to clarify the simple functional business logic implemented in order to perform the functional bundle integration tests.



**3.** **Functional Bundle Execution**

The Functional Bundle Execution taskperforms the activities of running functional integration's scenarios based on the bundle platform.

The functional scenarios use some of the main GE interfaces to test the interaction among the GEs of the platform. Here follows the list of the scenarios grouped by functional area contents.

Each Integration Test Scenario is a test case containing a list of steps explaining the flow of the communication through the GEs and for each test case is reported the log of the main process steps.

Here follows the summary result table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Suite** | **ID**  **TestSuite\_TestCase** | **Test case** | **State** |
| Security setup | ‎TS01\_TC01 | Register User | Passed |
| ‎TS01\_TC02 | Register Application | Passed |
| ‎TS01\_TC03 | Register new PEP-PROXY | Passed |
| Access Control | ‎TS02\_TC01 | Login as Guest | Passed |
| ‎TS02\_TC02 | Login as Operator | Passed |
| ‎TS02\_TC03 | Login as Administrator | Passed |
| Entity Management | ‎TS03\_TC01 | Create Entity Parking | Passed |
| ‎TS03\_TC02 | Remove Entity Parking | Passed |
| ‎TS03\_TC03 | Modify Entity Parking - Access Permit | Passed |
| ‎TS03\_TC04 | Modify Entity Parking - Access Denied | Not Passed |
| ‎TS03\_TC05 | Get Entity Parking | Passed |
| Service Registration | ‎TS04\_TC01 | Create service Parking | Passed |
| Device Registration | ‎TS05\_TC01 | Create device: Sensor sCarEntry | Passed |
| ‎TS05\_TC02 | Create device: Sensor sCarExit | Passed |
| ‎TS05\_TC03 | Create device: Sensor sCO2 | Passed |
| Observation Measurement | ‎TS06\_TC01 | Sensor sCarEntry detects a car entry (parking not full) | Not Passed |
| ‎TS06\_TC02 | Sensor sCarEntry detects a car entry (parking full) | Not Passed |
| ‎TS06\_TC03 | Sensor sCarExit detects a car exit | Not Passed |
| ‎TS06\_TC04 | Sensor sCO2 measures the CO2 level | Passed |
| Data visualization | ‎TS07\_TC01 | Dynamic Report - Parking data | Passed |
| ‎TS07\_TC02 | Static Report – Parking data | Passed |
| ‎TS07\_TC03 | Static Report – Parking Statistics | Passed |

Each test case reports the evidence of the main actions performed through the system involved.

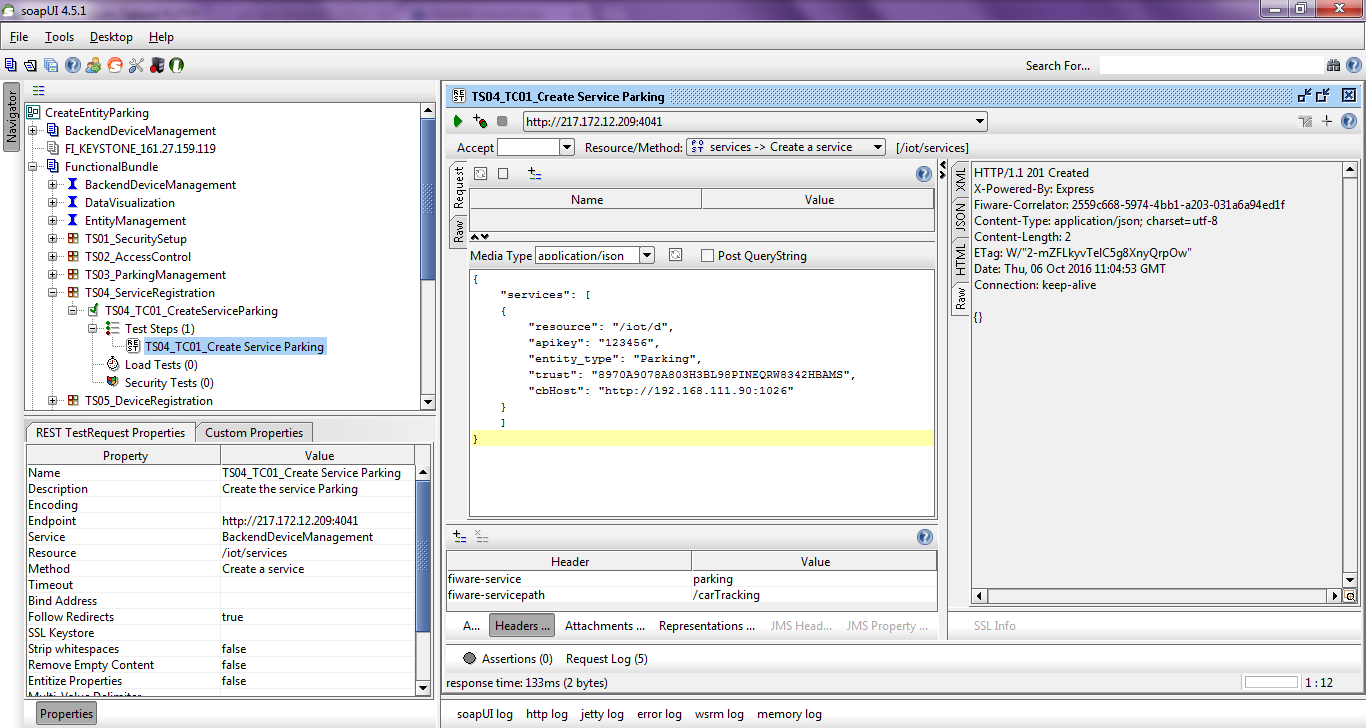
Here follows a test case by example.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| \*Tcase Title | | Create device: Sensor sCarEntry | | | TS/TC Ref |  | |
| Description | | Register a new device corresponding to the sensor sCarEntry able to send an observation for each new car that enters into the parking.  The client application sends an HTTP POST request towards IDAS component. IDAS registers the device itself in the Data Context Broker as the Context Provider. Context Broker sends back a response code.  The sensor is called sCarEntry and is identified by  "device\_id": "sCarEntry"  "entity\_name": "BrinParking" "entity\_type": "Parking"  "attributes": "name": "carMoving" | | | Priority | H | |
| Pre-Requirements | | IDAS - IoTAgent is Up and running.  Orion Data Context Broker is up and running.  Login as ADMIN user to the Client Application.  Entity Parking already exists in Orion CB. | | | | | |
| Input Data | | The client application runs an HTTP POST request towards IDAS component to register the new device Sensor sCarEntry. | | | | | |
| TCase ID | Component | | | Actions to execute / Expected Outputs | | | Next TCase |
|  | Client Application | | | User ADMIN sends an HTTP POST request towards IDAS component:  curl --request POST --header "Content-Type: application/json" --header "fiware-service: parking" --header "fiware-servicepath: /carTracking" --data '{ "devices": [ { "device\_id": "sCarEntry", "entity\_name": "BrinParking", "entity\_type": "Parking", "attributes": [ { "name": "carMoving"} ] } ] }' http://IP\_ADDRESS:4041/iot/devices | | |  |
|  | IDAS – IoTAgent | | | The IoTAgent receives the request.  The IoTAgent registers the new sensor sCarEntry.  The IoTAgent sends back the response code success or failure to the sender. | | |  |
|  | Client Application | | | User ADMIN receives the success or failure message. | | |  |
| Validation criteria | | | Verify in the Log files of IoTAgent the trace messages about the sequence of the transactions.  msg=Device provisioning request succeeded | | | |  |
| Evidence | | | Log messages:  msg=Device provisioning request succeeded | | | |  |

The whole detail results are reported in the ANNEX 2.

The integration tests have been performed using the tool soapUI 4.5.1 to simulate a Client Application similar to a real application. The soapUI xml file contains the procedures to run the tests improving the test automation. Some tests are traced as manual steps, because they are performed using the IDM Account Portal, the others are REST Requests. The soapUI xml file is public and stored in the [Github functional test repository](https://github.com/Fiware/test.Functional/tree/master/API.test).

Here follows the screenshot of soapUI project produced to automate the test cases.



### ***2.3.4 Academy courses testing***

The testing activity of the FIWARE Academy courses is the task performed to evaluate the efficiency of training.

The courses are publicly available at edu.fiware.org, and any user interested in using FIWARE can easily understand whether a given course does fit the user's need or not.

The training material created and published to the FIWARE Academy was evaluated on the basis of the criteria listed in the guidelines to be followed by the editor of the course for granting an efficient training offering. The guidelines are publicly available in the FIWARE wiki page “working with the FIWARE Academy# Course Evaluation for Efficient Training”[[1]](#footnote-1) , and the GE Owners are asked to follow those guidelines, reported below.

The criteria to evaluate the efficiency of training are listed below.

1. The course title has to contain the GE/GEis name (if related to a GE).

\* *Description*: The course title must be explicitly referred to the GE. E.g.: “GE Name (GE Implementation Name)”.

\* *Type*: Mandatory | F

2. The GE version must be evident and the course has to be updated to the latest version.

\* *Description*: Specify in the course which is the GE version to which the course refers to. Make sure that the course is up to date with the latest GE version.

\* *Type*: Mandatory | QA

3. Each lesson has to clearly define its purpose and explaining its contents in exhaustive way.

\* *Description*: Each lesson (topic) must be self-consistent and clearly describe its purpose: what does the lesson teach? Which audience or community of users could be interested to this lesson?

\* *Type*: Recommended | QA

4. The course must have an introduction about the GE/the context with a summary of the course itself.

\* *Description*: A clear introduction explaining what

\* *Type*: Mandatory | F

5. The course must have an introduction about the GE/the context with a summary of the course itself.

\* *Description*: The introduction should describe the context and or the GE, and the scope of the course. It has to explain the purpose and the structure of the whole course

\* *Type*: Mandatory | QA

6. An overview lesson giving a high level description of the Architecture must be present.

\* *Description*: An overview lesson on the general context is required.

\* *Type*: Mandatory | F

7. An overview lesson giving a high level description of the Architecture must be present.

\* *Description*: The overview lesson has to describe at a high level the component or GE object of the course, the reference context, its architecture, its functionalities.

\* *Type*: Mandatory | QA

8. Specific technical lessons are required, on main functionalities having dedicated external resources.

\* *Description*: The course must be organized with specific technical lessons on specific topics with dedicated external resources (video, audio, slides, docs, etc.). A description of each lesson (topic) is required.

\* *Type*: Mandatory | F

9. Specific technical lessons are required, on main functionalities having dedicated external resources.

\* *Description*: The course must be organized with dedicated lessons describing single or group of functionalities, on specific topics, explaining how to implement them and to work with them. Each lesson has to include dedicated resources, declaring their type (video, audio, slides, docs, etc.).

\* *Type*: Mandatory | QA

10. Video, Audio, Slides presentations must be not too long.

\* *Description*: The duration of each audio and or video resource included in the course must be about 10-15 minutes. It’s not acceptable to present the entire course in a unique video or slide presentation.

\* *Type*: Mandatory | QA

11. Each lesson must have a meaningful title and the same for each (external) resource added to the lesson.

\* *Description*: Each lesson (topic) created must have a significant title (it’s not acceptable to read “topic 1”, “topic 2”, or “lesson 1”, “Lesson 2”, etc.). Each external resource included must have a meaningful title too.

\* *Type*: Mandatory | F

12. External links must be available and have to contain updated information.

\* *Description*: Check any external link added to the course and verify that it does work and does contain information up to date.

\* *Type*: Mandatory | QA

13. A short summary for displaying info (i) is required.

\* *Description*: The summary of the course (field “description”) must be present giving a short overview of the course.

\* *Type*: Mandatory | F

14. A short summary for displaying info (i) is required.

\* *Description*: The summary/info (i) section (field “description”) must contain a brief statement on the topic/content (GE) the course deals with and a short summary on the course itself (its goals and structure). The summary must be useful to understand whether it is what the user is searching for.

\* *Type*: Mandatory | QA

15. Lessons should contain external resources consisting of videos and/or (audio)slides.

\* *Description*: It is strongly recommended to use multimedia content, such as audio, video, slides, etc. to give support to those willing to make use of the training course. Try to avoid to publish lessons having only written text or documentation explaining the topic.

\* *Type*: Recommended | QA

16. The course should make available Use Cases, Examples and Exercises.

\* *Description*: Within the course it is recommended to introduce dedicated lessons or resources to describe specific Use Case stories related to the GE and reference context, as well as dedicated examples or exercises for implementation.

\* *Type*: Recommended | QA

17. Availability of installation procedures, pre-requirements, and basic configurations.

\* *Description*: Within the course should be clearly mentioned references to any installation procedures, pre-requirements, and basic configurations.

\* *Type*: Recommended | QA

18. Availability of technical support, discussion forum, “more info”.

\* *Description*: Add to the course: (a) links to forums; (b) contact email for technical support; (c) useful links to get more information.

\* *Type*: Recommended | QA

Some criteria may have same definition but meaning different checks from a formal and a QA review (see the *description*).

*The Formal (F) requirements* are the minimum set of recommendations to be followed for creating/updating a training course. Formal criteria are essentially related to the structure of the course and to its efficient output. On the basis of those requirements FIWARE Academy courses have always been reviewed within the task for Training content development and organization.

The *Quality Assurance (QA) requirements* were added to the formal ones and correspond to the QA criteria to be implemented in order to have a good quality of the course. On the basis of QA criteria the course is reviewed and gets the “Not Good/Sufficient/Good” label by the QA team.

Of course, in most cases QA requirements are complementary to the formal ones and their verification cannot be split, and also the fulfillment of formal requirements could influence the QA label of the course.

The following table shows the list of the academy courses evaluated and the relative results.

|  |  |  |  |
| --- | --- | --- | --- |
| **Chapter** | **Course ID** | **Status** | **Score** |
| **Security** |  |  |  |
| Identity Management | [79](https://edu.fiware.org/course/view.php?id=79) | Completed | Good |
| PEP Proxy | [131](https://edu.fiware.org/course/view.php?id=131) | Completed | Good |
| Authorization PDP | [144](https://edu.fiware.org/course/view.php?id=144) | Completed | To Improve |
| Access Control (OAUTH-API-AZ) | [57](https://edu.fiware.org/course/view.php?id=57) | Completed | To Improve |
| **Applications/Services** |  |  |  |
| Apps and Services Overview | [52](https://edu.fiware.org/course/view.php?id=52) | Completed | Sufficient |
| DataVisualization | [141](https://edu.fiware.org/course/view.php?id=141) | Completed | Sufficient |
| Application Mashup | [53](https://edu.fiware.org/course/view.php?id=53) | Completed | Good |
| Marketplace | [21](https://edu.fiware.org/course/view.php?id=21) | Completed | Sufficient |
| Repository | [127](https://edu.fiware.org/course/view.php?id=127) | Completed | Sufficient |
| Revenue Settlement and Sharing System | [117](https://edu.fiware.org/course/view.php?id=117) | Completed | To Improve |
| Store | [104](https://edu.fiware.org/course/view.php?id=104) | Completed | Sufficient |
| **Data/Context Management** |  |  |  |
| Context Broker | [132](https://edu.fiware.org/course/view.php?id=132) | Completed | Good |
| Context Broker (2) | [44](https://edu.fiware.org/course/view.php?id=44) | Completed | Sufficient |
| Context Broker (5) | [149](https://edu.fiware.org/course/view.php?id=69) | Completed | Sufficient |
| Big Data | [69](https://edu.fiware.org/course/view.php?id=69) | Completed | Sufficient |
| Complex Event Processing | [58](https://edu.fiware.org/course/view.php?id=58) | Completed | Good |
| Stream Oriented | [62](https://edu.fiware.org/course/view.php?id=62) | Completed | Good |
| Short Term Historic Open Data Repository (CKAN) | [145](https://edu.fiware.org/course/view.php?id=145) | Completed | To Improve |
| **Interface to Networks and Devices (I2ND)** |  |  |  |
| Network Information And Control (OFNIC) | [72](https://edu.fiware.org/course/view.php?id=72) | Completed | To Improve |
| Advanced Middleware (Kiara) | [140](https://edu.fiware.org/course/view.php?id=140) | Completed | To Improve |
| **Internet of Things (IoT) Services Enablement** |  |  |  |
| Backend Device Management (IDAS) | [128](https://edu.fiware.org/course/view.php?id=128) | Completed | Sufficient |
| IoT Broker | [33](https://edu.fiware.org/course/view.php?id=33) | Completed | Sufficient |
| IoT Data Edge Consolidation | [36](https://edu.fiware.org/course/view.php?id=36) | Completed | Sufficient |
| IoT Discovery | [40](https://edu.fiware.org/course/view.php?id=40) | Completed | Sufficient |
| **Advanced WebUI** |  |  |  |
| 3D-UI-XML3D | [97](https://edu.fiware.org/course/view.php?id=97) | Completed | To Improve |
| Cloud Rendering | [92](https://edu.fiware.org/course/view.php?id=92) | Completed | Sufficient |
| GIS Data Provider | [88](https://edu.fiware.org/course/view.php?id=88) | Completed | To Improve |
| Interface Designer | [91](https://edu.fiware.org/course/view.php?id=91) | Completed | Sufficient |
| POI Data Provider | [96](https://edu.fiware.org/course/view.php?id=96) | Completed | Sufficient |
| Synchronization | [111](https://edu.fiware.org/course/view.php?id=111) | Completed | Sufficient |
| Virtual Characters | [112](https://edu.fiware.org/course/view.php?id=112) | Completed | Sufficient |
| **Cloud Hosting** |  |  |  |
| Policy Manager | [119](https://edu.fiware.org/course/view.php?id=119) | Completed | Sufficient |

Table 6: Results of academy courses testing

## **2.4 NGSI APIs testing**

This section describes the research work performed by EGM about using model based testing (MBT) for generating automating tests to verify the compliance of FIWARE GEs APIs with NGSI standard.

### ***2.4.1 Introducing MBT in FIWARE (NGSI Model)***

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
* Unit testing won’t be sufficient to check the functionalities
* To ensure that the system is behaving in the same sequence of actions.

In this work we are interested in the NGSI compliant Generic Enablers (GE) of FIWARE, and more specifically in verifying their compliance to the standard.

The fundamental MBT process 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.

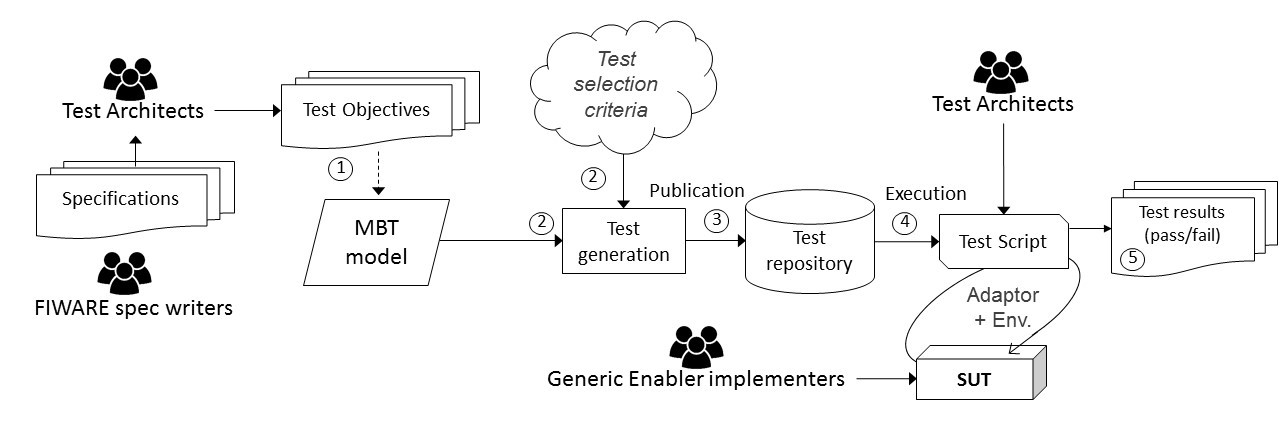


Figure 6: MBT Process

In this section we present the MBT process we used for testing a FIWARE generic enabler implementation. As given in the preceding Figure, 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). In the following figure, we have an example of requirement extraction from the specification. The specification states that implementation of NGSI v2 should be able to list entities.

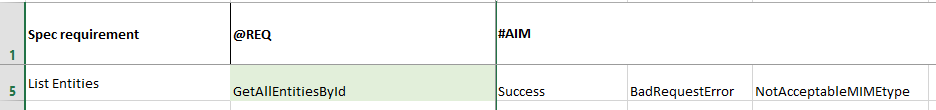


Figure 7: Test Objective Charter Example on "getAllEntitiesById" NGSI v2 operation

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 other 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 (step 5) and feedback is sent to the user.

In our approach for NGSI compliance testing we used coverage-based test selection criteria and it considers, among others, UML class and object diagrams to develop MBT models. Each type has a separate role in the test generation process. Note that among other test generation tools, we are using Smartesting CertifyIt as tests generator. 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 (in our example: NGSI v2 implementation in a GE)
* Observations (usually denoted as check operations) that serve as oracles, for instance an observation returns the current state of the user's connection on a web site

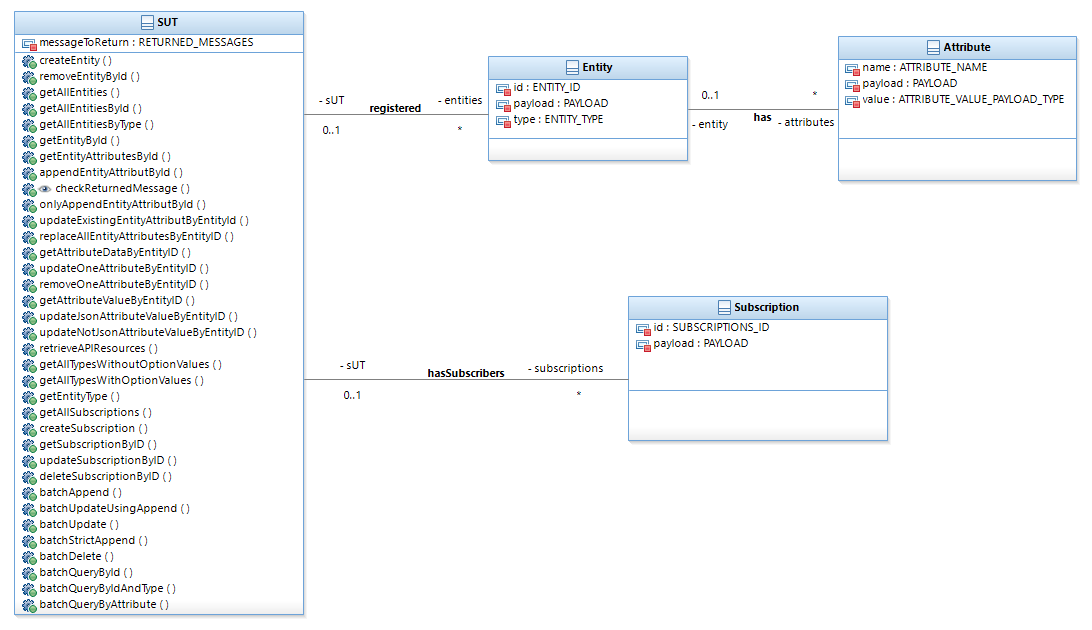


Figure 8: UML Model applied to NGSI v2

This static view, given by the class diagram, is instantiated by an object diagram. The object diagram provides the initial status of the system and also all objects that will be used in the test input data as parameters for the operations in the generated tests.

The **dynamic view** of the system or its behaviors is described by Object Constraint Language (OCL) constraints written as pre/post-condition in operations in a class of a class diagram. The test generation engine sees these behavior objects as test targets.

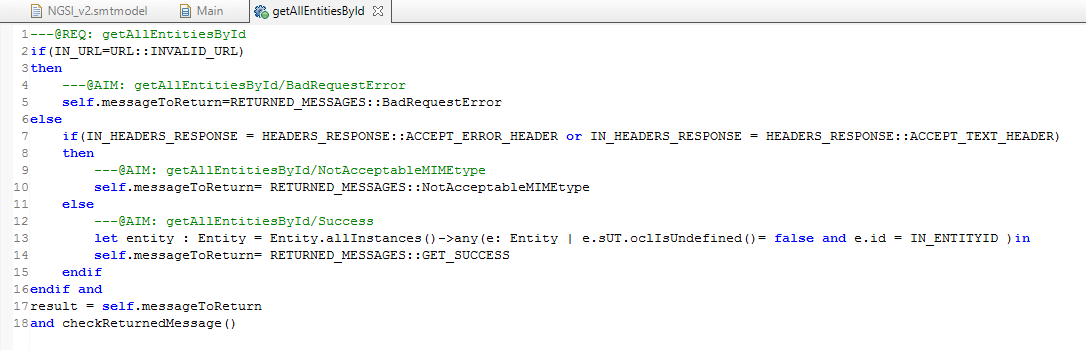


Figure 9: OCL example for “getAllEntitiesByID” NGSI v2 operation

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 exist:

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

Both followed by an identifier. Finally, one major advantage is the possibility to automatically deduce the test oracle[[2]](#footnote-2). A specific type of operations, called observations, defines 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.

As discussed previously each test has a set of tags associated for which it will ensure the coverage. The following figure, gives a snapshot of generated tests. On the left side, the test generation tool lists all generated tests clustered per covered requirement. On the right side, we can visualize a 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, "checkReturnedMessage" observes the return code of each NGSI v2 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).

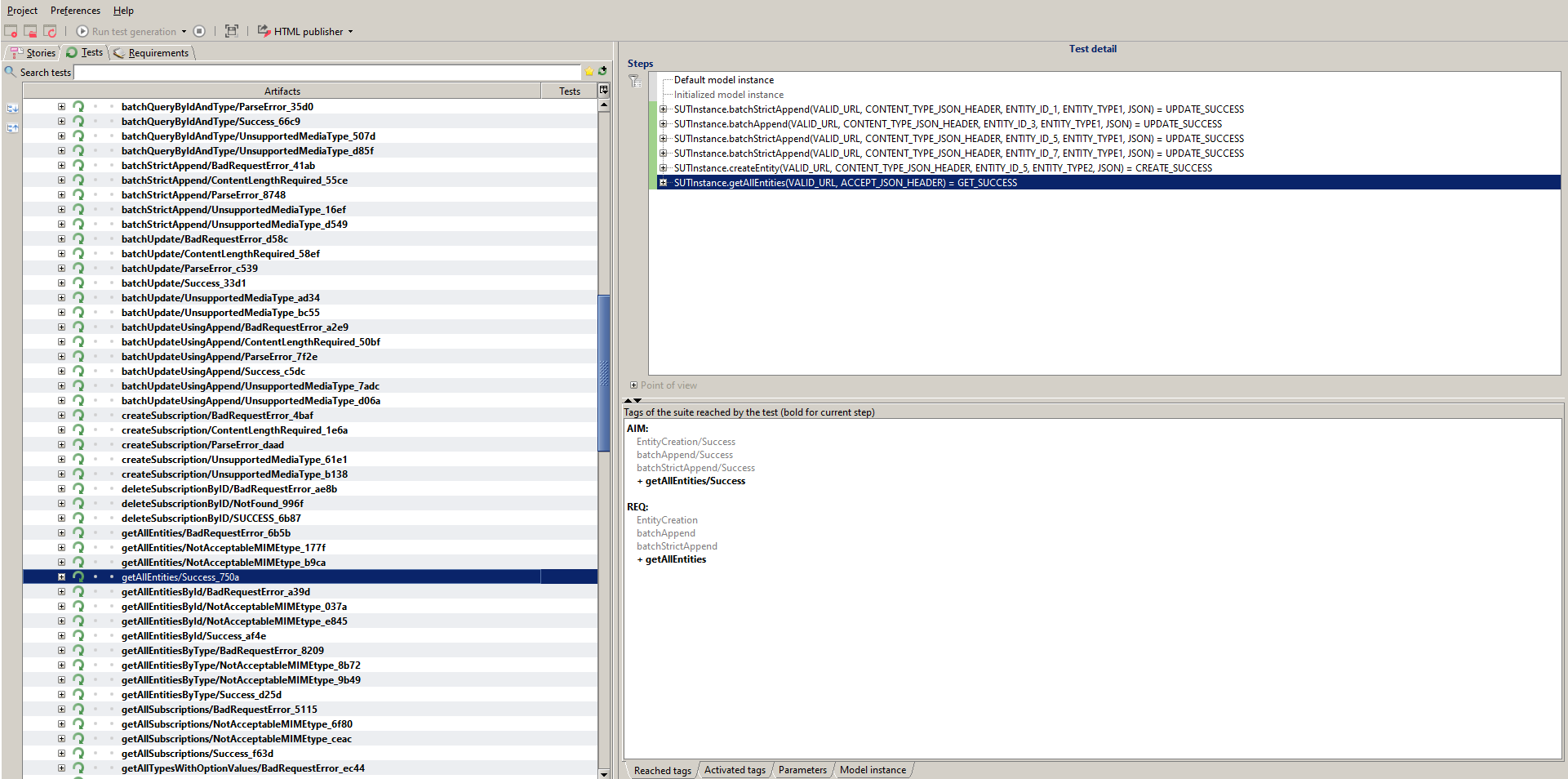


Figure 10: Excerpt of Generated Tests

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

### ***2.4.2 Test Adaptation and 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 adaptation activities, we first exported the abstract test cases into executable script, in our case tests in XML that can be automatically imported and executed.

In our early developments, we started exporting generated tests in the form of SoapUI projects. As a matter of fact, it is adapted to NGSI Restful API testing. The tool has its advantages, with the automation of tests suites executions.

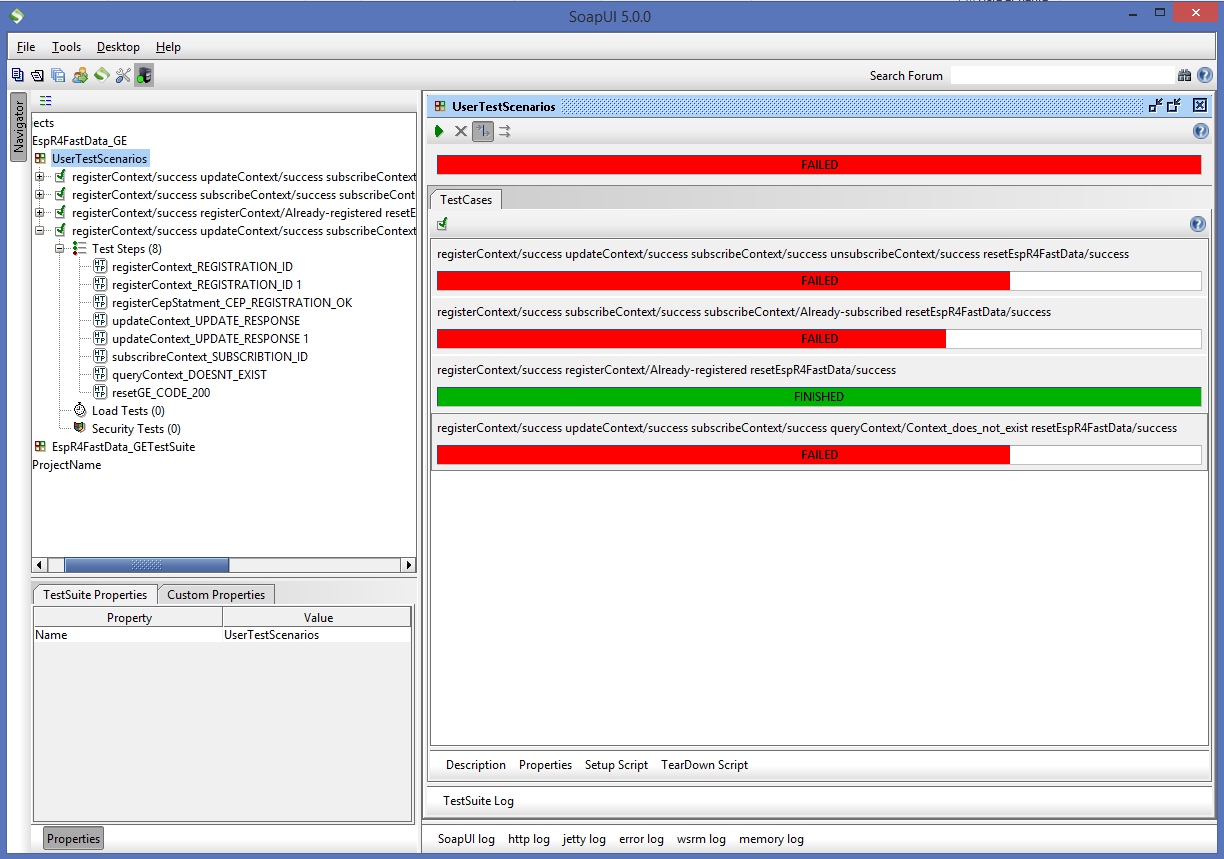


Figure 11: SoapUI tests on NGSI v1 experiment

On the other hand, we were quickly limited by the tool for a full automation process. Having a graphical user interface meant that launching the test campaigns had to be done by a tester. After further investigation, we discovered SoapUI APIs that enables the creation of REST requests without the GUI. We tried the JAVA version of the SoapUI API. And still, results obtained were unsatisfying as the process is heavy in time, little support and not modular. To resolve the problem, we decided to implement our own solution: EGM\_TAAS.

EGM\_TAAS stands for Easy Global Market Testing As A Service. It’s developed on a cloud micro-services based architecture.

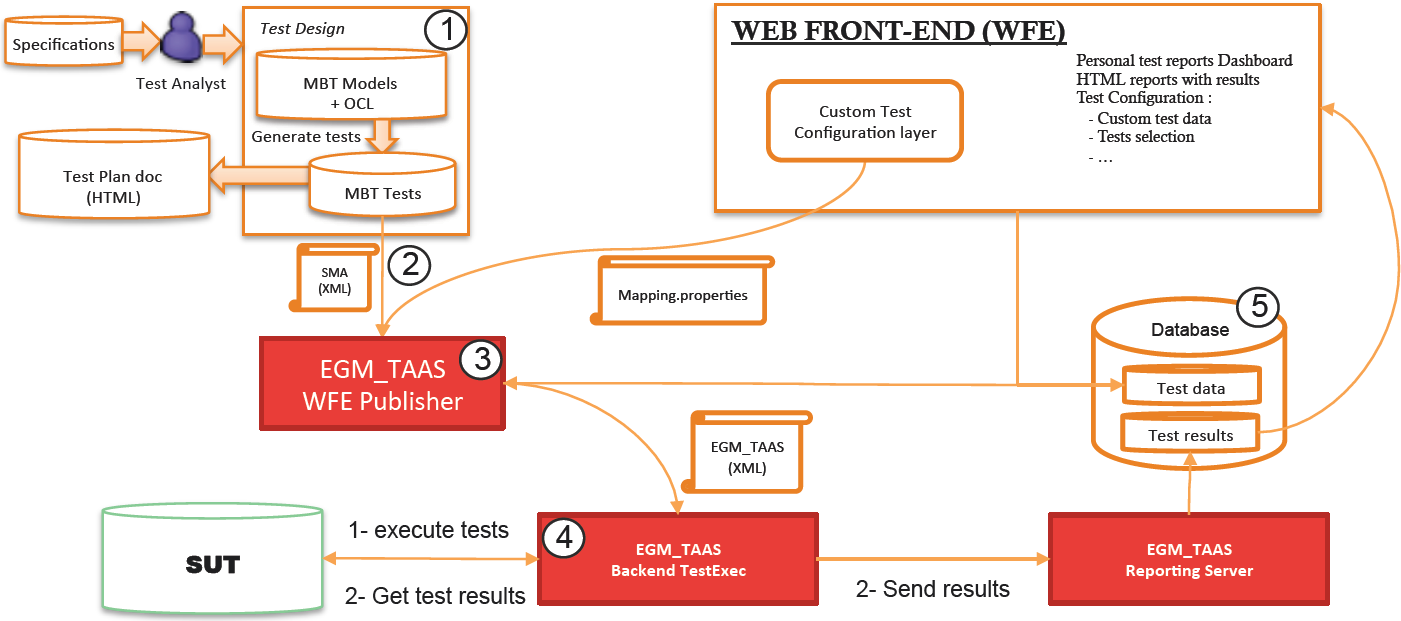


Figure 12: EGM\_TAAS micro-services architecture

In the previous figure we find the four main steps of the MBT approach (MBT modeling, test generation, test implementation and execution). However, to the difference of a classical MBT process, MBTAAS implements several webservices, which communicate with each other in order to realize testing steps. A webservice, uses web technology such as HTTP, for machine-to-machine communication, more specifically for transferring machine readable file formats such as XML 4 and JSON 5. In addition to the classical MBT process, the central piece of the architecture is the database service (5) that is used by all the other services. The database service can be separated from all the other services, it can be running in the cloud where it is accessible. The database stores important information such as test data (input data for test execution) and test results. The entry point of the system is the web-front end service (customization service). This service takes a user input to customize a testing session and it communicates it to a Publication service (3). The purpose of the publisher service is to gather the MBT results file and user custom data in order to produce a customized test description file (EGM TAAS file). This file is then sent to an Execution service (4) which takes in charge the execution of the customized tests. It stimulates the SUT with input data in order to get response as SUT output data. The execution service then finally builds a result response and sends it to a last service, the reporting service. The reporting service is configured to feed the database service with the test results. These test results are used by the web-front end service in order to submit them to the end-user. This testing architecture is taken to a modular level in order to respond to the heterogeneity of an IoT platform.

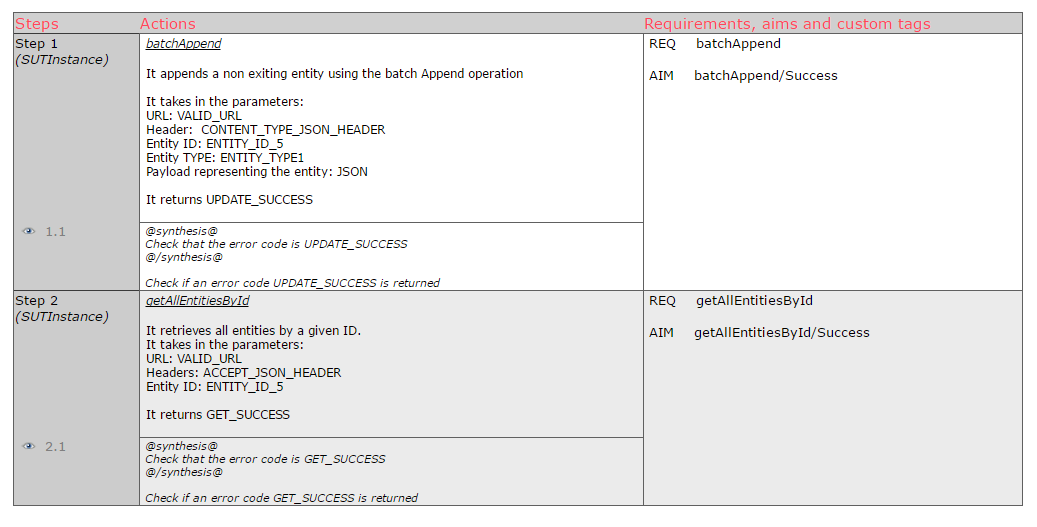
The next session describes and discusses the test results gathered on NGSI.

### ***2.4.3 Test results discussion***

In this section the test results will be presented. As mentioned above a set of abstract tests have been generated using CertifyIT from the model. In the first subsection, the abstract tests will be presented and in the second, the real test execution result.

#### **2.4.3.1** **Generated Abstract tests**

As detailed in the first sub-section an abstract test cases has been generated from the NGSI v2 model. The following figure illustrates the abstract test case “getAllEntitiesById\_Success\_af4e”. This test case contains two test steps “batchAppend” and “getAllEntitiesById”. The inputs and outputs of each test step is defined although no real values are related to them.



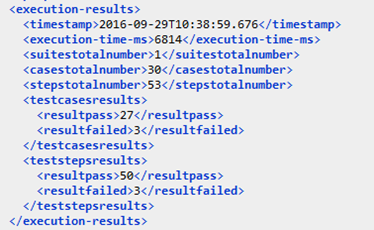
#### **2.4.3.2** **Test result execution**

##### NGSi v1

The generated tests from the NGSI v1 model have been executed on Orion GEi version 1.2.1, IoT Broker GEi and Cepheus Broker version 0.1.8.

**Orion 1.2.1**

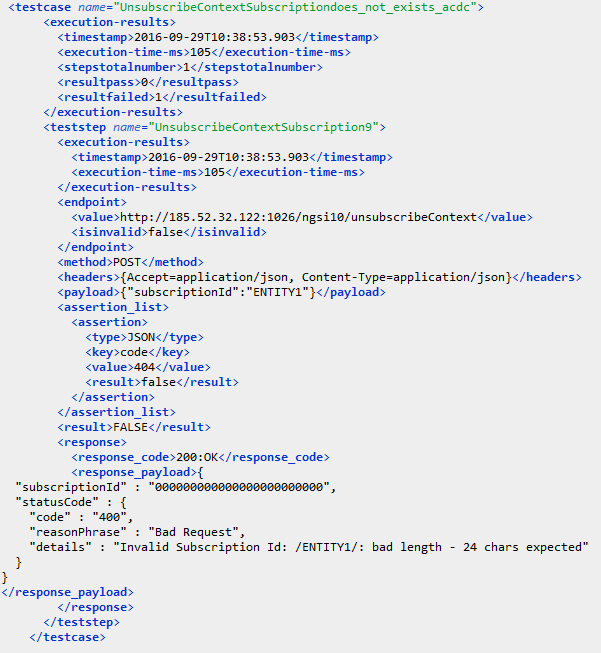
The following figure provides the test suite result.



30 test cases have been generated. 3 have failed. The failure causes are either specification issues or implementation issues.

***The specification issues***

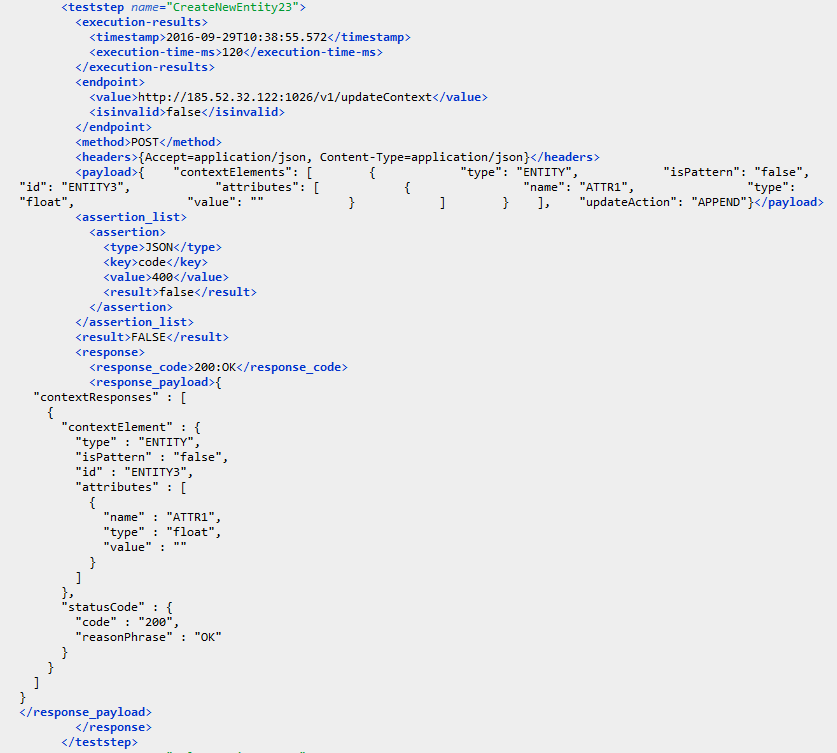
The specification does not detail the error to raise in case of subscription does not exist. In the test case *“UnsubscribeContextSubscriptiondoes\_not\_exists\_acdc”,* the test step *“UnsubscribeContextSubscription9”.* The wanted result is “*NotFound*” although the returned result is “*BadRequest*”.



***The implementation issues***

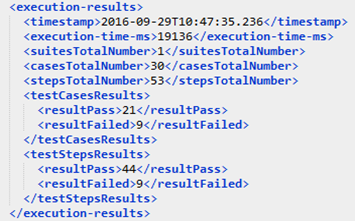
Two test cases failed due to implementation issue. In the specification an entity create or update cannot be performed using an empty context value although we got a positive result in the following two test cases.

|  |  |  |
| --- | --- | --- |
| **Test Case** | **Test step** | **Failure cause** |
| EntityCreationEmptyContextValueError\_dcc8 | CreateNewEntity23 | an entity with an empty context value should not be created |
| EntityUpdateEmptyContextValueError\_e532 | UpdateEntity38 | an entity with an empty context value should not be updated |



**IoT Broker**

The following figure provides the test suite result.



30 test cases have been generated. 9 have failed. Among the failure test cases there are 5 that not applicable in the case of the IoT broker. The rest are a specification issues.

***Tests not applicable***

The NGSI v1 model was initially done based on Orion GEi behaviour that is why we got tests that are not applicable in the case of the IoT broker.

As an example the test case “*EntityUpdateContextDoesNotExists*\_*d06a*” failed due to the test step “*UpdateEntity3*”, the wanted result is “*NotFound*” in the case of trying to update a context that does not exist although in the IoT broker the request of context update is forwarded to the context provider and the returned result is 200 OK.



***The specification issues:***

The following table summarizes the spec issues detected.

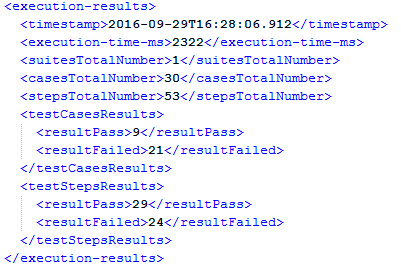
|  |  |  |
| --- | --- | --- |
| **Test Case** | **Test step** | **Failure cause** |
| UnsubscribeContextSubscriptiondoes\_not\_exists\_acdc | UnsubscribeContextSubscription9 | spec issue: the spec does not detail the error to raise in case of subscription does not exist |
| QueryContextContextDoesNotExists\_871a | QueryEntity13 | spec issue: the spec does not detail the error to raise in case of entities does not exist |
| RegisterContextJSONPayloadError\_aa75 | RegisterContext18 | spec issue: the spec does not detail the error to raise in case of Json payload error |
| DiscoverContextAvailabilityJSONPayloadError\_96f4 | DiscoverContextAvailability25 | spec issue: the spec does not detail the error to raise in case of Json payload error |

As an example the test case “*QueryContextContextDoesNotExists\_871a*”, in the specification we can’t find the error to raise in the case of a query on a non-existing context.



**Cepheus 0.1.8**

The following figure provides the test suite result.



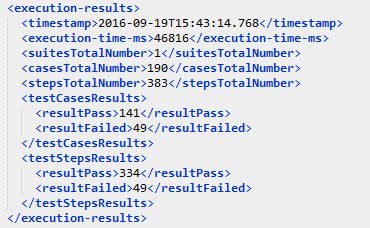
30 test cases have been generated. 9 successful tests and 21 have failed. Cepheus-Broker is a NGSI lightweight broker. Light means only a subset of NGSI operations are supported. This is mainly why we have failure test cases and the high number does not represent miss behaviour but rather not applicable test cases.

##### NGSI v2

The generated tests from the NGSI v2 model have been executed on Orion GEi version 1.2.1 and version 1.3.0.

**Orion 1.2.1**

The following figure provides the test suite result.



190 test cases have been generated. 49 have failed. The failure causes are either a specification issues or an implementation issues.

**The specification issues**

* The specification does not list the errors to be raised in each operation, it gives just a list of errors available for all the specification. As a result **3** test cases have failed.

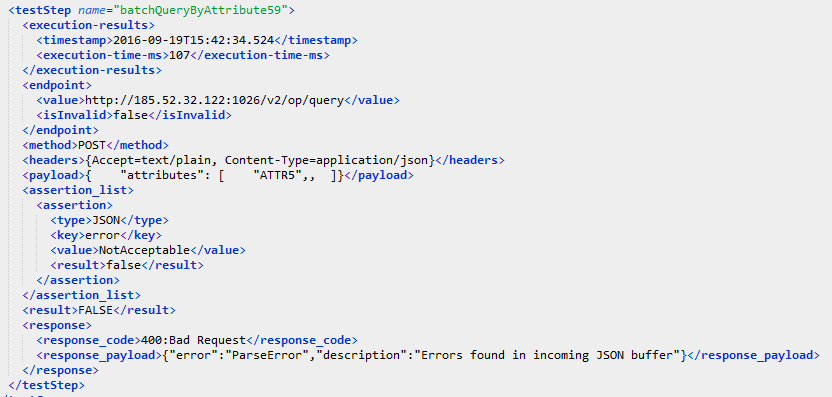
As an example, the test case “getSubscriptionByIDNotFound\_de63”, has failed due to the test step: “getSubscriptionByID110”. The wanted result is “NotFound”, this wanted result has been got from the errors list defined in the specification. Although this error is not clearly mentioned to be raised in this specific test case. It is not stated what kind of error to raise in case the subscription to be retrieved does not exist. As a result in Orion implementation a “BadRequest” error has been chosen to be raised in this case.

This is the link to the related GitHub issue:<https://github.com/telefonicaid/fiware-orion/issues/2316>



* The specification does not specify the errors priority. In case one test case contains two different error, the priority between them need to be specified so two different implementations will raise the same error for the same test case. As a result, **18** test case have failed.

As an example, the test case “batchQueryByAttributeNotAcceptableMIMEtype\_96be” has been due to the test step “batchQueryByAttribute59”. In the specification it is not stated in the case of having two errors parseError and notAcceptable, which one to raise.



This is the link to the related github issue: <https://github.com/telefonicaid/fiware-orion/issues/2342>

**The implementation issues**

Only one implementation issue has been found: the Orion GEi implementation does not respect the MIME type stated in the specification. As a result, **28** test case have failed.

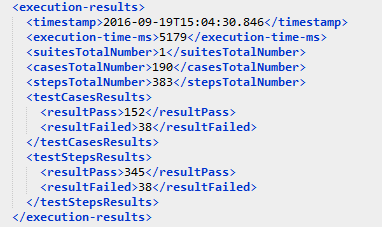
For example the test case “getAttributeDataByEntityIDNotAcceptableMIMEtype\_270f” has failed due to the test step “getAttributeDataByEntityID322”. In this test case the Accept header was set to “application/xml” which is not an accepted mime type however a positive response has been got from Orion GEi.



This is the link to the related github issue:<https://github.com/telefonicaid/fiware-orion/issues/2343>

**Orion 1.3.0**

In this version the implementation issue detected on Orion version 1.2.1 have been corrected based on the previous test report so we only have for the moment specifications issues. The figure below presents the global result.



Compared to the 1.2.1 version, the number of failed test cases decreased to **38.** The causes are the specifications issues stated above. As a result:

* **33** test cases failed due to the lack of errors priority definition in the specification in case of having more than a failure cause in a test step.
* **5** test cases failed due to the lack of errors definition for each NGSI operation.

### ***2.4.4 Continuous Integration (CI) Testing***

By integrating regularly, we can detect errors quickly, and locate them more easily.

As a trial, we successfully demonstrated the coupling of Model Based Testing to Jenkins CI. The following figure shows the previously described MBT process used as standalone (model à export executable à execute tests on SUT) and the CI process.

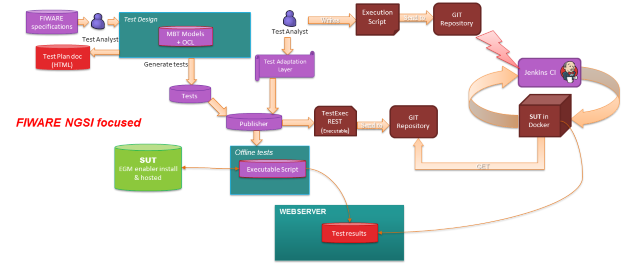


Figure 13: Continuous Integration testing architecture

The CI process is composed of two dependent tasks. First the generated tests are exported as an executable and are made available for download (in our example in a GIT repository). Then when any change is committed to the GEi git monitored by the Jenkins CI, it will launch the testing process.

In order to test the latest NGSI API on a specific GEi, Jenkins launches the Enabler implementation in a dockerized environment. Docker is a containerization platform which aims to simplify the problems of environment standardization so the deployment of applications can also be standardized. For developers, Docker allows to simulate production environments on local machines by running application components in local containers. These containers are easily automatable using Docker Compose, independently of the application and the underlying OS. Then, the test script is downloaded and run on the dockerized enabler in its own container. This method makes sure we are running every test execution on a fresh and uniform application environment. On tests completion, a detailed testing report is collected and as shown in the following figure, we can have a quick look on the overall test results of a GEi with the Jenkins build status.



Figure 14: Jenkins Integration testing example on Orion Context Broker GEi

## **2.5 Automatic online documentation testing**

This section describes the research work performed by Fraunhofer and Grassroot to provide an example of how to automate the testing of FIWARE online documentation, basically, the Catalogue. It will support the functional testing of GEs in future stages.

Every Generic Enabler in the Catalogue has to match a set of more or less formal criteria as specified by the Compliance Guide:

<https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Working_with_the_FIWARE_catalogue>

Due to the distributed nature of the FIWARE organization, Enabler owners are themselves responsible for contribution and maintenance of the online documentation. To ensure high compliance level, considerable personal resources are required to review new and updated versions of the documentation pages. Also, for new versions of the Compliance Guide complete review process is required.

To support these review procedures and save human time, we have developed two sets of metrics to make the online documentation quality measurable and comparable, in a fully automated way.

* FIWARE metrics derived from the Compliance Guide Catalogue e.g. optimal text length or mentioning of context relevant concepts;
* Common language metrics to ensure good style and legibility of online documentation e.g. average sentence length or lexical complexity.

Technical realization is done by the following software modules:

* an online text scraper to parse online catalogue;
* a text miner to analyze scraped contents and pursue measurements;
* a reporting module to inform Enabler owners on weekly base how their Enablers comply with required standards (in development).

We plan also to support a live data feed transporting metrics and measurements to the planned Quality Database, so that Enablers can attain an automatically issued Quality Label also for their contribution to the Enabler Catalogue.

The source code of this functionality will be published as the following before moving to a central repository:

* Text scraper (Scrapy/Python)

<https://github.com/sigspl/org.fiware.qa.documentation.scraper>

* Other modules (Java SE/Jersey)

<https://github.com/sigspl/org.fiware.qa.documentation.measurements>

This method is extensible to other sources of written online documentation.

During the development time, we discovered an URL pattern irregularity for Enabler pages and reported a connected bug in the Catalogue UI: <https://jira.fiware.org/browse/CAT-326>

As of August 2016, defined metric rendered best rated Enablers to get 34% of all available scores, which we consider to be equivalent to a quality label value “D”.

## **2.6 Summary of obtained results**

The quality assurance functional testing activities have been performed by a team with technical competences corresponding to the target teams who will use the FIWARE platform.

Also the human factor is an important value in the propagation of innovative platforms even if not perfect and experimentally because they are easily accepted, fitted and implemented by open mind teams.

The activities have achieved positive results and in case of insufficiencies, using the Jira tool, bugs have been reported to the owners of the generic enables to improve their products. The Jira tool has been very useful to track the problems and also to require information, in most cases a successful cooperation has been reached between the QA team and the owner of the GEs.

Few simple comments to summarize the functional test activities:

* **Documentation testing**

The GE manual allows installing the components but the documentation is not always clear, readily available from the many links.

* **APIs testing**

The installed software package implements the API declared into Open Specification.

The main failures concern the missing information on documentation.

* **Bundle Integration tests**

This activity is the most interesting, because it combines more generic enablers going to compose a platform. They are test on how they work together by simulating functional scenarios.

* **Academy courses testing**

The training is efficient if it fits the user's need and grants to acquire a quick understanding of the product. Overall the academy courses are sufficient; also in this case many tips have been reported on Jira tool to help the owner of the GEs to improve the training.

# 3 Non-functional (stress) testing

The non-functional test activity assesses the behavior of each GE in limit conditions of loading and stress.

## **3.1 Motivation and approach**

FIWARE is approaching real life and production environments in which the platform must behave in reliable and real workload conditions. This fact implies that all FIWARE GEs must work at an adequate quality, reliability and performance level for these conditions. In previous platform stages, testing at component level has been performed by GE owners, but now, both functional and stress testing need to be put in place, helping GE owners to improve the quality of their GEs. The present section states why stress (or non-functional) testing is convenient at this stage, under which assumptions this kind of testing are conducted and who are involved in such task.

Main motivation for having a task like this in the project is to evaluate the performance and stability of FIWARE GEs in similar conditions to a production and real environment by stressing the GEs up to their maximum capacity and reaching load peaks. Time responses, usage of memory, response error rate among other parameters are measured by professional testing units of FIWARE partners (Atos and ENG) who are performing the evaluation.

In order to be practical and operative, a light and practical method commonly agreed by involved partners has been followed (see Section 3.2). The section 3.3 shows all the performed stress tests and the results obtained. The tests have been reported homogeneously following a common template (as it is included as annexes of this document).

Finally, an external third party has elaborated an assessment of the testing process and obtained results to verify that tests are being done properly under standard method. The assessment has been conducted twice, one at the end of first wave of tests; and a second one at the end of all tests. The recommendations provided by the first assessment were taken into consideration for the further iterations of tests. The two reports are annexed to this deliverable.

Due to the objective itself of the task, it was planned very operative and driven by prone results. This fact implied to establish a set of principles that allowed us to perform the task in optimal conditions adapted to these external factors. Thus, following principles can be enumerated:

* The non-functional testing will first focused on performance and stability, extended to scalability when possible. Orion GE has been the most deeply tested about scalability (horizontal and vertical).
* The approach will be very practical, but some method is need to be followed defining how to test and how to produce the results, as describe in section 3.2.
* The method will be non intrusive with GE owners’ testing assurance procedures, but providing them recommendations about their GE’s behavior to improve in future GE versions
* When possible, the task will re-use some existing infrastructure for performing the tests; otherwise ad-hoc environment might be created
* The testing execution and results are updated with every release of GEs, to keep constant the performance level of GEs or even improve it ideally. In the future, the tests will be automated when possible to be launched for each new release.
* Automation is a nice to have but not a must, since it is not trivial due to the diverse nature and typology of the GEs. Each GE requires a completely different infrastructure and method of testing.
* All the tests scripts and methods are published to allow anyone to replicate the test when wished. A dedicated project in FIWARE GitHub is available for such purpose (named as test.nonfunctional).

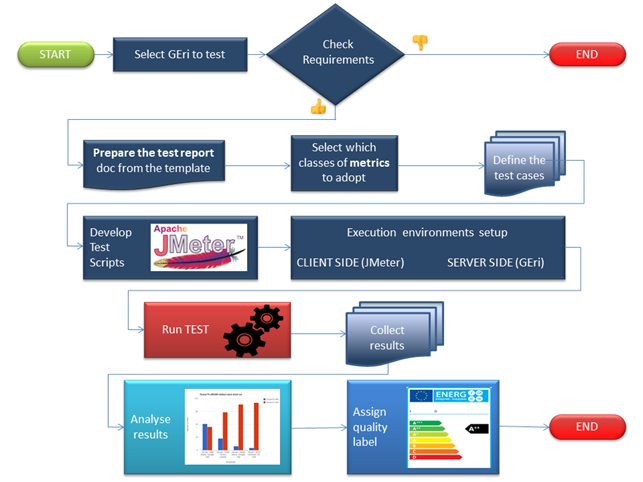
## **3.2 Methodology**

As already mentioned, a light and practical method was chosen for carrying out the stress testing. Each Generic Enabler (GE) was tested by following, as much as possible, a “black box” approach, namely soliciting its APIs and examining some non-functional parameters such as, for example, stability and performance, without any knowledge of the internal implementation.

|  |
| --- |
|  |

The purpose of this process was in fact to determine the point at which extremes of scalability or performance lead to unstable execution thus assessing the limits/breakpoints of each GE implementation.

In the following chart is represented the flow adopted in order to test each Generic Enabler reference implementation (GEri).



This flow was realized with the execution of ten steps, which are listed as following and then briefly described:

1. Select the GE implementation to be tested
2. Check requirements
3. Prepare the Test report doc from the template
4. Select which classes of metrics to adopt
5. Define the Test cases
6. Develop Test scripts
7. Run Tests
8. Collect results
9. Analyze results
10. Assign quality labels (shared with Functional testing)

***Step 1: Select the GE implementation to be tested***

The first step was to identify, among the plethora of Generic Enabler published into the catalogue at http://catalogue.fiware.org/enablers/, a subset of enablers implementations which most needed such a test. The selection of this first set of GEs was thus taken based on following criteria:

* The Cloud GEs were discarded as they rely on OpenStack, and this is a proven and very common cloud management software, so it was considered stress testing was already done by OpenStack community. Same argument was applicable for CKAN tool.
* It was given high priority to the backend GEs (data/context mng, security and IoT) as they are basic for most of the applications
* Statistics from accelerators programme were taken into account since they give figures about which ones are the most used ones. The usage level has been measured looking at:
  + if they have integrated the GE (I)
  + if that component is considered at design (D)
  + if they are only evaluating to use it (E).
  + take into consideration the sum of all three (I+D+E)

Therefore the final list of GE reference implementations selected for these tests was:

1. IDAS - Backend device management (IoT Agents)
2. AERON - IoT Broker
3. AUTHZFORCE - Authorization PDP
4. ORION - Context Broker
5. PROTON - Complex Event Processing (CEP)
6. KEYROCK - Identity Management
7. WILMA - PEP Proxy
8. KURENTO - Stream oriented SW stack
9. CEPHEUS - IoT Data Edge Consolidation
10. BOSUN - Policy Manager

***Step 2: Check requirements***

The second step of the process was to verify that the GE satisfied the following requirements:

1. The GEri should be packaged for a Docker container, whenever the “Dockerized” version is available.
2. “Stand-alone” component testing: any not necessary middle layer (SWs, networks) which could affect the test results should be removed.
3. Each test must be launched from a machine other than the one where the GE APIs are running and in the same LAN.
4. In the case of Dockerized version of GE, the Docker container must be available on the server side (GE machine).
5. JMeter must be available on the client side (Tester machine)

***Step 3: Prepare the Test report doc from the template***

A new document was created in order to report all the information needed for the test, as well as those gathered during its execution, were created. The methodology defined the template to be used as basis to collect these different types of information in a cohesive and comparable way among the different GE tested.

Once the test report document was created, a first set of basilar information was provided, simply through filling out of a set of tables defined in a dedicated template and of which an example is reported below.

|  |  |
| --- | --- |
| Attribute | Value |
| Generic Enabler | IdM Identity Management |
| Chapter | Security |
| GEri name (implementation) tested | IdM KeyRock |
| GEri version tested | 5.3.0 |
| GEri owner | UPM |
| Organisation performing the test | Engineering Ingegneria Informatica S.p.a |
| Docker file link | N/A |

***Step 4: Select which classes of metrics to adopt***

Through this step, what type of taken measure was defined simply by selecting in a table like the following the proper metric. In fact, not all the classes were used for testing every Generic Enabler, depending mainly on its specific architecture, HW topology and criticality.

|  |  |  |  |
| --- | --- | --- | --- |
| Class | Metric id | Metric | Yes/No |
| Performance and stability | MPS01 | Response times for <n> concurrent threads |  |
| MPS02 | Number of responses per second |  |
| MPS03 | Number of Bytes per second (Throughput) |  |
| MPS04 | HTTP response codes |  |
| MPS05 | Error rate |  |
| MPS06 | Maximum threads number handled |  |
| MPS07 | CPU usage |  |
| MPS08 | Memory usage |  |
| MPS09 | Network traffic |  |
| Scalability and elasticity testing | MSE01 | Scalability |  |
| MSE02 | Elasticity |  |
| Failover management and availability | MFA01 | Failover |  |
| MFA02 | Availability |  |

***Step 5: Define the Test cases***

Non-functional tests should leverage, at least partially the use cases already defined for the functional tests of the GE implementation. In this stage, the owner of the GE implementation to be tested was involved in order to find out what could be reused –once properly adapted- of the functional test previously designed, developed and executed. Each defined test case was documented through filling out a table like the following one:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | GE API method | Operation | Type | Payload | Max. Concurrent Threads |
| 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |

*GE API method*: The method being tested. E.g. from the ContextBroker GEri -> updateContext()

*Operation*: The operation being tested. E.g. from the ContextBroker GEri–> Entity creation

*Type*: the HTTP method is defined such as POST/GET

*Payload*: the message content with which the specific API method is invoked.

*Max concurrent threads:* it defines how many threads are created to simulate as many users that concurrently invoke this API method.

***Step 6: Develop Test scripts***

Starting from the test case defined in the previous step, JMeter scripts were developed in order to automate the execution of selected scenarios. These scripts were made available as part/annex of the final test report and in GitHub repository.

***Step 7: Select the GE implementation to be tested***

This step represents the execution stage of the process, where the scripts had to be launched and the servers resources usage had to be monitored (CPU, Memory, Network traffic,) on all the machines involved.

***Step 8: Collect results***

This is the step where the results (in CSV format) coming out from the JMeter scripts execution were collected.

***Step 9: Analyse results***

JMeter frontend is able to display the results as tables or graphs (with limited capability) as well as to write them to a log file (xml, CSV) that can be loaded and analysed from different tools. This methodology did not prescribe any specific charting tool for presenting data, therefore partners involved in the task could make use of their own reporting tools for providing more elaborated reports beyond JMeter basic results.

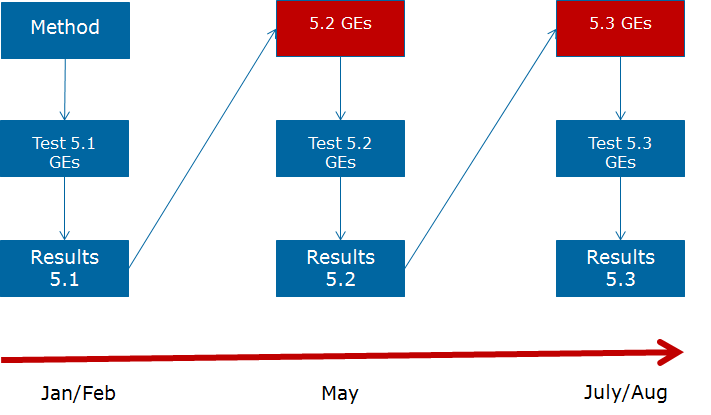
|  |
| --- |
|  |

***Step 10: Assign quality labels***

The most relevant and comparable results from each tested Generic Enabler were summarized and highlighted through a labelling system very similar to that currently adopted for years in appliances. These labels, as known, are able to inform the buyer, at a glance, about their performance in terms of energy consumption, noise level, capacity and other more specific characteristics dependently on the appliance type. The whole model that was adopted for GE labeling is described exhaustively in the dedicated section (4) of this document. This phase has been equally performed by functional working group of the task.

## **3.3 Performed tests**

Three waves of tests were done along the task. The first one was executed by beginning of 2016 considering the release 5.1 of the nine selected GEs (Orion, Wilma, KeyRock, Cepheus, Bosun, Proton, IDAS, Kurento, IoT Broker). The results were provided to the GEs who were able to consider them in some cases for the next release, 5.2, which was tested by May 2016. In this occasion, only six GEs out of the initially ten selected were tested (Orion, KeyRock, Wilma, Kurento, Cepheus and Bosun). The others did not produce significant changes in their versions to consider a new relevant testing. Once again the results were sent to the GE owners for improving next version. And finally, version 5.3 of all the GEs (the ten in total) was tested during this summer. All the reports corresponding to each wave of tests can be found in the annexes of this document. This section is summarizing the performed tests to each GE and the obtained results in the last wave of tests, that is, for version 5.3 in most of the cases.



### **3.3.1 Security – Identity Management - KeyRock**

The Identity Manager is the central component that provides a bridge between IdM systems at connectivity-level and application-level. This enabler provides authentication/access control and identity/attribute assertions as a service to relying parties. The relying parties are typically service providers that provide easy and secure access to their services to users/IoT/other services for instance by means of SSO and that rely on (personal user) attributes (e.g. preferences, location, home address, etc.)

The authentication feature of the enabler also covers authentication of things to services, objects, and users and vice-versa. End users benefit from having simplified and easy access to services (User Centric Identity Management).

Four main test cases were identified to assess the behaviour and robustness under stress conditions of a selected subset of the APIs provided by this GE. In particular **Authentication** and **Authorisation** methods, which represent the main use of KeyRock, were taken into account, to get tokens and validate them against operations that the token gives access to, such as to “list projects” and “roles”.

* In the first test case, a group of 50 threads was created in order to gradually simulate as many users requiring authentication uninterruptedly for a period of 30’. The goal of this scenario was to assess the performance of one of the most used services of the IdM that is the “**Authenticate User**”.
* In the second one, a group of 50 threads was created in order to gradually simulate as many users requiring authorisation uninterruptedly for a period of 30’. The goal of this scenario was to assess the performance of the GE **requesting roles assigned to users**.
* In the third test case, as in the previous one, a group of 50 threads was created in order to gradually simulate as many users requiring authorisation uninterruptedly for a period of 30’. The goal of this scenario was to assess the performance of the GE **requesting roles assigned to projects**.
* In the fourth and final test case a group of 10 threads was created in order to solicit the authorisation API “**check user roles**” for a period of 8 h. The goal of this scenario was to assess the stability of the system during authorisation.

The obtained results were:

* **Error rate** detected was 0%, so the system can be considered reliable.
* **Response times**:
* The measured **authentication** response timesregistered a goodaverage value (less than 1 second for more than 90% of transactions) with a number of concurrent threads up to 20, but over this threshold, response times increased significantly up to almost 4 s.
* The measured **authorisation** response timesregistered a goodaverage value (**164 ms** and **211 ms** in 2nd and 3th test scenarios).
* **Performance**:
* The GE was able to handle about **22 authentication requests/sec** which remained stable also when the number of concurrent threads increased.
* The GE was able to handle a significantly greater number of authorisation requests per second, compared to the authentication. In fact, it fulfilled on average **220** transactions/sec in the 2nd scenario and 179 transactions/sec in the 3rd one; these numbers remained stable even after having increased the number of concurrent threads.
* **GEs never crashed**: Load tests were handled by the system without registering any crash.
* **Resources usage**: In all test scenarios the CPU load was steadily over the 95%; system memory usage was stable with an average value between 2 and 2,2 GBs.

### **3.3.2 Security - PEP Proxy - Wilma**

The PEP Proxy GE Wilma is a backend component that checks authentication and authorization of users in FIWARE applications. It is designed to perform **three levels of security** for the backend REST APIs. Requests to proxy are made with a special HTTP Header: X-Auth-Token. This header contains the OAuth access token obtained from FIWARE IdM GE. In order to validate the request and forward it to the backend application, PEP Proxy check with Identity Management (IdM) and Authorization PDP GEs different parameters depending on the security level configured. This obviously implies that Wilma GE, in order to work properly, needs to be integrated with at least another GE component, the IdM KeyRock, as well as with other GEs (for Authorization) depending on the aforementioned security levels, which are:

* Level 1: Authentication PEP Proxy checks if the token included in the request corresponds to an authenticated user in FIWARE.
* Level 2: Basic Authorization PEP Proxy checks if the token included in the request corresponds to an authenticated user in FIWARE but also if the roles that the user has allow it to access the resource specified in the request. This is based in the HTTP verb and the path. (It requires integration also with AuthZForce GE)
* Level 3: Advanced Authorization PEP Proxy checks if the token included in the request corresponds to an authenticated user in FIWARE but also other advanced parameters such us the body or the headers of the request. (It requires integration also with AuthZForce GE)

Since tests carried out on this GE were focused on the first of these levels, two main test cases were identified to assess the behavior and robustness under stress conditions of its authentication API.

* The first test case aimed to evaluate the system behavior and performance; therefore a step by step approach was adopted by creating a group of 300 threads that gradually simulated as many users requiring authentication for a period of 40’. More in detail, in order to understand the system behavior with a growing input workload, as well as with a stable input workload, the test was organized in two halves. In the first test part (first 30 minutes), the number of concurrent threads (users) was variable and incremental: the test started with 1 running thread and in 30 minutes 300 running threads were reached. On the other hand, during the second half, the number of concurrent requests was steady with 300 running threads.
* The second test case aimed to verify the stability and system behavior of the system under a not too demanding input workload; therefore the test was executed for 8 hours with a fixed number of concurrent requests (originated by 80 running threads) to maintain stable and quite low this workload.

The obtained results were:

* **Error rate:** the global error rate was 0%
* **Response times**: a low average response time (219 ms in the first test and 83 ms in the second one) allowed the GEri to guarantee, on average, up to 839 requests/s.
* **GE never crashed** during each test case.
* **Resources usage**: in all test scenarios CPU usage was stable under the 55%; in the long period (stability test) the system active memory usage was stable with an average value of around 2 GB.

### **3.3.3 Security – Authorization PDP - AuthZForce**

The PDP AuthZForce GE reference implementation is a component that extends security features implemented on an IDM GE evaluating which operation a user can perform on specific domain resources. The GE provides two main features:

* **Authorization policy decision evaluation**: This is the main feature of this GE as a PDP. Indeed, PDP stands for Policy Decision Point and its main feature consists to evaluate authorization decisions based on XACML policies and attributes related to a given access request (e.g. requester’s identity, requested resource, requested action), following the policy evaluation logic defined in the XACML standard. This feature is provided to external clients through a REST API that we call the PDP API, where PDP is short for the term Policy Decision Point defined by the XACML standard.
* **Authorization policy administration**: creation, retrieval, update and removal of XACML policies. This feature is provided to external clients through a REST API that we call the PAP API, where PAP is short for the term Policy Administration Point defined by the XACML standard. This feature is necessary to support the previous feature. Indeed, it allows policy administrators (such as application developers potentially) to configure the XACML policies to be evaluated by the GE when calling the PDP API.

Two test cases were identified to assess the performance and stability of this implementation, both based on the first of the two aforementioned features, namely the Authorization policy decision evaluation.

* The first test case aimed to evaluate the system behavior and performance; therefore a step by step approach was adopted by creating a group of 150 threads that gradually simulated as many users requiring authentication for a period of 30’. More in detail, in order to understand the system behavior with a growing input workload, as well as with a stable input workload, the test was organized in two halves. In the first test part (first 15’), the number of concurrent threads (users) was variable and incremental: the test started with 1 running thread and in 15’ it reached 300 running threads. On the other hand, during the second half, the number of concurrent requests was steady with 150 running threads.
* The second test case aimed to verify the stability and system behavior of the system with a workload size near to its most production point (4627 requests/s); therefore the test was executed for 8 hours with a fixed number of concurrent requests (4130 requests/s on average originated by 50 running threads) to maintain stable and quite low this workload.

The obtained results were:

* **Error rate:** the global error rate was 0%
* **Response times** and load results were very good in both tests; in fact they maintained an average value around 11 ms with a load average of around 4376 requests per second.
* **GE never crashed**. Load and duration tests were well handled by the system without any trouble or crash.

### **3.3.4 IoT - IoTBroker - Aeron**

The GE IoT Broker implementation Aeron is a backend component that offers the following three main features:

1. It offers a single point of contact to the user, hiding the complexity of the multi-provider nature of the Internet of Things.
2. It collects and aggregates information about thousands of real-world objects on behalf of the user.
3. It provides means to assemble lower-level device information (device-centric access).

Moreover, it has unique properties that cannot be found currently in other IoT platforms such as, on the one hand: information is generated and exchanged only when needed (this is in contrast to the state-of-the-art middleware components where any piece of information - whether needed or not - is stored inside a central repository); on the other hand IoT applications can be written without consideration of the device installation.

1. The scenarios defined for stress testing were taken, as for the first wave of tests, from the most used operations of IoT Broker, namely the following API: *updateContext*, *queryContext* and *subscriptionContext*.   
   The first scenario aimed to stress the NGSI-10 updateContext operation that represents the push-mode interaction offered by this GE, namely a scenario where no previously subscriptions are needed by the related data flow; in this test, 200 users were simulated by as many JMeter threads which solicited the related API with an overall amount of 70.978 requests for a period of 40’. The main objective was to monitor the behavior and performance of this operation under a growing workload in order to find the most production point.
2. The second test scenario aimed to assess stability of the NGSI-10 queryContext operation that represents the interaction offered by this GE designed for the synchronous retrieval of data. This type of scenario required data to be queried with which the system had to be previously filled in; secondly a very light workload of five users was simulated through as many JMeter threads that solicited with 27.844 requests the queryContext API for a period of 5 hours in a row, evaluating in this way, the system stability.
3. Finally, the third test scenario, very similar to the first one, aimed to stress the NGSI-10 subscribeContext, an operation that represents the asynchronous data retrieval interaction offered by this GE. For this stress test 200 users were simulated by as many JMeter threads which solicited with 40.619 requests this API for a period of 40’. The main objective of this test case was, in the same way as for the first one, to monitor the behavior and performance of this operation under a growing workload in order to find the most production point.

The obtained results were:

* **Error rate** was 0% for the updateContext and queryContext scenarios while, SubscribeContext had an error rate of 44,58% which would need further investigation at development side (even because the same type of error was registered in previous test wave of tests but with far fewer occurrences 2,75%).
* **Response times:** the UpdateContext API showed to be able to fulfill at most about 30 requests/s generated by 16 concurrent threads with an average response time (RT) of 468 ms; after this threshold, RT increased dramatically till to 4 s on average.
* **GE never crashed** during each test.
* **Resources usage** in the first two scenarios RAM usage was always above 1,6 GB while CPU > 70% with peaks of 99,5%. In the third scenario (that one with the lowest stressful conditions) RAM usage was 2,28 GB with peaks of 3,32 GB and release memory problems while CPU was around 68% with peak of 94,5%.

### **3.3.4 Data- Context Broker - Orion**

The Orion Context Broker is an implementation of the Publish/Subscribe Context Broker GE, providing the NGSI9 and NGSI10 interfaces. Using these interfaces, clients can do several operations:

1. Register context producer applications, e.g. a temperature sensor within a room

2. Update context information, e.g. send updates of temperature

3. Being notified (subscription) when changes on context information take place (e.g. the temperature has changed) or with a given frequency (e.g. get the temperature each minute)

4. Query context information. The Orion Context Broker stores context information updated from applications, so queries are resolved based on that information.

For the performance test, two identical physical machines have been used. The first one was used to deploy Orion Context Broker and the MongoDB database. The other one was used to execute the Jmeter tests scripts, injecting the load.

The requirements of the used infrastructure have been:

· CPU: 1 CPU Intel(R) Xeon(R) CPU E31230 @ 3.20GHz with 4 cores and 8 threads

· RAM: 4GB DIMM DDR3 Synchronous 1333 MHz

· Hard Disk: 128GB

· Operating System: CentOS release 6.7 - 64 bits

· NICs: 2 interfaces 82574L Gigabit Network Connection

The performance tests executed are composed by nine scenarios:

1. Update Stress Scenario (1-20 attributes)

2. Update Stress Scenario (1-6 attributes)  
3. Convenience Update Stress Scenario  
4. NGSIv2 Update Stress Scenario  
5. Update Stress with notifications  
6. NGSIv2 Update Stress with notifications  
7. Stability Scenario  
8. Optimized Stability Scenario  
9. No-cache Optimized Stability Scenario

Scenarios 1, 3 and 6 are the same than in the first Performance Testing launched over Orion.

Scenarios 4 and 5 are updated scenarios over the NGSIv2 API instead NGSIv1. Scenario 2 is the same than scenario 1, but the number of attributes is lower, in order to compare the results with scenario 2 (NGSIv1 vs NGSIv2).

Scenarios 4 and 5 are the same than scenarios 1 and 3, but a previous load of 1000 subscriptions has been launched over the database, in order to makes Orion to generate notifications.

Scenario 7 is the same than scenario 6, with the optimizations recommended by Orion’s developer:  
1. Added the next parameters to Orion startup:  
-reqMutexPolicy none -writeConcern 0 -logLevel ERROR -notificationMode threadpool:q:n  
2. Created four indexes over the entities collection in the database:  
db.entities.createIndex( { "\_id.id": 1 } )  
db.entities.createIndex( { "\_id.type": 1 } )  
db.entities.createIndex( { "\_id.servicePath": 1 } )  
db.entities.createIndex( { "attrNames": 1 } )

Finally scenario 8 is the same than scenario 7, previously disabling the cache.

The obtained results were:

* **Error rate** was 0% for all the scenarios.
* **Response times:** 1 second response time and 190 responses per seconds for update 1-20 attributes. 250 milliseconds RT and 750 responses per second for update 1-6 attributes. 1 second RT and 181 responses per second for convenience update 1-20 attributes. 600 milliseconds and 310 responses per second for NGSIv2 update 1-6 attributes. 423 ms RT and 450 responses per second for update 1-6 attributes with notifications. 754 ms RT and 249 responses per second for NGSIv2 update 1-6 attributes. 2,1 seconds and 14 responses per second for stability scenario. Around 2 seconds RT and 23,5 responses per second for optimized stability scenario. More than 40 seconds RT and 12 responses per second for no-Cache optimizes stability scenario.
* **GE crashes** in optimized stability scenario with cache enabled, due to memory run out.
* **Resources usage** the RAM usage is between 200 and 400 MB, excepting Stability with cache enabled, when the system uses the whole memory. The CPU usage is around 15-16% for most of the cases. The maximum is 65% in Stability with cache disabled.

In summary, the performance has improved a lot. The performance is better in NGSIv1 than in NGSIv2, but the bandwidth usage is much more optimized in NGSIv2. The memory problem is avoidable disabling the cache.

### **3.3.5 IoT - Data Edge Consolidation - Cepheus**

The IoT Data Edge Consolidation (IDEC) GE addresses the need to process data in real time. Frequently implemented features include filtering, aggregating and merging real-time data from different sources. Complex Event Processing (CEP) allows 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.

Cepheus is divided in two deployable applications. The first one is a light NGSI broker which features an implementation of the standardized NGSI API. And a CEP which features a dedicated REST management API to configure real time analysis on NGSI events.

For deploying Cepheus, it has been used a Raspberry Pi model 2 with these features:

* CPU: 1 CPU Broadcom BCM2836 ARM Cortex-A7@ 900 MHz with 4 cores and 8 threads
* RAM: 1GB
* Hard Disk: 32 GB micro SD card
* Operating System: Raspbian 7 armv71
* NICs: 1 interface 10/100MBps

For executing the performance Jmeter test scripts, it has been used three Windows 7 machines injecting load.

The performance tests executed are composed by four scenarios:

1. CEP Stress scenario.

2. Broker Stress scenario.

3. Load scenario.

4. Stability scenario.

For all the scenarios, it has been used the same script, updating the data from a device. The first one consists in a stress scenario sending the requests directly to CEP. The second scenario is the same that the first scenario, but the requests are sent to Cepheus Broker instead of CEP.

The third scenario is the same that the second one, with half load, and longer duration, trying to match with the performance requirements provided by the GE’s owner.

Finally, for the stability scenario, we used again the same script, sending the requests to Cepheus Broker, with a moderate load, and a high duration.

The obtained results were:

* **Error rate** was 42’16% for Broker stress scenario, and 0% for all the rest.
* **Response times:** about 67 milliseconds in maximum production point, reaching 200 milliseconds at maximum in CEP stress scenario, and handles 140 responses per second, 60 responses per second for Broker stress scenario reaching 100 seconds of response time (the system falls), 100 responses per second and 23 milliseconds response time for load scenario, and finally, for stability scenario, response times similar to load scenario ones, and the system can handle 45 request per second.
* **GE crashed** in Broker stress scenario, running out of memory after about half an hour.
* **Resources usage** the RAM usage is full due to a memory leak. The CPU usage is 40% for CEP stress scenario, 36% for Broker stress scenario, 55% for load scenario and 27.5% for Stability scenario.

### **3.3.6 Cloud - Policy Manager - Bosun**

The Policy Manager GE provides the basic management of cloud resources based on rules, as well as management of the corresponding resources within the FIWARE Cloud Instance like actions based on physical monitoring or infrastructure, security monitoring of resources and services or whatever that could be defined by facts, actions and rules. Policy Manager is a rule engine designed to be used in the OpenStack ecosystem and inside the FIWARE Cloud.

Bosun is composed by two components: Facts for event receiving and Cloto for the creation of rules definitions and subscription to rules management.

For the performance tests, two identical physical machines have been used. The first one was used to install Bosun and all his requirements (Redis, MySQL, RabbitMQ, Orion Context Broker, etc.).

The other one was used to execute the Jmeter tests scripts, injecting the load. Furthermore, an Oracle Linux Virtual Machine with Openstack was deployed. That was another requirement in order to use Bosun.

The features of the machines are:

* CPU: 1 CPU Intel(R) Xeon(R) CPU E31230 @ 3.20GHz with 4 cores and 8 threads
* RAM: 4GB DIMM DDR3 Synchronous 1333 MHz
* Hard Disk: 128GB
* Operating System: CentOS release 6.7 - 64 bits
* NICs: 2 interfaces 82574L Gigabit Network Connection

The performance tests executed are composed by two scenarios:

1. Stress scenario – High Security, Debug log.  
2. Stress scenario – Low Security, Info log.  
3. Stability scenario.  
4. Facts stress scenario.  
5. Facts stability scenario.

For the first three, it has been used the same test case. With one server and one tenant, a rule creation request is sent. After a one second pause, a subscription request to that rule is sent. Finally, after another one second pause, an unsubscription request to that subscription is sent.

For the last two, the test case has been a fact sending to the Facts component.

For the five scenarios, a think time of one second has been placed between each request for the same thread (like in the performance testing for version 2.3.0).

The first one consists in a stress scenario with a high load in a short period of time, configuring the “Security” parameter to “High” (token checking in each request), and the log file in debug mode.

The second scenario is the same than the first one, but this time the “Security” parameter is configured to “Low”, and the log file to info mode.

The third one is the same stability scenario than for the version 2.3.0.

The fourth scenario is a stress scenario for the facts component, similar to the stress scenarios for Cloto.

Finally, the fifth scenario is a stability scenario for the facts component, similar to the stability scenario for Cloto.

The obtained results were:

* **Error rate** was about 3.5% for all the stress scenarios. No errors in the rest of scenarios
* **Response times:** about 300 milliseconds for Cloto when the maximum production point is reached. When the errors start to appear, response times dramatically grow (six seconds average time in stress test). Cloto component can handle about 26 responses per second. The behaviour is the same for facts component, but can handle about 30.5 responses per second, and the response times are under 100 milliseconds when the maximum production point is reached.
* **GE crashes** after about 3 hours in stability test.
* **Resources usage** the RAM usage grows in time and, with enough running time, it could take the whole memory. The CPU usage is about 7% in all cases.

In summary, Bosun can handle about 26 HTTP responses per second for Cloto (much better than the past version), and 30.5 for Facts. The memory and CPU usage are stable (just a little grow for facts version). Cloto still falling, as in the past version, but it seems to fall after some hours even without using it.

### **3.3.7 Data - Stream Oriented - Kurento**

The Stream Oriented GE is a development framework that provides an abstraction layer for multimedia capabilities, allowing non-expert developers to include interactive media components to their applications.

For deploying Kurento, it has been used a physical machine with these features:

* CPU: 4 Intel(R) Xeon(R) CPU @ 2.00GHz with 2 cores and 8 threads (64 total virtual CPU’s)
* RAM: 128GB DIMM DDR3 Synchronous 1333 MHz
* Hard Disk: 1TB
* Operating System: Ubuntu 14.04.3 LTS - 64 bits
* NICs: 4 interfaces BCM5720 Gigabit Ethernet Network Connection

For executing the performance test scripts, it has been used an Openstack Environment, with 9 Windows 2012 Server virtual machines. Each one had 4 GB of RAM and 2 Virtual CPU’s. All of them executed Selenium scripts in order to inject the load to Kurento.

The performance tests executed are composed by two scenarios:

1. Loopback Stress scenario.

2. Loopback Stability scenario.

The first one is a stress scenario with a Loopback operation: the client sends via browser (Google Chrome) his webcam signal. Then Kurento receives the data, and send it back to the client. For this scenario, it has been used up to 90 simultaneous virtual users.

For the stability scenario, it has been used the same test case, but with a moderate load and a very long time duration (more than 4 hours).

The obtained results were:

* **Error rate**: two peaks of 2.5 fraction lost in Stress scenario. 0 in Stability scenario.
* **Response times:** about 250 milliseconds RTT for Stress scenario, and about 50 for Stability scenario. 40 Kbps for Stress scenario (90 users) and 70 for Stability scenario (20 users)
* **GE never crashed** during each test.
* **Resources usage** the RAM usage is about 1GB and 10% CPU usage for Stress scenario. About 500 MB and 5% CPU usage for Stability scenario.

We then conclude that the Generic Enabler Kurento has a good behavior for less than 50 users, and low quality videos.

### **3.3.6 IoT – Backend Device Management – IDAS**

The IDAS component is an implementation of the Backend Device Management GE.

Its main usage is to translate the Device or Gateway specific communication protocol into NGSI. It will always handle sensor notifications from the device towards the ContextBroker and for some specific protocols actuation/command messages from the ContextBroker to the device.

For the performance test, two identical physical machines have been used. The first one was used to deploy IDAS and all his requirements (Orion Context Broker with its MongoDB database). The other one was used to execute the Jmeter tests scripts, injecting the load.

The features of the machines are:

* CPU: 1 CPU Intel(R) Xeon(R) CPU E31230 @ 3.20GHz with 4 cores and 8 threads
* RAM: 4GB DIMM DDR3 Synchronous 1333 MHz
* Hard Disk: 128GB
* Operating System: CentOS release 6.7 - 64 bits
* NICs: 2 interfaces 82574L Gigabit Network Connection

The performance tests executed were composed by two scenarios:

1. Stress scenario.

2. Stability scenario.

The first one is a stress scenario of an attribute update. The second scenario is composed by the same operation, but with lower load than in the first one, and with very higher test duration.

The obtained results were:

* **Error rate**: 0%.
* **Response times:** In stress scenario, with 200 simultaneous threads, the response times are below 100 milliseconds at his maximum production point, and in stability, with 100 threads, the times are between 0.7 and 1.35 seconds. The system can handle about 140 device attribute updates per second.
* **GE never crashed** during each test.
* **Resources usage** the RAM usage is about 3’5GB and 14% CPU usage for both scenarios.

Thus, we conclude that Generic enabler IDAS IoT Agent, has high reliability and performance. The memory leak from past versions has gone

### **3.3.6 Data - Complex Event Processing – Proton**

The CEP GE analyses event data in real-time, generates immediate insight and enables instant response to changing conditions. While standard reactive applications are based on reactions to single events, the CEP GE reacts to situations rather than to single events. A situation is a condition that is based on a series of events that have occurred within a dynamic time window called processing context. Situations include composite events (e.g., sequence), counting operators on events (e.g., aggregation) and absence operators.

The Proactive Technology Online is an implementation of the FIWARE CEP (Complex Event Processing) GE. The Proactive Technology Online is a scalable integrated platform to support the development, deployment, and maintenance of event-driven applications.

The Proactive Technology Online authoring tool allows the definition of CEP applications using a web user interface.

The Proactive Technology Online engine is a runtime tool that receives information on the occurrence of events from event producers, detects situations, and reports the detected situations to external consumers.

For the performance tests, two identical physical machines have been used. The first one was used to install Tomcat with all the Proton’s war files deployed in it. The other one was used to execute the Jmeter tests scripts, injecting the load.

The features of the machines are:

* CPU: 1 CPU Intel(R) Xeon(R) CPU E31230 @ 3.20GHz with 4 cores and 8 threads
* RAM: 4GB DIMM DDR3 Synchronous 1333 MHz
* Hard Disk: 128GB
* Operating System: CentOS release 6.7 - 64 bits
* NICs: 2 interfaces 82574L Gigabit Network Connection

The performance tests executed consists of three scenarios:

1. Event Stress scenario.

2. Definition Stress scenario.

3. Stability scenario.

The first two consist of separate stress scenarios for each of the selected services of PROTON: Events and Definitions. The Stability is a combined (30 threads of Events and 3 threads of Definitions) long-running scenario.

The obtained results were:

* **Error rate**: 0.39% in Events stress scenario. Almost 0% for all the rest.
* **Response times** below 100 milliseconds and 500 responses/second for Events stress scenario (before the system gets overloaded. 9 milliseconds and 950 responses/second for definitions stress scenario. About 850 milliseconds in average and about 30 responses per second for stability scenario.
* **GE crashes in stability scenario** due to a memory leak (the threads opened are never closed).
* **Resources usage:** In every scenarios, the system runs out of memory. The CPU usages are 40% for event stress before the system runs out of memory and 80% after. 6% for definitions stress scenario. 90% at the end of Stability scenario.

We can conclude that PROTON, has excellent reliability and high performance, but its stability is adversely affected by the RAM management, especially in its service Events, so it is recommended to fix this issue to reach optimal stability.

## **3.5 External assessment**

In order to ensure the quality itself of the carried out methods and tests in non-functional testing to assess the GEs quality, an external expert was in charge of assessing them to state the validity of the performed work.

The expert who took over this assessment was Holger Schlingloff, the Chief Scientist of the System Quality Center (SQC) at Fraunhofer, FOKUS. His profile can be seen at <https://www.fokus.fraunhofer.de/usr/en_schlingloff>

The assessment was done twice, one in February 2016, after the testing of 5.1 release; and the second one at the end of task in September 2016.

The main statement of the first report was “*In general, the process for non-functional testing is adequate and the preliminary results obtained by the non-functional tests are satisfactory*”. However, some recommendations were provided to be considered:

* *The baseline (minimal expected load) and targets (maximal load reasonably to be expected) for the non-functional testing should be defined and justified before the actual testing takes place*.

These values were requested to GE owners and most of them claimed that they were not aware of such values. In those cases where the GE owner provided the reference values, they were used for establishing the limit of the GE. When values were not available, reference values were obtained from the accelerators usage of GEs.

* *Potential bottlenecks which could occur in the application of (combinations of) GEris should be better analysed*.

Initially the task wanted to isolate the GEs in order to know the properties and values of each GE, but combination of GEs has been considered and two bundles have been tested: the one formed by Orion, IDAS and Cygnus; and the one composed of Wilma, KeyRock and AuthZForce.

* *Testing results should be better interpreted, and alternative routes and variants which may occur as a result of non-functional testing should be explicated*.

An effort to homogenise the tests results has been done. The two partners involved in the task have worked on common templates for reports and agreed on common criteria and measures to align the testing process across all GEs. It has been also making an effort in explaining the rationale behind the obtained results, especially to allow GE owners to understand the reason of got values and how they could improve them.

* *In mid-term, non-functional testing should be linked to other software-development activities*.

This is a nice to have feature that in the context of present project has been started although not completed. It is planned to enhance and fulfil this objective in near future. Anyway, it is not clear how much automation and integration of stress testing with software development process will be able to be reached. The main concern is the variety of testing scenarios and environments are required for each GE, as they are from diverse type of application and way of usage.

Full first report can be consulted at [Non-Functional Assessment Report](https://forge.fiware.org/docman/view.php/7/5516/20160224_Fraunhofer_FIWARE-Testing-Assessment_v2.pdf).

The first report results were presented in Vienna at the FIWARE Foundation on 1st June 2016.

The second report was requested to the expert by mid September, but at the time to write this report was not yet received.

# 4 Labelling model and application

Once the GEs are tested, a way to make evident the result of the test was thought as needed. A model of labels to state the quality of each GE by using a graphical visualization was decided. This section explains the selected approach, the method used for calculating the labels and the application of the model to ten selected GEs.

## **4.1 Motivation and approach**

*You cannot control what you cannot measure*” is a well-known old motto by Tom Demarco [3]. But you cannot measure what you are not able to properly define and you cannot define what you do not properly know. Thus, the first step is properly to know and define the entities of interests (EoI) for our analysis. Second, measure and establish a measurement protocol for reducing and minimizing the probability of a bad measurement. Third, to store historical data for determining trends, useful to refine forecasts and estimations and, in our case, better determine the effort needed to use the ‘building blocks’ (GE – Generic Enablers) from this project. Each GE needs to have a ‘value rating’ expressing its goodness, hybridizing quantitative (*measurement*) and qualitative (*evaluation*) viewpoints into a unique, consolidated view: this involves also non-technical stakeholders by proper (visual) communication mechanisms, possibly with analogies from the common daily life: the simpler, the better. An example is therefore to use the ‘labeling’ concept, as well as done for the consumption of energy. A rating is not a measurement, but the result of an evaluation. In this case better to use an even scale in order to avoid choosing the central value in such distribution. ‘Adopt and adapting’ such concept can reduce in the target users their possible psychological resistance to change and easily being accepted and understood. That’s why it has been set up the seven (7) values (from A+++ to D), as well as in the energy ratings for home appliances also for rating GE’s.

Figure : EU labelling system

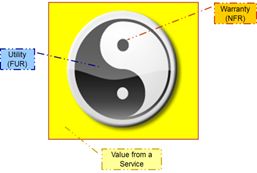
Such label will express the ‘value’ of such component, where such term – as well as in the Service Management domain (e.g. in ITIL, ISO 20000, etc.) – means the logical summation of ‘*utility*’ (what a ‘service’ should do, according to the customer viewpoint) and ‘*warranty*’ (how and how much a ‘service’ should operate, according to the users’ viewpoint). Only putting together at least such two views, a provider can bring back a ‘quality’ product/service. In the system/software engineering domain, ‘utility’ matches with *FURs* (Functional User Requirements) and ‘warranty’ with *NFRs* (Non-Functional Requirements). Referring to the standard ISO glossary[[3]](#footnote-3), a product is the ‘component’ for delivering a service and – from a provider perspective, as the FIWARE consortium is – the product ‘is’ the ‘service’ for the European Commission. Here a visual example of such complementarity between the two aspects:

Figure : A (service) value representation

A well-known and logical best practice is to ‘adopt and adapt’ a well-known concept from other (physical) domains and (personal) experiences reducing the possible resistance to change and be easily accept by the target users. Thus, looking to the FIWARE GE Catalogue (<http://catalogue.fiware.org/enablers>), here it follows a preview about how it shall appear.

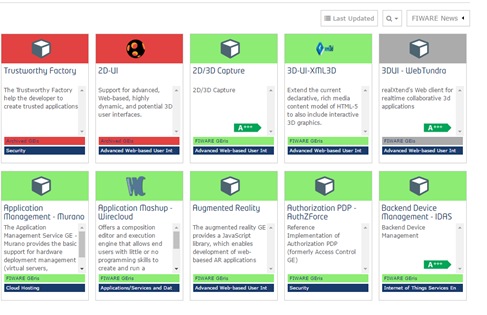
.

Figure 17: A preview of GE quality labelling in FIWARE Catalogue

Thus, the hybrid measurement (‘labeling’) for GEs matching functional and non-functional attributes will follow the consolidated assessment criteria in process management in Maturity & Capability Models (MCM), that’s an ordinal 4-scale rating (**N/P/L/F**; Not/Partially/Largely/Fully achieved). You cannot achieve the next level if the previous level is not (at least) a L/F evaluation [3]. Using a grid-like approach and a set of criteria for determining the overall quality of a GE, considering both functionalities (*utility*) and non-functionalities (*warranty*), we can determine a three-dimensional ‘(GE) labeling cube’, where each GE shall be described by a set of attributes (as described in the following sections). The criteria for evaluating a product shall arise from the ISO/IEC 25010:2011 quality model applicable both to products and services and for each criteria (attribute) one or more measures shall be applied, obtaining a hierarchy with three levels.

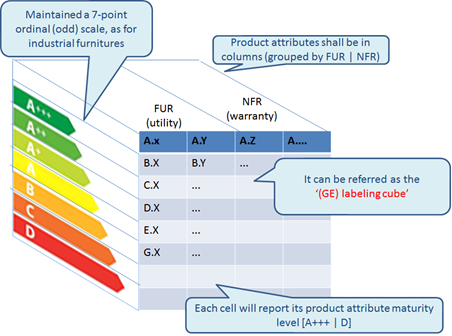


Figure 18: The GE labelling cube

In this way each GE shall have few possible product attributes describing it, using a 7-point ordinal scale as the final rating (labeling), from ‘A+++’ to ‘D’. ‘Adopting and adapting’ such approach to rate process (aggregating NPLF values for determining Capability Levels) to GE’s (products), each defined measure (described in the next sections) will establish thresholds for determining the quality levels and the final overall GE value for determining its label will be average level achieved by ALL the evaluation criteria adopted, as in the following figure.

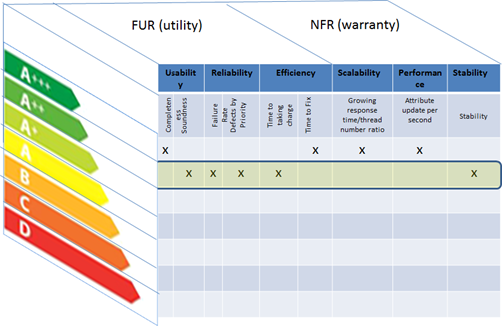


Figure 19: An example of rating (detailed) results

The overall rating for such GE would be a Level ‘A++’; (2) the detailed result from the GE Labeling cube will be visualized clicking on the label value from the FIWARE GE catalogue, allowing interested stakeholders to that specific GE to properly understand where such component could be reworked and/or eventually improved first. That’s what it is needed to provide them.

See Section 4.6 for the results obtained for the set of GE's.

## **4.2 Labelling model representation**

Once established the labels (detailed and overall) for all GEs, the model must be represented and visualized somehow the obtained results can be shown to the GE’s users in the Catalogue and automatically updated as soon as new measurements are taking place. Thus, to represent the metrics models and measurement process, it is proposed the following data model:

* a “Metric” data object to specify the metric/attribute pair and the role of the metric in the software delivery process
* a “Scales” data object to decode label value from measured value
* a “Measurement transaction” data object to register and store events of measurement procedures and results taken on components.

The following table specifies proposed design of the data objects. Details are omitted since the work has not been completed at the time of this report.

|  |  |  |
| --- | --- | --- |
| **metric** | **lookup** | **measurement transaction** |
| id  unique\_name  domain  phase  segment  entity  attribute  measure  purpose  revision  date  formula  unit1  unit2  unit3  unit4  quantitative  meas\_scale  note  description  changelog | measure\_id  range\_start  range\_end  label  revision\_date  unit  meas\_scale  framework1  framework2 | id  timestamp  vendor  component  subcomponent  surveyor  metric  value  protocol |

Further development steps require implementation of a service to allow exchange this data:

* flow of measurements from testers to the database;
* data queries from the Catalogue.

A service prototype project is available on GitHub using Grizzly/Jersey, but not completed yet at the time to write this report: <https://github.com/sigspl/org.fiware.qa.documentation.measurements> (status: alpha).

By proposed design, labels should not be static images, but an output of a labeling service. In the backend, measurement data as described in following sections is aggregated and by applying a set of business rules the service returns a label bound to an Enabler. This service can be contextually called from the Catalogue. As currently the Catalogue is a Drupal CMS instance, a Drupal UI integration is required.

How label images would visually integrate with the Catalogue, is shown in the Figure 16 above. Additionally, detailed data will be displayed in a popup or separate page similar to the EC Energy Label detail view.

To have a status overview of labels across different Enablers, we propose additional dashboard as a standalone app independent of the Catalogue. The dashboard can be seen in the screenshot below. Labels are generated per API request using PHP/GD platform. The code is released on GitHub: <https://github.com/sigspl/labelservice.prototype.php> (Note: current version implements labels A-F as shown in the screenshot; according to the new requirement the labels are A+++ to D, this needs to be updated in the software).

While the automation of labels visualization is completed by the assigned team (Fraunhofer/Grassroot), Atos has worked on a static visualization of the obtained labels so far in the Catalogue. A set of icons have been generated from shield.io web site for representing each of labels value:

Figure 20: Labels icons

Then, the corresponding overall label has been embedded into corresponding GE box in the Catalogue, as for example can be seen below for Orion. By clicking on the label icon, a pop up window shows the detailed labels along the obtained values and meaning of each evaluated criteria.

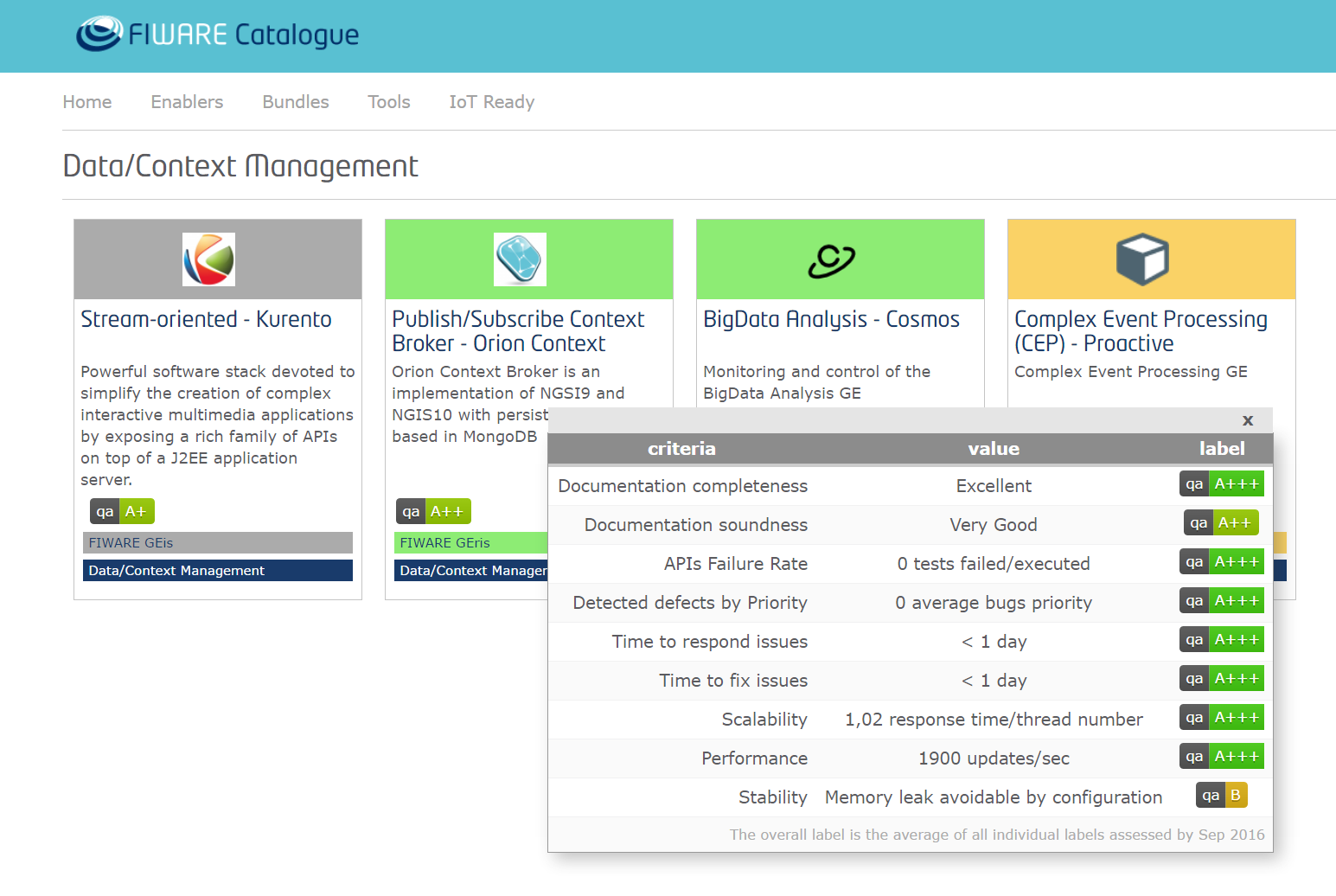


Figure 21: Visualization of Orion labels in FIWARE Catalogue

## **4.3 Governance model**

From usual practices in certification schemes, a governance model has been proposed. In particular, the ISO 17000 standards series specifies general terms and definitions relating to conformity assessment, including the accreditation of conformity assessment bodies, and to the use of conformity assessment to facilitate trade. Some concerns have been raised by many certification specialists when entities like a forum manage several functions inside its organization. This remark is regularly made in many sectors and easily solved with a set of pragmatic solutions. The guidelines for such issues are the ISO 17020 and ISO 17025 which set the basic principles for organizing the certification and includes requirement concerning independence of the players.

Different roles are identified within labelling programs. While some of the roles may need to be hold by different instances (to separate “execution and control) of the FIWARE governing structure, others could be cumulated. The table below provides a synthesis of the roles which have been proposed to be set-up, suggesting within the FIWARE governance model under development.

|  |  |  |
| --- | --- | --- |
| **Role** | **Description** | **Suggested Implementation in FIWARE** |
| Authority | Top-most decision authority of the labelling program. It defines the main orientation and objectives of the program. | Board of Directors (BoD) |
| Specifier | Specifies the label. This includes the identification of dimensions to be evaluated, the definition of the scoring system and the methods to be used to run the evaluation in a transparent and reproducible way. | *Body to be created - Currently done by the Quality Assurance task* |
| labelling body | The labelling body examines the test reports and provides feedback and final decision to issue the label to the applicants. This is a sensitive function which needs to be trusted so to create overall confidence in the label. It maintains the overall evaluation procedures. | *Body to be created* |
| Test lab | The Test lab realises the evaluation, based on the process defined by the Specifier. The Test lab can be within the Labelling body or external (3rd party lab) Self-testing approached can also be proposed as long as a validated test system, satisfying trust and accuracy of results is provided. This is called self-labelling. The Test Lab has to provide authorized means of testing. | *Body to be created - Currently done by the Quality Assurance task* |
| Accreditor | This role is optional. Accreditor approves the Test laboratory allowed to conduct tests. This role is mainly of interest when tests are to be realised by 3rd party labs. | Not required in first step |
| Technical Advisor | Gives advice on (technical) difficulties or experience related to the test activities. | FIWARE Technical Steering Committee |
| Applicant | Applies to the label by providing the software/hardware artefact to be tested. | GE owners |

Table 7: Proposed roles in labelling model governance

The overall process to set and manage the label has been proposed as follow:

1. The Authority defines the main orientation and objectives of the program and appoints a body, the Specifier, in charge of defining the label program
2. The Specifier defines the label details in an operational way so the label can be deployed without misinterpretation and appoints the Certifier who will be in charge of running the certification process. It also identifies obligations and accreditation process for Testers. Several certifiers could be appointed.
3. The Certifier accredits the Test lab, who is allowed to run the evaluation. Several Testers may be accredited. The accreditation process has to ensure that evaluations are made in a managed and reproducible way, within a Tester facility and among the selected Testers. A Round robin evaluation could be run do to demonstrate such reproducibility.
4. The Applicant registers its candidate GE for labelling. This can be a new GE or major release of an existing GE.
5. The Applicant gets its evaluation made, either itself (self-labelling) or by the Test lab, using authorized means of testing.
6. Results are provided to the labelling body as a test report
7. The Labelling body evaluates the test report and provides the label to the GE.

This is represented in the figure hereunder.

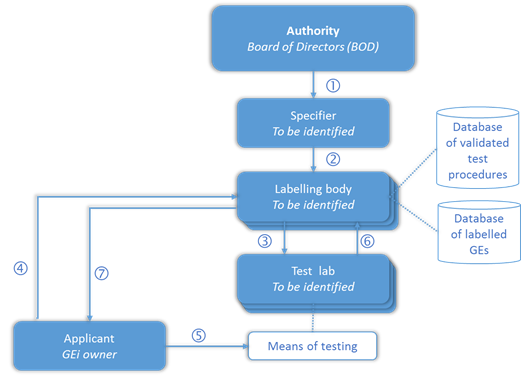


Figure 22: Proposed flow for labelling model governance

This model will be proposed and approved by the Technical Steering Committee for future labelling processes.

## **4.4 Labelling functional aspects**

In order to label GEs according to *functional* attributes, a table as the following one was filled, with an evaluation of three main product attributes: Functionality, Reliability and Efficiency. Each one will be split in a set of related measures and the average of them would be considered.

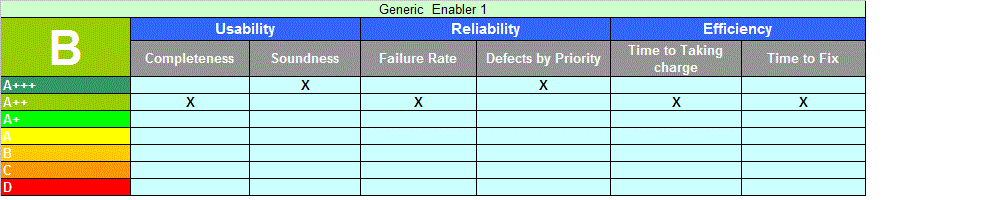


Figure 23: Functional-related labels for GE

The proposed measures were introduced in the following table and later discussed in detail:

|  |  |  |  |
| --- | --- | --- | --- |
| **Prod. Attribute** | **Derived Measure** | **Base Measures** | **Formula** |
| Usability | Completeness | Nominal value based on document verification |  |
| Soundness | Nominal value based on document verification |  |
| Reliability | Failure Rate | Total number of Test failed (TF)  Total number of Test cases executed (TE) | TF/TE |
| Defects by Priority | Total number of Blocking/Critical level Defects (DB)  Total number of Major level Defects (DM)  Total number of Minor/Trivial level Defects (DT)  Total number of Defects (D) | (DB\*w1+DM\*w2+  DT\*w3)/D    Note: *Defect Blocking/Critical weight (w1=3)*  *Defect Major weight (w2=2)*  *Defect Minor/Trivial weight (w3=1)* |
| Efficiency | Time to Taking charge | Date of taking charge of each defect (TD)  Date open for each defect (OD)  Total number of Closed defects (D) | ∑(TD-OD)/D |
| Time to Fix | Date of fix for each defect (FD)  Date of taking charge of each defect (TD)  Total number of Closed defects (D) | ∑(FD-TD)/D |

Table 8: Calculation criteria for functional labels

**1.** **Usability measures**

This set of measures evaluates the ease of use of documentation, therefore the ability to find “the right information at the right time”, verifying that a document is complete, findable and consistent.

**1.1. Completeness measure**

This is a nominal measure based on the documentation availability and its completeness to ensure the document usability.

The analyzed items, sorted by relevance, are:

· Open Specification

· Installation guide

· Docker for installation

· Course or Tutorial in Academy site

|  |  |  |  |
| --- | --- | --- | --- |
| **Completeness** | | | |
| **Label** | **Value** | **Base Measures** | **Formula** |
| A+++ | Excellent | Nominal value based on document verification | Each expected document is available. The information is exhaustive, easily accessible and easy to use. There are examples, comments or other utilities that improve the reading/comprehension. |
| A++ | Very good | Each expected document is available. The information are exhaustive, easily accessible and easy to use. |
| A+ | Good | Each expected document is available. The information are fully exhaustive. |
| A | Fair | Each expected document is available and enough exhaustive |
| B | Poor | Each expected document is available but the information are not always exhaustive, easily accessible and easy to use. |
| C | Very poor | Some documents are missing. Those available are not always exhaustive, easily accessible and easy to use. |
| D | Bad | All or many documents are missing |

Table 9: Measurement criteria for Completeness evaluation

**1.2.** **Soundness measure**

This is a nominal measure based on the documentation soundness to ensure the correctness of APIs usability.

The analyzed items (sorted by relevance) are:

· Programmer’s Guide

· Software Package

· APIs Specification

|  |  |  |  |
| --- | --- | --- | --- |
| **Soundness** | | | |
| **Label** | **Value** | **Base Measures** | **Formula** |
| A+++ | Excellent | Nominal value based on document verification | The information is exhaustive, easily accessible and easy to use. There are examples, comments or other utilities that improve the reading/comprehension. |
| A++ | Very good | The information are exhaustive, easily accessible and easy to use. |
| A+ | Good | The information are fully exhaustive. |
| A | Fair | The information is enough exhaustive |
| B | Poor | The information are not always exhaustive, easily accessible and easy to use. |
| C | Very poor | Some information is missing. Those available are not always exhaustive, easily accessible and easy to use. |
| D | Bad | Core information are missing |

Table 10: Measurement criteria for Soundness evaluation

**2.** **Reliability measures**

This second set of measures is about the capability of a GE to be reliable as much as possible in order to maximize its MTBF (Mean Time Between Failures). Even if this would be a product attributes in ISO/IEC 25010:2011, (thus as a non-functional attributes), here it should be seen as a set of attributes (and related measures) for showing how much it should be fixed for obtaining a proper ‘uptime’ for the observed GE.

**2.1.** **Failure Rate measure**

It determines the % of Failures in the executed test cases.

This measure is calculated from data gathered by Functional Test Reports.

|  |  |  |  |
| --- | --- | --- | --- |
| **Failure Rate** | | | |
| **Label** | **Value** | **Base Measures** | **Formula** |
| A+++ | < 0,05 | Total number of Test failed (TF)  Total number of Test cases executed (TE) | TF/TE |
| A++ | 0,05 - 0,16 |
| A+ | 0,17 - 0,27 |
| A | 0,28 - 0,38 |
| B | 0,39 - 0,49 |
| C | 0,50 - 0,6 |
| D | > 0,6 |

Table 11: Measurement criteria for Failure Rate evaluation

**2.2.** **Defects by Priority metric**

It determines the number of defects based on “Blocking/Critical” - “Major” – “Minor/Trivial” Priority (Service Desk’s viewpoint)

This measure is calculated from data gathered by a Jira installation (<https://www.atlassian.com/software/jira>)

|  |  |
| --- | --- |
| Priority | Description |
| Blocking/Critical | This has to be fixed immediately. This generally occurs in cases when an entire functionality is blocked and no testing can proceed as a result of this or, in certain other cases, if there are significant memory leaks |
| Major | Normally when a feature is not usable as it’s supposed to be, due to a program defect, or that a new code has to be written or sometimes even because some environmental problem has to be handled through the code |
| Minor/Trivial | A defect with low priority indicates that there is definitely an issue, but it doesn’t have to be fixed to match the “exit” criteria. However this must be fixed before the delivery. Sometimes it could be even a cosmetic error. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Defects by Priority** | | | |
| **Label** | **Value** | **Base Measures** | **Formula** |
| A+++ | < 1,5 | Total number of Blocking/Critical level Defects (DB)  Total number of Major level Defects (DM)  Total number of Minor/Trivial level Defects (DT)  Total number of Defects (D) | (DB\*w1+DM\*w2+DT\*w3)/D  Note: Defect Blocking/Critical weight (w1=3)  Defect Major weight (w2=2)  Defect Minor/Trivial weight (w3=1) |
| A++ | 1,5 - 1,7 |
| A+ | 1,71 - 1,9 |
| A | 1,91 - 2,1 |
| B | 2,11 - 2,3 |
| C | 2,31 - 2,5 |
| D | > 2,5 |

Table 12: Measurement criteria for Defects by Priority evaluation

**3.** **Efficiency Metrics**

This third set of measures is about the capability of a Service Desk (SD) to solve incidents related to GEs and manage them for maximizing the customer/user satisfaction.

**3.1.** **Time to Taking charge measure**

It determines the number of average working days valuated from the opening date to the taking charge date of the defect.

This measure is calculated from data gathered by a Jira installation (<https://www.atlassian.com/software/jira>)

|  |  |  |  |
| --- | --- | --- | --- |
| **Time to Taking charge** | | | |
| **Label** | **Value** | **Base Measures** | **Formula** |
| A+++ | < 1 | Date of taking charge of each defect (TD)  Date open for each defect (OD)  Total number of Closed defects (D) | ∑(TD-OD)/D |
| A++ | 1 - 3,7 |
| A+ | 3,8 - 6,5 |
| A | 6,6 - 9,3 |
| B | 9,4 - 12,1 |
| C | 12,2 - 15 |
| D | > 15 |

Table 13: Measurement criteria for Time to Taking evaluation

**3.2.** **Time to Fix measure**

It determines the number of average working days valuated from the taking charge date to the fix date of the defect.

This measure is calculated from data gathered by a Jira installation (<https://www.atlassian.com/software/jira>)

|  |  |  |  |
| --- | --- | --- | --- |
| **Time to Fix** | | | |
| **Label** | **Value** | **Base Measures** | **Formula** |
| A+++ | < 1 | Date of fix for each defect (FD)  Date of taking charge of each defect (TD)  Total number of Closed defects (D) | ∑(FD-TD)/D |
| A++ | 1 - 4,7 |
| A+ | 4,8 - 8,5 |
| A | 8,6 - 12,3 |
| B | 12,4 - 16,1 |
| C | 16,2 - 20 |
| D | > 20 |

Table 14: Measurement criteria for Time to Fix evaluation

The GEs have been labelled according to the functional measures and for each one are reported the measures. The GEs labelled are the same labelled for the non-functional aspects. See section 4.6 for details.

## **4.5 Labelling non-functional aspects**

In order to label GEs according to non-functional attributes, a table as below was filled, with an evaluation of three measures: Scalability, Stability and Performance. In the next pages, the way to fill these categories is explained.

**1. Scalability measure**

In this section, the behavior of the GEri when the load increased was studied.

In order to assess the Stability, we have to choose a fixed interval where the system has already reached the maximum requests per second data, and the system is not failing yet. For that fixed time interval, we calculate the ratio between final/starting response time and the ratio between final/starting thread number. Then, we calculate growing response time ratio / growing thread number ratio and check the result in the next table:

|  |  |
| --- | --- |
| Growing Response Time ratio/ Growing thread number ratio | |
| **Label** | **Value** |
| A+++ | < 1.05 |
| A++ | 1.22 - 1.05 |
| A+ | 1.42 - 1.21 |
| B | 1.74 - 1.43 |
| C | 2.15 - 1.75 |
| D | 2.6 - 2.16 |
| E | > 2.6 |

Table 15: Value ranges for scalability evaluation

**2. Stability measure**

This is the evolution of the system along the time. To label the stability, the results in a stability scenario have to be checked. Then, it would be labeled depending on the existence of leaks:

|  |  |
| --- | --- |
| Stability[rz3] [CLH4] | |
| **Label** | **Value** |
| A+++ | Nor CPU nor memory increases in the whole test (when the load is stable) |
| A++ | Memory usage is a little higher at the end of the test than at the beginning (probably due to data generated). |
| A+ | Memory and CPU usages are a little higher at the end of the test |
| A | Memory and/or CPU are significantly higher at the end of the test, but doesn’t seem to exist a leak |
| B | Memory leak avoidable with configuration or load limitation |
| C | Memory leak not avoidable. The system crashes after a few hours. |
| D | High leak. System crashes before half an hour |

Table 16: Value ranges for stability evaluation

**3. Performance measure**

To evaluate performance, the maximum request handling per second would be considered. Because the requests of different GE’s can have different complexity, a study of the functionality must be done before. The most common functionalities would be considered in the next points.

**3.1 GEs with NGSI attribute updates**

These are the GEs which are capable of create/update NGSI entities and attributes. In this category is included, for example, the Context Broker GE.

#### Update attribute measuring

For this measure, is needed to find the most production point at first (the first point where the maximum responses per second rate is reached) for request that updates attributes. Then, at this point, response times and responses per second have to be measured.

#### Updated attributes per second

In order to do an equitable comparison, it has to be considered the number of the attributes which are updated in each request. The reason is that the processing effort is higher if is needed to do more update transactions in database, and some other operations. If this is a random number (for example, each request updates a random number of attributes, between 1 and 3), then the average number can be used (in this example, two attribute updates would be considered). When that number is found, it has to get the number of attribute updates per second. For example, if in the maximum production number, the responses per second rate is 200, and (as seen before), the average attributes updates per requests is 2, then the number of attribute updates per request is 400. In our example, it would be labelled with a ‘B’ according to the table below.

|  |  |
| --- | --- |
| Updated attributes/second | |
| **Label** | **Value** |
| A+++ | > 591 |
| A++ | 541 – 590 |
| A+ | 481 – 540 |
| A | 391 – 480 |
| B | 271 – 390 |
| C | 141 – 270 |
| D | <140 |

Table 17: Value ranges for performance (updated att/sec) evaluation

**3.2 GEs with event receiving**

These are the GEs which are capable to handle events, for example, the Proton GE. For evaluating them, it is considered their performance in responding to HTTP requests containing events.

#### Event receiving measuring

As before, it is needed to find the most production point at first (the first point where the maximum responses per second rate is reached) for requests that update attributes. Then, at this point, response times and responses per second have to be measured.

#### Events per second

Despite the updated attributes case, usually only an event for each HTTP request is sent. The next table shows how to label the event processing.

|  |  |
| --- | --- |
| Events/second | |
| **Label** | **Value** |
| A+++ | > 492 |
| A++ | 451 – 492 |
| A+ | 401 – 491 |
| A | 313 – 400 |
| B | 195 – 312 |
| C | 94 – 192 |
| D | < 94 |

Table 18: Value ranges for performance (events/sec) evaluation

Again using our previous example, it would be labelled with a ‘B’ according to the table.

**3.3. WebRTC based GEs**

GEs based on WebRTC protocol fits into this category. For these GEs, standard metrics for HTTP protocol based GEs cannot be used. Instead of that, following metrics are needed.

#### Bit Rate metric

In this case, the most meaning value in order to assess the performance is the Total Bit Rate that the system can serve. For that, it is needed to multiply the Bit Rate by the number of users.

#### Bit rate and users number

The next table shows how to label in function of the Bit Rate and users number.

|  |  |
| --- | --- |
| (Bit Rate \* users number) ratio | |
| **Label** | **Value** |
| A+++ | > 4'5 Mbps |
| A++ | 2'01 - 4'5 Mbps |
| A+ | 1'01 - 2 Mbps |
| A | 0'501 Mbps - 1 Mbps |
| B | 251 Kbps - 500 Kbps |
| C | 100 Kbps - 250 Kbps |
| D | < 100 Kbps |

Table 19: Value ranges for performance (bit rate/user) evaluation

In our example, it would be labelled with an ‘A++’ according to the table.

**3.4. GEs handling authentication/authorisation request**

Some GEs provide the most used features handling HTTP requests that imply operations such as the user authentication and/or authorisation. These operations can be directly invoked by a hypothetical IdM GE as well as indirectly by another GE that strictly depends on the first one. In order to evaluate both operations, their performances in response to HTTP requests have to be taken separately and then merged in a unique value through the process described below.

#### Completed request measuring

Firstly, starting from the output of the non-functional tests, the already measured response times and the number of responses per second have to be taken into account in order to find the most production point (the first point where the maximum responses per second rate is reached) for completed transactions; secondly, since authorization and authentication operations have different costs, they are weighted in order to make them comparable to each other and finally their average is calculated. The resulting value is then checked with the table provided below in order to find the range which fits with this value and thus the corresponding label can be identified. The operation cost to be taken into account, as suggested for the case of Generic GE, is meant in terms of DB operations -of any type- originated by each type of request. For example, on the one hand, an “authentication” request, in order to be fulfilled, involves 30 DB operations; on the other hand, the corresponding “authorization” involves only 2 DB operations. The ratio between the potential cost of the two operations is thus equal to 30/2=15. A half point of weight is then assigned for each unit of the resulting ratio (0,5 X 15) and the result used as multiplying factor of the “authentication” most production point calculated earlier. Finally, the two values of the most production point are averaged and the resulting value compared with the ranges provided by the following table.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Authentication/Authorization requests/second | | | **Label** | **Value** | | A+++ | > 591 | | A++ | 541 – 590 | | A+ | 481 – 540 | | A | 391 – 480 | | B | 271 – 390 | | C | 141 – 270 | | D | <140 | |  |

Table 20: Value ranges for performance (auth reqs/sec) evaluation

**3.5. Generic GE**

These are the GEs which not fit into the other categories. For labeling other GEs which do not fit in the ones mentioned before, a ponderation should be done, comparing the functionality of the GE with one reference functionality (for example, attribute updates). In order to do this, it must be considered:

* Number of operations in the database per request
* Connections with other systems
* Functionality complexity

For example, an NGSI attribute update makes 2 database operations (1 select and 1 update queries). If a functionality makes 4 database operations, the functionality complexity appears to be similar, and it makes no connections with other systems, then it would be considered like a 2 NGSI attributes update query.

#### Generic GEs not involving DB operations

This category specializes the aforementioned “Generic GE” category to those components that on the one hand do not fit with any dedicated category, but on the other hand they have in common the characteristic of not using any DB system. Therefore this type of GE could be tested in a completely isolated way, namely with no other systems connected.

#### Completed request measuring

Firstly, starting from the output of the non-functional tests, the already measured response times and the number of responses per second have to be taken into account in order to find the most production point (the first point where the maximum responses per second rate is reached) for completed transactions. This value is then directly compared with the table provided below in order to find the range which fits with it and thus the label to be assigned.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Requests/second | | | **Label** | **Value** | | A+++ | > 7200 | | A++ | 6701 – 7200 | | A+ | 5901 – 6700 | | A | 4701 – 5900 | | B | 3201 – 4700 | | C | 1600 – 3200 | | D | <1600 |   Table 21: Value ranges for performance (requests/sec) evaluation |  |

**3.6 Generic GEs applied to PEP Proxy GE**

This category defines a range of values for classifying the results coming out from the test carried out on any implementation of the PEP Proxy GE.

#### Completed request measuring

Likewise the originating category that this one specializes, firstly, the already measured response times and the number of responses per second have to be taken into account in order to find the most production point (the first point where the maximum responses per second rate is reached) for completed transactions. This value is then directly compared with the table provided below in order to find the range which fits with it and thus with the label to be assigned.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Requests/second | | | **Label** | **Value** | | A+++ | > 1340 | | A++ | 1221 – 1340 | | A+ | 1061 – 1220 | | A | 821 – 1060 | | B | 561 – 820 | | C | 281 – 560 | | D | <280 | |  |

Table 22: Value ranges for performance (requests/sec) evaluation (PEP Proxy)

### **4.5.1. Benchmarking for adjusting the non-functional results**

Due to the possible hardware differences between different GEs’ performance tests, it is necessary to adjust the metrics depending on the results of a benchmark.

For the adjustment mentioned before, the benchmark “Phoronix Test Suite” has been used. Inside Phoronix, benchmark “pts/apache-1.6.1” was used. From this benchmark, a score is obtained that is needed to get a weighting rate. Annex 3 proposes the steps to get this score.

To get this, we take as reference the value “**16733.24**”, which is the score obtained for a server reference. With this reference value “x”, and the score “y” obtained by the benchmark, the ponderation rate “z” is obtained with the formula z = y/x. For example, for a score of 22013.47, the weighting rate would be 22013.47 /16733.24 = 1.315

Once the weighted rate has been found, it is easy to get the weight for a measure to be multiplied by the value by the weighting rate when the measure is “lower is better”, and subdividing when the measure is “higher is better”.

For example, if the weighting rate is 1.315, and we have a performance value of 600 NGSI attribute request per second (higher is better), then we have to divide 600 by 1.315, which result is 456,27, labelled with a ‘B’.

## **4.6 Labelling FIWARE GEs**

In order to validate the labelling model, a set of GEs were selected to test the model and obtained their labels. The selected GEs (10 in total) have been tested both from functional and non-functional aspects, so they are a subset of the overall tested GEs, but enough for demonstrating whether the model is valid or it must be adjusted.

After gathering values for the defined FUR/NFR measures from all the observed GEs, here the summary:

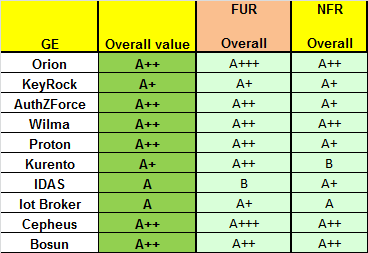


Figure 24: Consolidated labelling (overall values) for 10 observed GEs

For instance, looking at Orion, the overall value is ‘A++’, with a higher value from functional-related attributes (A+++) more than from its non-functional-related ones (A++). But in order to better understand which attribute could be the one on which improving the rating with a further implementation with priority, here in the following figures there is the detailed for the ‘two sides of the story’ (FUR and NFR).

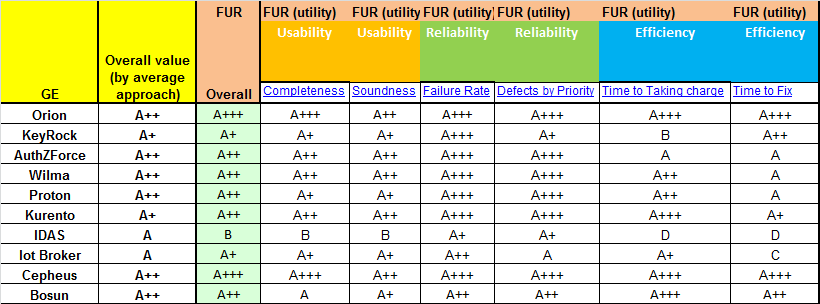


Figure 25: Consolidated functional labelling (detailed values) for 10 observed GEs

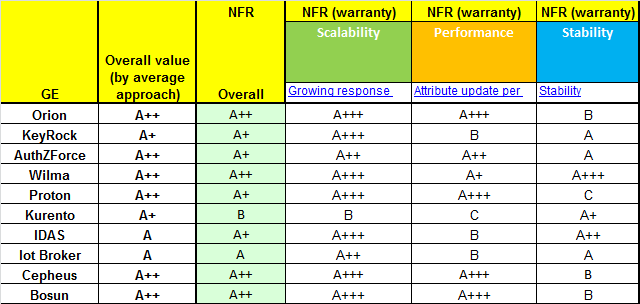


Figure 26: Consolidated non-functional labelling (detailed values) for 10 observed GEs

The spreadsheet with the detailed labelling by GE and category is available at <https://forge.fiware.org/docman/view.php/7/5716/Labelling_Results_Func_Measures_v08.xlsx>.

In Annex 3 explanations about how the labels were obtained for some GEs are included.

# 5 Conclusions

This task was conceived with the goal to be an instrument to demonstrate and improve the FIWARE GEs quality in order to make them more reliable and trusty. Along the process, the task has become very useful for the GEs developers to understand better the behavior of their components and the limits they had, allowing them to correct defects and improve documentation, performance, to make them more stable or to provide better training courses.

The starting was not easy as there were many things to test, from different sources, having to test GEs from very different nature and technologies, and overcoming the initial reluctance of collaboration since the task was seen as an overhead and intrusive for some GE owners. But, slowly these drawbacks were overpassed and the testing process became fluent and continuous.

Now, after several phases in the process some overall conclusions can be stated. There exists a significant heterogeneity in the GEs quality, having more mature GEs and ready for market than others. The stated labels are proof of that, having diversity of quality levels. Although there is always a room for improvement in documentation and support for most of the GEs, both of them have improved significantly during last year of the task. It can be also stated significant improvements in performance from first iteration to the second and third one, due to the addressing of recommendations through the iterative testing reports by the GE responsible, which is also a demonstration of the value this activity can bring to FIWARE.

In near future, the main focus will be to enlarge number and type of tests and to automate the tests as much as possible, including them as part of the software development process. Also the visualization of the labels in the Catalogue will be updated automatically as soon as a new label was changed due to new value in tested criteria. In the meantime, a set of guidelines have been created in order to be able to replicate all the conducted tests by anyone.

# ANNEX 1: Non-functional (stress) testing report

## **A1.1 Non-functional testing reports (for version 5.1)**

[Orion stress testing report (release 5.1)](https://forge.fiware.org/docman/view.php/7/5531/FIWARE-GEri_test_report_Orion_0_26_0+%281%29.docx)

[Bosun stress testing report (release 5.1)](https://forge.fiware.org/docman/view.php/7/5526/FIWARE-GEri_test_report_Bosun_2_3_0+%281%29.docx)

[Cepheus stress testing report (release 5.1)](https://forge.fiware.org/docman/view.php/7/5527/FIWARE-GEri_test_report_Cepheus_0_1_4+%281%29.docx)

[IDAS stress testing report (release 5.1)](https://forge.fiware.org/docman/view.php/7/5528/FIWARE-GEri_test_report_IDAS_1_31_1+%281%29.docx)

[KeyRock stress testing report (release 5.1)](https://forge.fiware.org/docman/view.php/7/5529/FIWARE-GEri_test_Keyrock_v1.3.docx)

[Kurento stress testing report (release 5.1)](https://forge.fiware.org/docman/view.php/7/5530/FIWARE-GEri_test_report_Kurento_6_2_0+%281%29.docx)

[Proton stress testing report (release 5.1)](https://forge.fiware.org/docman/view.php/7/5532/FIWARE-GEri_test_report_PROTON_4_0+%281%29.docx)

[Wilma stress testing report (release 5.1)](https://forge.fiware.org/docman/view.php/7/5524/FIWARE-GEri_test_PEPWilma_v1.5.docx)

[IoTBroker stress testing report (release 5.1)](https://forge.fiware.org/docman/view.php/7/5525/FIWARE-IoTBroker5.2.3-Performance+test+v1.0.docx)

## **A1.2 Non-functional testing reports (for version 5.2)**

[Bosun stress testing report (release 5.2)](https://forge.fiware.org/docman/view.php/7/5588/FIWARE-GEri_test_report_Bosun_2_5_0.docx)

[Cepheus stress testing report (release 5.2)](https://forge.fiware.org/docman/view.php/7/5589/FIWARE-GEri_test_report_Cepheus_0_1_8.docx)

[KeyRock stress testing report (release 5.2)](https://forge.fiware.org/docman/view.php/7/5618/FIWARE-GEri_test_Keyrock-5.2.0_v1.0.docx)

[Kurento stress testing report (release 5.2)](https://forge.fiware.org/docman/view.php/7/5590/FIWARE-GEri_test_report_Kurento_6_5_0.docx)

[Orion stress testing report (release 5.2)](https://forge.fiware.org/docman/view.php/7/5547/FIWARE-GEri_test_report_Orion_1.0.0.docx)

[Wilma stress testing report (release 5.2)](https://forge.fiware.org/docman/view.php/7/5617/FIWARE-GEri_test_PEPWilma-5.2.0_v1.0.docx)

## **A1.3 Non-functional testing reports (for version 5.3)**

[Bundle Orion-IDAS-Cygnus stress testing report](https://forge.fiware.org/docman/view.php/7/5657/FIWARE-GEri_test_Bundle_IDAS_Orion_Cygnus_5.3.0.docx)

[Bundle Wilma-KeyRock-AuthZForce stress testing report](https://forge.fiware.org/docman/view.php/7/5614/FIWARE-GEri-bundle_test_authzforce-wilma-keyrock-5.3.0.doc)

[IoTBroker stress testing report (release 5.3)](https://forge.fiware.org/docman/view.php/7/5615/FIWARE-GEri_test_IoTBroker_5.3.0.docx)

[AuthZForce stress testing report (release 5.3)](https://forge.fiware.org/docman/view.php/7/5616/FIWARE-GEri_test_AZforce-5.3.0.docx)

[IDAS stress testing report (release 5.3)](https://forge.fiware.org/docman/view.php/7/5619/FIWARE-GEri_test_IDAS_5.3.0.docx)

[KeyRock stress testing report (release 5.3)](https://forge.fiware.org/docman/view.php/7/5639/FIWARE-GEri_test_Keyrock-5.3.0.doc)

[Orion stress testing report (release 5.3)](https://forge.fiware.org/docman/view.php/7/5687/FIWARE-GEri_test_Orion_5.3.0.docx)

[Orion scalability testing report (release 5.3)](https://forge.fiware.org/docman/view.php/7/5620/FIWARE-GEri_Scalability_Test_Orion-5.3.0.docx)

[Wilma stress testing report (release 5.3)](https://forge.fiware.org/docman/view.php/7/5640/FIWARE-GEri_test_PEPWilma-5.3.0.docx)

[Proton stress testing report (release 5.3)](https://forge.fiware.org/docman/view.php/7/5725/FIWARE-GEri_test_Proton_5.3.0.docx)

[Bosun stress testing report (release v5.3)](https://forge.fiware.org/docman/view.php/7/5729/FIWARE-GEri_test_Bosun_5.3.0.docx)

# ANNEX 2: Functional testing report

The functional testing activity reports the results in different spreadsheets listed below:

1. The spreadsheet containing the Test Cases list and the Test execution information used and reported by the functional testing action can be downloaded[here](https://forge.fiware.org/docman/view.php/7/5712/Report+on+FIWARE+QA+Testing-Functional+Test.xls).

2. The spreadsheet containing the Academy Courses verifications and results can be downloaded [here](https://forge.fiware.org/docman/view.php/7/5713/FIWARE-Academy_Courses_Test.xls).

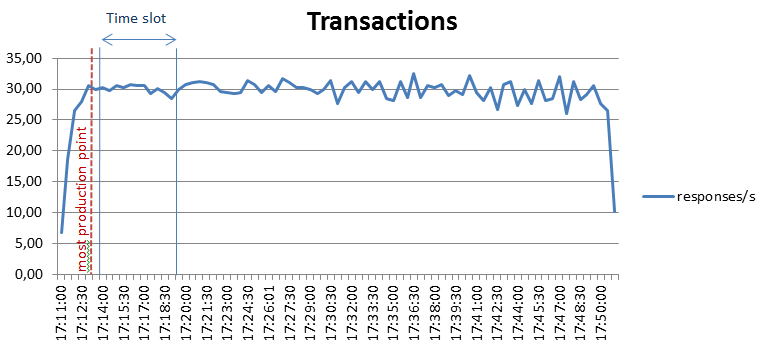
3. The spreadsheet containing the Bundle Functional Integration test can be downloaded [here](https://forge.fiware.org/docman/view.php/7/5730/FIWARE_BundleIntegrationTest_v1.docx).

# ANNEX 3: Non-functional labelling for some GEs

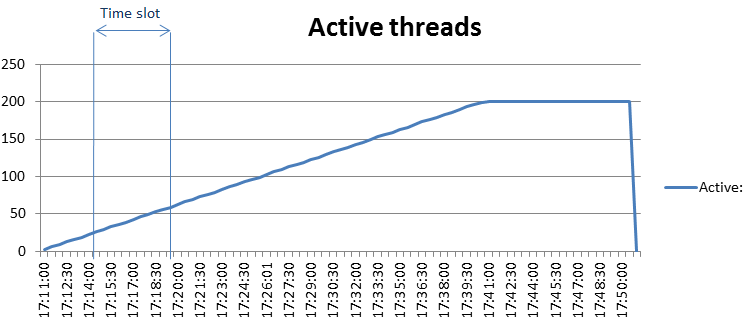
## **A3.1 IoTBroker non functional labelling**

#### Scalability

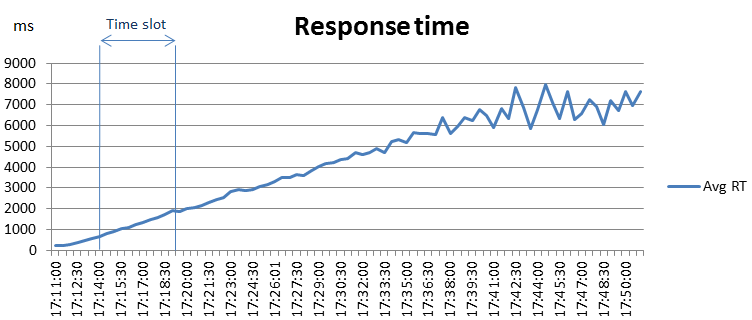
Once defined the most production point (red line in the Transactions chart below which corresponds to 30,5 requests/s with 16 concurrent threads), a fixed interval is chosen where the transactions per second rate is stable, after this point; this interval is represented by the time slot of the chart below which is five minutes wide being between 17:14:00 and 17:19:00.



The Active threads chart area correlated to this time slot shows a beginning value of 23 threads and ending value of 56 threads, thus the calculated threads ratio is 56/23=2,43.



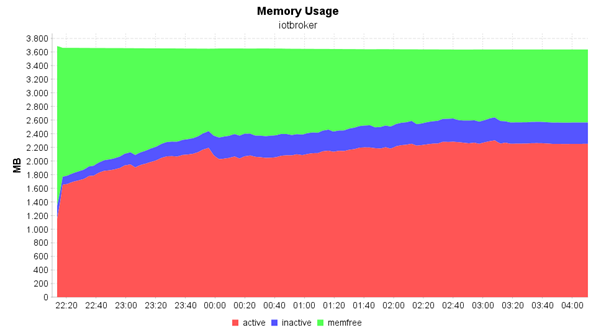
On the other hand, the Response Time chart area correlated to this time slot shows a beginning value of 688 ms and ending value of 1932 ms, thus the calculated growing response times ratio is 1932 /688 =2,81.

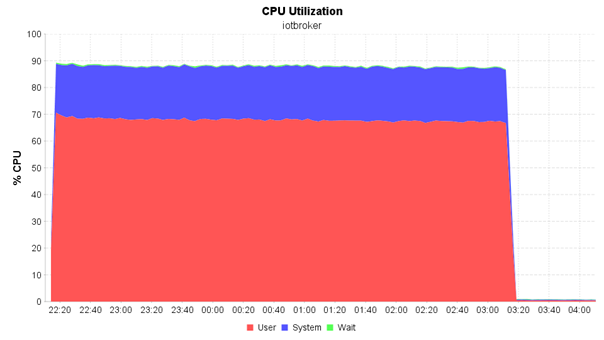


Given those two ratios, the final ratio is: 2,81/2,43 = **1,15**, which corresponds to a **‘A++’ label.**

#### Stability

Results from a stability scenario, which involved the QueryContext API, are considered for this measure. The charts below show a high memory usage as well as a significant CPU usage (average usage of CPU=67,84%, with peak of 84,80% and RAM=2,08 GB with peak of 2,33 GB).





It is useful to compare the memory chart with the CPU utilization in order to highlight where, in the memory chart, the test finished, otherwise it would have been impossible; in fact at this time (03:20) CPU load fell down while memory usage stopped its growing trend, but remaining constant afterwards, without ever decreasing. The described trend leads therefore to label the stability of this GEri with an “**A**” (“Memory and/or CPU are significantly higher at the end of the test, but it doesn’t seem to exist a leak”).

#### Performance

Finally, about performance, the Aeron IoT Broker belongs to “GEri’s with NGSI attribute updates” category.

As shown previously in the Scalability evaluation by the Transactions chart (from a stress scenario), the number of transactions per second that the system can handle is 30,5.

In this scenario, each operation did a random number of attribute updates between 1 and 20. Taking into account the attribute value of 10 as average number of updates per operation, the system demonstrated to be able to fulfil 30,5 X 10 = **305** attribute updates per second; once this value is weighted –in order to take into account the HW involved- it reaches the value of **388,95** as shown by the following calculations, which classifies system performance with a ‘**B**’ value.

Phoronix reference value = 16733,24

Phoronix score for this VM where tests took place = 13121,50

Ponderation rate = 13121.50/16733.24 = 0,78

Weighted value = 305/0,78 = **388,95**

As a summary for non-functional attributes, a table is shown below:

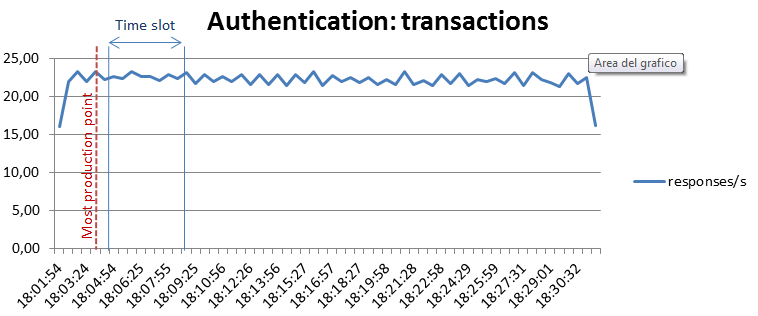
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Measure name** | **Gathered Value** | **Label (A+++/D)** | **Notes/Comments/ Objective Evidences (OE)** |
| Performance | Attribute updates per second | 388,95 | B | See 4.5.3 Performance |
| Scalability | Growing response time /thread number ratio | 1,15 | A++ | See 4.5.1 Scalability |
| Stability | Stability | Memory and/or CPU are significantly higher at the end of the test, but it doesn’t seem to exist a leak. | A | See 4.5.2  Stability |

## **A3.2 KeyRock IdM non functional labelling**

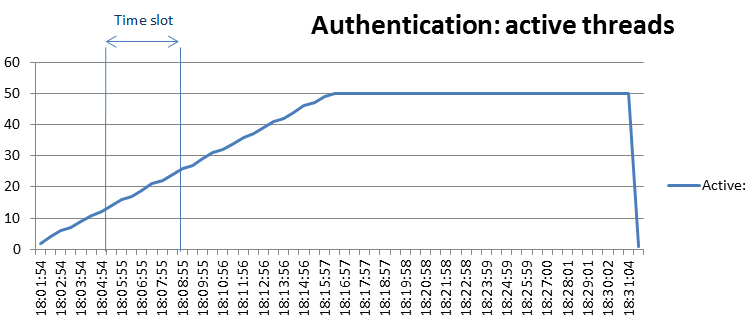
#### Scalability

The most production point is taken for the operations authorization and authentication as illustrated by the following charts.

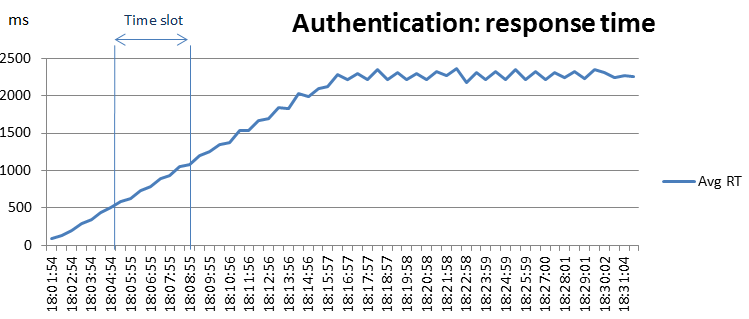
For the Authentication operation the most production point is placed at 23,30 requests/s with 9 concurrent threads as shown by the related transactions chart below; after this point a fixed interval of five minutes is chosen where the transactions per second rate is stable;



The Active threads chart area correlated to this time slot shows a beginning value of 12 threads and ending value of 29 threads, thus the calculated threads ratio is 29/12=2,42.

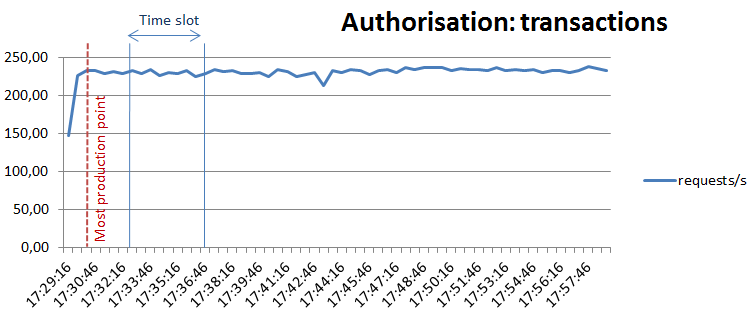


On the other hand, the Response Time chart area correlated to this time slot shows a beginning value of 501 ms and ending value of 1250 ms, thus the calculated growing response times ratio is 1250/501 =2,49.

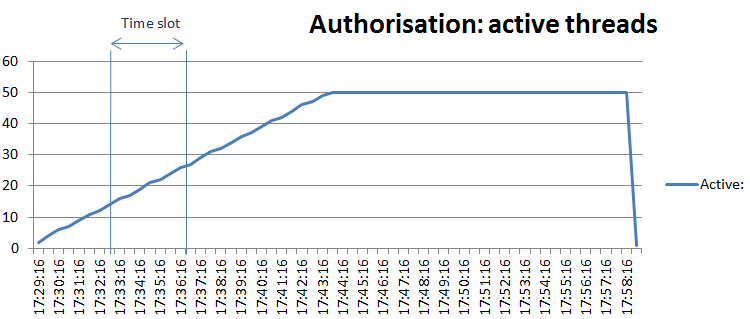


Given those two ratios, the final ratio for the Authentication operation is: 2,49/2,42 = **1,03**, which corresponds to a **‘A+++’ label.**

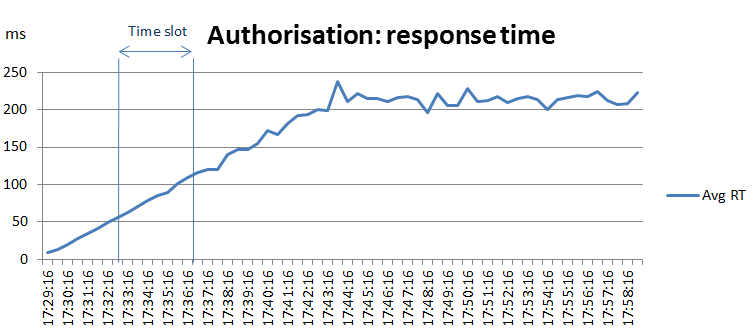
For the Authorisation operation the most production point is placed at 232 requests/s with 7 concurrent threads as shown by the related transactions chart below; after this point a fixed interval of five minutes is chosen where the transactions per second rate is stable;



The Active threads chart area correlated to this time slot shows a beginning value, as already seen for the Authoentication, of 12 threads and ending value of 29 threads, thus the calculated threads ratio is 29/12=**2,42**.



On the other hand, the Response Time chart area correlated to this time slot shows a beginning value of 48,73 ms and ending value of 119,87 ms, thus the calculated growing response times ratio is 119,87/48,73 =**2,46**.

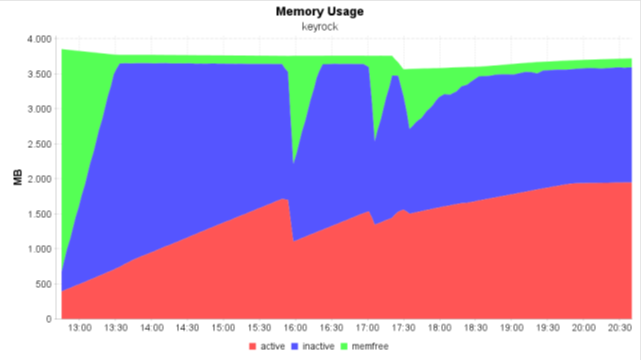


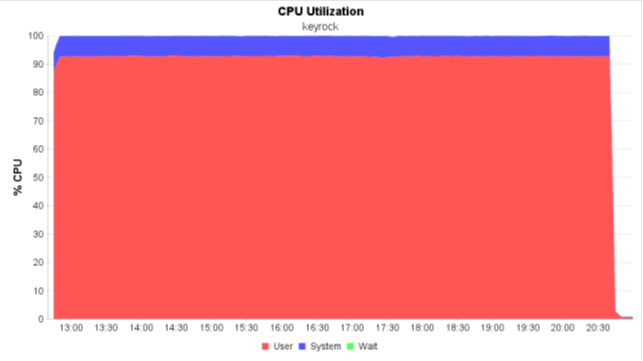
Given those two ratios, the final ratio for the Authentication operation is: 2,42/2,46 = **1,02**, which corresponds to a **‘A+++’ label.**

Taking into account both of the most used operations of this GE, the Scalability of is rated with **A+++** label.

#### Stability

The trend depicted in the memory chart below, which is taken from the stability test scenario results -8 hours long- shows a quite stable memory usage on the long run; in fact it can be noted an incremental memory usage in the first hours of test then partially released in few occurrences and stabilised near the end of the test. With this memory utilisation, as well as with that one registered by the CPU, and the good performance registered, no memory leak can be assumed and thus the assigned label is **A** (“Memory and/or CPU are significantly higher at the end of the test, but doesn’t seem to exist a leak”).





#### Performance

About performance, this GEri belongs to “GEri’s handling authentication/authorisation request” category.

The most production point identified earlier for the Scalability measure is taken into account for both of the most used operations of the GEri which are the Authentication and Authorisation.

The number of transactions per second that the system can handle, as regards the Authentication is 23,30, as regards the Authorisation is instead 232,00.

Authentication involves, as stated by the MySQL Server log, 32 database operations; on the other hand Authorisation operation involves only 2 database operations. Therefore the ratio of the potential cost between the two operations is 32/2 =16. Taking into account a half point of weight for each unit of the resulting ratio, the multiplying factor of the “authentication” most production point is equal to: 0,5 X 16 = 8, thus the weighted “authentication” most production point is 8 \* 23,30 = 186,40.

With these values the average most production point is calculated as following:

(Authentication MPP) 186,40 + 306,19 (Authorisation MPP)/2 = 246,30

Taking into account the Phoronix score, this value increases to **324** as shown by the following calculations, which classifies system performance with a ‘**B**’ value.

Phoronix reference value = 16733,24

Phoronix score for this VM where tests took place = 11296,98

Ponderation rate =12678.63/16733.24 = 0,76

Weighted value = 246,30 /0,76 = **324**

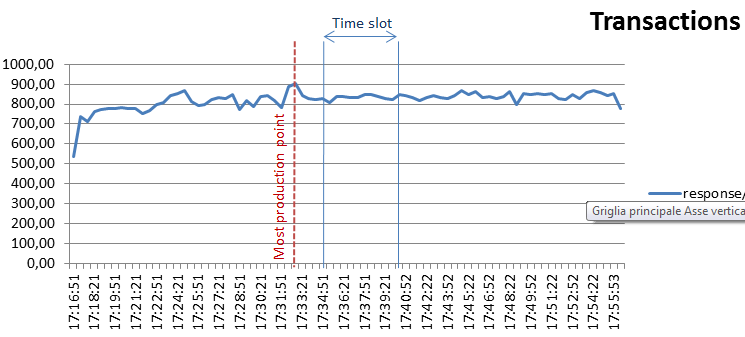
As a summary for non-functional attributes, a table is shown:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Measure name** | **Gathered Value** | **Label (A+++/D)** | **Notes/Comments/ Objective Evidences (OE)** |
| Performance | Requests fulfilled per second | 324 | B | See 4.5.3 Performance |
| Scalability | Growing response time /thread number ratio | 1,03 | A+++ | See 4.5.1  Scalability |
| Stability | Stability | Memory and/or CPU are significantly higher at the end of the test, but it doesn’t seem to exist a leak. | A | See 4.5.2  Stability |

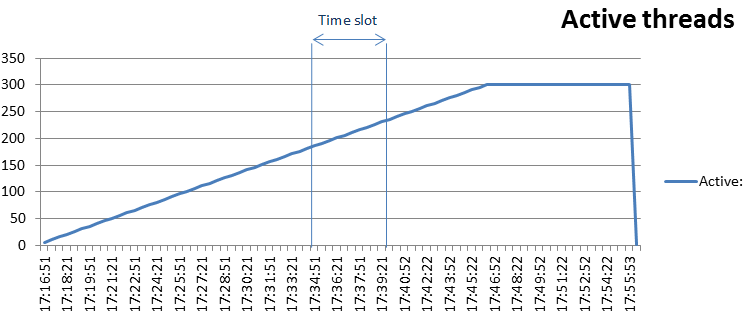
## **A3.3 Wilma PEP Proxy non functional labelling**

#### Scalability

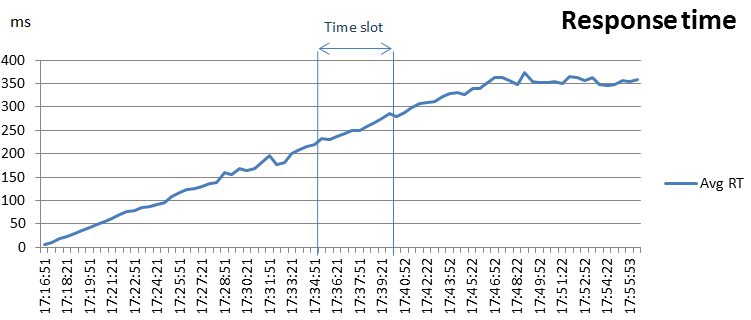
Once defined the most production point (red line in the Transactions chart below which corresponds to 901 requests/s with 166 concurrent threads), a fixed interval is chosen where the transactions per second rate is stable, after this point; this interval is represented by the time slot of the chart below which is five minutes wide being between 17:34:51 and 17:39:52.



The Active threads chart area correlated to this time slot shows a beginning value of 186 threads and ending value of 236 threads, thus the calculated threads ratio is 236/187 = **1,27**.



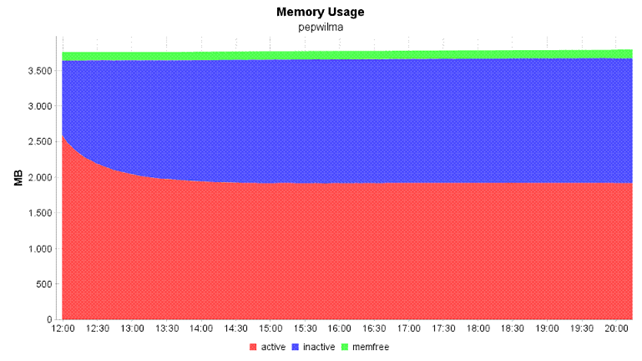
On the other hand, the Response Time chart area correlated to this time slot shows a beginning value of 220 ms and ending value of 285 ms, thus the calculated growing response times ratio is 285/220 = **1,29**.

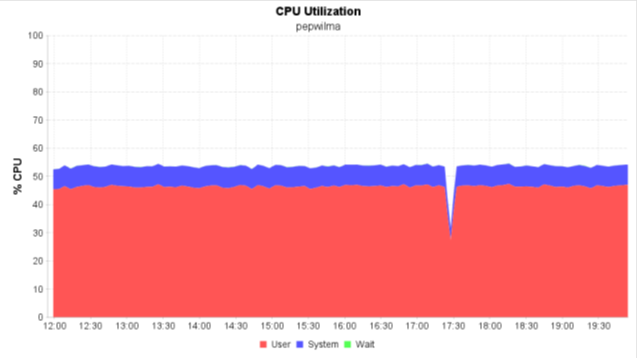


Given those two ratios, the final ratio is: 1,29/1,27 = **1,02**, which corresponds to a **‘A+++’ label.**

#### Stability

The trend depicted in the charts below, which is taken from the stability test scenario results -8 hours long- show, after an initial peak, a quite stable memory usage on the long run as well as a moderate CPU usage; with this resources utilisation, and the good performance registered, no memory leak can be assumed. For this reason, the assigned label is **A+++ (“**Nor CPU nor memory increases in the whole test -when the load is stable-”).





#### Performance

The number of transactions per second that the system can handle is **901**. Taking into account the Phoronix score, this already good value increases to **1126,25** as shown by the following calculations, which classifies system performance with an **A+** value.

Phoronix reference value = 16733,24

Phoronix score for this VM where tests took place = 13382,34

Ponderation rate = 13382,34/16733.24 = 0,80

Weighted value = 901/0,80 = **1126,25**

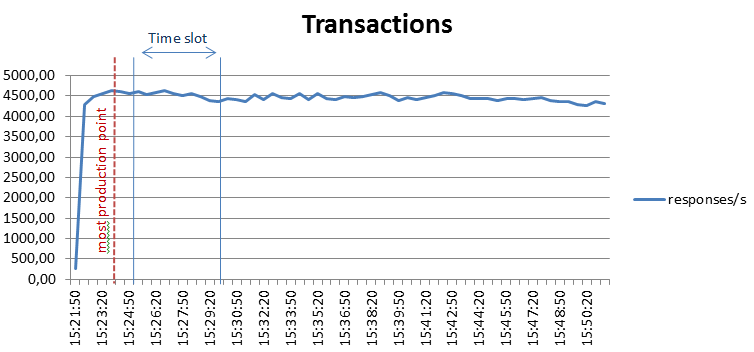
As a summary for non-functional attributes, a table is shown:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Measure name** | **Gathered Value** | **Label (A+++/D)** | **Notes/Comments/ Objective Evidences (OE)** |
| Performance | Requests fulfilled per second | 1126,25 | A+ | See 4.5.3 Performance |
| Scalability | Growing response time /thread number ratio | 1,02 | A+++ | See 4.5.1  Scalability |
| Stability | Stability | Nor CPU nor memory increases in the whole test (when the load is stable) | A+++ | See 4.5.2  Stability |

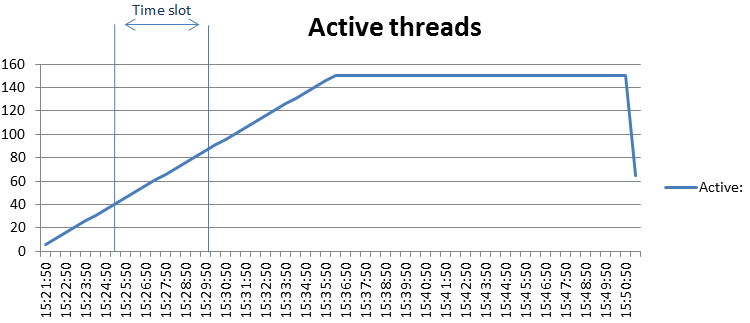
## **A3.4 AuthZForce PDP non functional labelling**

#### Scalability

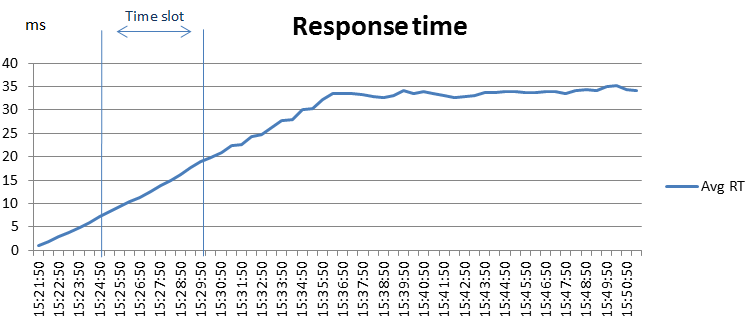
Once defined the most production point (red line in the Transactions chart below), a fixed interval is chosen where the transactions per second rate is stable, after this point; this interval is represented by the time slot of the chart below which is five minutes wide being comprised between 15:24:50 and 15:29:50.



The Active threads chart area correlated to this time slot shows a beginning value of 36 threads and ending value of 86 threads, thus the calculated **threads ratio** is 36/86=**2,39**



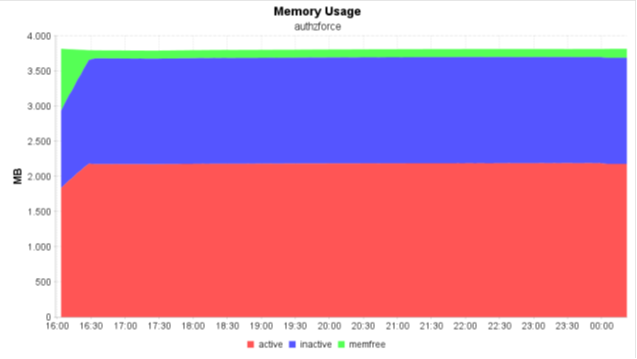
On the other hand, the Response Time chart area correlated to this time slot shows a beginning value of 7,2 ms and ending value of 19 ms, thus the calculated growing **response times ratio** is 19 /7,2 =**2,64**.

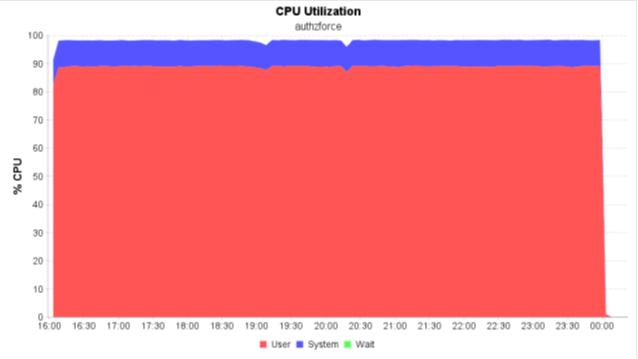


Given those two ratios, the **final ratio** is: 2,64/2,39 = **1,10**, which corresponds to a **‘A++’ label.**

#### Stability

The trend depicted in the memory chart below shows a quite stable memory usage during the whole period of 8 hours that along with good performance registered, no memory leak can be assumed; on the other hand, a high resource usage is depicted by the trend about the CPU. For this reason the assigned label is **A** (“Memory and/or CPU are significantly higher at the end of the test, but it doesn’t seem to exist a leak”).





#### Performance

About performance, the AuthZForce PDP GEri belongs to “Generic GEri’s not involving DB operations” category.

As shown previously in the Scalability evaluation by the Transactions chart (from a stress scenario), the number of transactions per second that the system can handle is 4.627. Taking into account the Phoronix score, this already good value increases to **6853,58** as shown by the following calculations, which classifies system performance with a ‘**A++**’ value.

Phoronix reference value = 16733,24

Phoronix score for this VM where tests took place = 11296,98

Ponderation rate = 11296,98/16733.24 = 0,68

Weighted value = 4.627/0,68 = **6853,58**

As a summary for non-functional attributes, a table is shown:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Measure name** | **Gathered Value** | **Label (A+++/D)** | **Notes/Comments/ Objective Evidences (OE)** |
| Performance | Requests fulfilled per second | 6853,58 | A++ | See 4.5.3 Performance |
| Scalability | Growing response time /thread number ratio | 1,15 | A++ | See 4.5.1  Scalability |
| Stability | Stability | Memory and/or CPU are significantly higher at the end of the test, but it doesn’t seem to exist a leak. | A | See 4.5.2  Stability |

## **A3.5 Orion Context Broker non functional labelling**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Measure name** | **Gathered Value** | **Label (A+++/D)** | **Notes/Comments/ Objective Evidences (OE)** |
| Scalability | Scalability | 1,02 ratio | A+++ |  |
| Performance | Performance | 1900 updates/sec | A+++ |  |
| Stability | Stability | Leak avoidable | B |  |

## **A3.6 Proton non functional labelling (results from 5.1)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Measure name** | **Gathered Value** | **Label (A+++/D)** | **Notes/Comments/ Objective Evidences (OE)** |
| Scalability | Scalability | 1,03 ratio | A+++ |  |
| Performance | Performance | 500 events/sec | A+++ |  |
| Stability | Stability | Memory leak | C | System fall after 2 hours |

## **A3.7 Kurento non functional labelling**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Measure name** | **Gathered Value** | **Label (A+++/D)** | **Notes/Comments/ Objective Evidences (OE)** |
| Scalability | Scalability | 1,65 ratio | B |  |
| Performance | Performance | 140 Kbps \* user | C |  |
| Stability | Stability | Memory and CPU a little higher at the end of the test | A+ |  |

## **A3.8 IDAS non functional labelling (results from 5.3)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Measure name** | **Gathered Value** | **Label (A+++/D)** | **Notes/Comments/ Objective Evidences (OE)** |
| Scalability | Scalability | 1,03 ratio | A+++ |  |
| Performance | Performance | 140 updates/sec | B | As the updates where structural updates, and not a measure, they count as 2 attribute updates |
| Stability | Stability | Memory usage is a bit higher at the end | A++ |  |

## **A3.9 Cepheus non functional labelling**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Measure name** | **Gathered Value** | **Label (A+++/D)** | **Notes/Comments/ Objective Evidences (OE)** |
| Scalability | Scalability | 1,02 ratio | A+++ |  |
| Performance | Performance | 200 updates/sec | A+++ | After ponderation: 1639 updates/sec |
| Stability | Stability | Leak avoidable | B |  |

## **A3.10 Bosun non functional labelling**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Attribute** | **Measure name** | **Gathered Value** | **Label (A+++/D)** | **Notes/Comments/ Objective Evidences (OE)** |
| Scalability | Scalability | 1,13 ratio | A++ |  |
| Performance | Performance | 26 responses/sec | A+ | Complexity about 20 attribute updates = 520 attribute updates |
| Stability | Stability | Memory usage is a bit higher at the end | A++ |  |

1. https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Working\_with\_the\_FIWARE\_Academy#Course\_Evaluation\_for\_Efficient\_Training [↑](#footnote-ref-1)
2. Test oracle is a source to determine expected results to compare with the actual result of the software under test. [↑](#footnote-ref-2)
3. [www.computer.org/sevocab](http://www.computer.org/sevocab) [↑](#footnote-ref-3)