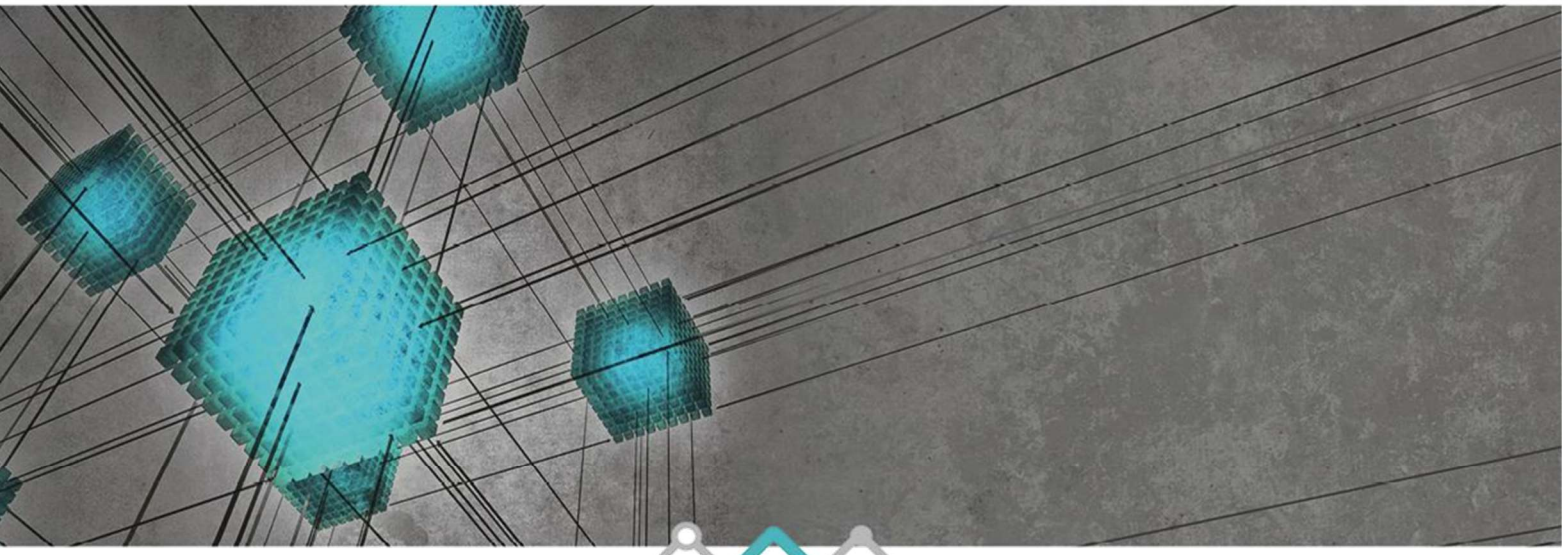


interiot

INTEROPERABILITY
OF HETEREOGENEUS
IOT PLATFORMS.

D6.3

Site Acceptance Test Plan

Version: 1.1 - Final

September 2018

INTER-IoT

INTER-IoT aim is to design, implement and test a framework that will allow interoperability among different Internet of Things (IoT) platforms.

Most current existing IoT developments are based on “closed-loop” concepts, focusing on a specific purpose and being isolated from the rest of the world. Integration between heterogeneous elements is usually done at device or network level, and is just limited to data gathering. Our belief is that a multi-layered approach integrating different IoT devices, networks, platforms, services and applications will allow a global continuum of data, infrastructures and services that will enhance different IoT scenarios. Moreover, reuse and integration of existing and future IoT systems will be facilitated, creating a de facto global ecosystem of interoperable IoT platforms.

In the absence of global IoT standards, the INTER-IoT results will allow any company to design and develop new IoT devices or services, leveraging on the existing ecosystem, and bring them to market as fast as possible.

INTER-IoT has been financed by the Horizon 2020 initiative of the European Commission, contract 687283.

INTER-IoT

Site Acceptance Test Plan

Version: 1.1

Security: Confidential

26 September 2018

The INTER-IoT project has been financed by the Horizon 2020 initiative of the European Commission, contract 687283

Disclaimer

Executive Summary

This document describes the Site Acceptance Test (SAT) plans of the INTER-IoT project.

During development units and components are tested and validated by the developers. This applies to the IoT framework components as well as to the project specific units and components. When all the components needed for the project are complete, tested and validated they are integrated into the system as defined for this project.

The system has already undergone the Factory Acceptance Test to test the readiness of the system. This is done in a LAB setup which approached the actual field deployment as much as possible. As the SAT has been successfully executed and has proved the system in a “factory” environment the system can advance to field integration and undergo the Site Acceptance Test (SAT).

This document describes all aspects of the SAT, from defining the versions of the used components and deliverable checklist up to, test setup, tooling, test description, etc. to be able to test the readiness of the system under test.

Finally, the structure of this document is divided into the following sections:

- Section A: Introduction
- Section B: Test strategy and approach
- Section I: SAT INTER-LogP
- Section II: SAT INTER-Health
- Section III: Opencall SATs
- Section IV: Open Call Third Parties Evaluation
- Section V: Conclusions

List of Authors

Organisation	Authors	Main organisations' contributions
Neways	Dennis Engbers	Document template, structure, Executive summary, Introduction, test setup, chapter templates and edit of the final document
Neways	Johan Schabbink	SAT documents review
Neways	Arnoud Groote-Venema	SAT documents review
VPF	Pablo Giménez	SAT for INTER-LogP
UPV-SABIEN	Álvaro Fides Valero	SAT for INTER-Health
CEA	Jander Nascimento	SAT for SensiNact
VUB	Kris Steenhaut,	SAT for INTER-OM2M
VUB	Benjamin Sartori	SAT for INTER-OM2M
UPF	Toni Adame	INTER-Hare System description, Deliverables, version overview, Requirements, scenario, Use cases to test, Test environment, Test description
UPF	Albert Bel	INTER-Hare Ethics
VPF	Pablo Giménez	INTER-Hare General revision
Nemergent Solutions	Iñigo Ruiz	Created SAT for Mission Critical operations based on IoT analytics
Nemergent Solutions	Jose Oscar Fajardo	Created SAT for Mission Critical operations based on IoT analytics
University of Twente	João Moreira	SAT for Early Warning System
Irideon	Bastian Faulhaber	SAT for Senshook
INFOLYSiS	Vaios Koumaras	SAT for SOFOS
INFOLYSiS	Christos Sakkas	SAT for SOFOS
AUEB	Nikos Fotiou	SAT for INTER-ACHILLES
AUEB	George C. Polyzos	SAT for INTER-ACHILLES
TU Wien	Hong-Linh Truong	INTER-Hinc - All parts of the report, document structure and tests
TU Wien	Bunjamin Memishi	INTER-Hinc - Drafted initial document structure
TU Wien	Lingfan Gao	INTER-Hinc - Support developing tests/services
TU Wien	Michael Hammerer	INTER-Hinc - Support developing tests/services
CNR-ITIA	Gianfranco Modoni	SAT for Semantic middleware
CNR-ITIA	Enrico Caldarola	SAT for Semantic middleware
AvailabilityPlus / SecurIoTy	Guenther Hoffmann	SAT for SecurIoTy
E3tcity	Javier Escalera Casillas	SAT for E3tcity

Change control datasheet

Version	Changes	Chapters	Pages
0.1	Creation, structure, Executive summary, Introduction, test setup, chapter templates	All	31
0.2	Added initial SAT contributions	List of Authors, Acronyms, Chapter 3	266
0.3	Added last two SAT contributions	3.3.2 and 3.3.5	301
0.4	Added evaluation and conclusions chapters and moved the document to review state	4 and 5	303
0.5	Added test remarks to INTER-Health	3.2.2 and 3.2.3	305
0.6	Added additional tests to INTER-Logp	3.1.12 and 3.1.13	306
0.7	Rework based on review by Noatum	3.1 and 5	306
1.0	Released document	All	306
1.1	Minor consistency and format changes after final review	All	305

Contents

List of Figures

List of Tables

Acronyms

AIOTI	Alliance for Internet of Things Innovation
API	Application Programming Interface
BDD	Behaviour Driven Development
CCB	Change Control Board
CNR-ITIA	National Research Council - Institute of Industrial Technologies and Automation
CSV	Comma-Separated Values
DMZ	Demilitarized zone
EC	European Commission
ESB	Enterprise service bus
FAT	Factory Acceptance Test
GCP	Google Cloud Platform
GDPR	General Data Protection Regulation
GOIoTP	Generic Ontology for IoT Platforms
ICT	Information and Communication Technology
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IoT-EPI	IoT European Platform Initiative
IPR	Intellectual property rights
JSON	JavaScript Object Notation
JSONoWS	JSON over WebSockets
LAB setup	Test setup in a controlled environment at the developer site
LPLAN	Low Power Local Area Network
LPWAN	Low Power Wide Area Network
LTL	Less than Truck Load
MC	Mission Critical
MC-IoT	Mission Critical Internet of Things
MCPTT	Mission Critical Push To Talk
MEP	MQTT Event publisher
MiCrOBIoTa	Mission Critical operations based on IoT analytics
MQTT	MQ Telemetry Transport
MS	Microsoft
MSK	Master Secret Key

MSSB	MS Service Broker
MW2MW	Middleware to Middleware
NFV	Network Function Virtualization
oBIX	Open Building Information eXchange
OM2M	open source implementation of oneM2M in Eclipse
OS	Operating System
OSGi	Open Services Gateway initiative
PDR	Packet Delivery Ratio
PIR	Passive Infrared Sensor
PMR	Professional Mobile Radio
PRM	Power Regulation Mechanism
REST	Representational State Transfer
SAT	Site Acceptance Test
SDN	Software Defined Networking
SSBEAS	MS SQL Service Broker External Activator Service
STA	Station
TDD	Test Driven Development
TDMA	Time Division Multiple Access
TED	Transducer Electronic Data Sheet
TRL	Technology Readiness Level
UPF	Universitat Pompeu Fabra
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VPF	Valenciaport Foundation
XML	eXtended Markup Language

1 Introduction

INTER-IoT project is aiming at the design, implementation and experimentation of an open cross-layer framework, an associated methodology and tools to enable voluntary interoperability among heterogeneous Internet of Things (IoT) platforms. The proposal will allow effective and efficient development of adaptive, smart IoT applications and services, atop different heterogeneous IoT platforms, spanning single and/or multiple application domains.
Most current existing sensor networks and IoT device deployments work as independent entities of homogenous elements that serve a specific purpose, and are isolated from “the rest of the world”. In a few cases where heterogeneous elements are integrated, this is done either at device or network level, and focused mostly on unidirectional gathering of information. A multi-layered approach to integrating heterogeneous IoT devices, networks, platforms, services and applications will allow heterogeneous elements to cooperate seamlessly to share data, infrastructures and services as in a homogenous scenario.

This document describes the Site Acceptance Test plan for INTER-IoT which is part of the experimentation step.

One of the main goals of INTER-IoT is to overcome fragmentation caused by typical IoT platforms being oriented to a specific solution, stakeholder and application domain. The cross-domain use case will show how verticality is avoided in INTER-IoT. The rationale behind this use case is that future IoT applications will not aim at a single application domain but multiple domains in which devices, networks, platforms, services or generated data will interact.
The scenarios defined in the cross application domain use case will integrate platforms from the two application domains in consideration, and also from different application domains (e.g. smart grid or smart cities). This use case will prove the extendibility of the project outcomes, achieving interoperability between IoT platforms from different application domains. Several scenarios have been foreseen in which IoT platforms from different application domains may be required to interoperate, e.g. logistics and health monitoring of transport workers for labour risk prevention, however new cross domain scenarios will be defined during the execution of the project and after the resolution of the Open Call, including e.g. road IoT ecosystems; supply chains or emergency response services IoT ecosystems used in fire brigades, ambulances or security forces.

According to the Grant Agreement the field trials will be successful once the following conditions are met:
“Trials of INTER-IoT concept, with involvement of 400 smart objects in the logistics use case and 200 subjects (with wearable devices) in the m-Health use case, and ~500 IoT units in the cross domain use case. Extensive testing of results of application of the INTER-IoT framework to instantiate multi-IoT-platform systems in real-world scenarios, validated by the corresponding stakeholders.”

2 Test strategy and approach

This Site Acceptance Test is performed to test and prove the system is operational and functional and complies with the defined requirements. The SAT takes place after integration at the customer site. During Site Acceptance Testing the solution is tested on: Integration, Performance, conformance to specifications and User acceptance testing. The SAT Document describes the Site Acceptance Testing plan and describes or points to previously defined test plans, use cases and test scenarios used during testing. The test outcomes can either be placed in the SAT document itself or a separate test report can be created

2.1 Testing strategy

This document will describe the system, test setup, tooling, test strategy, test activities and test results for the integrated setup. During these tests the system integration and functionality will be tested and proven. For this test, the following stakeholders shall be present:

- Project managers (From manufacturer and customer)
- Key engineering personnel (System Architect, Lead Engineer, Integrator)
- Operators
- Maintenance personnel

During the SAT test the actual deployed system is tested and proven. The SAT follows the same principles as the FAT but describes and tests the system integrated in the customer systems. During testing the result should be written in the SAT document which should be signed off by all the attendees at the end of the test. Signing the SAT is the actual acceptance of the system by the customer.

2.2 Entrance criteria

To start a SAT for this project the following deliverables should be present/ready:

- Integration of the tested and validated system component into a complete system
- Validation and Test reports of the system components
- Executed FAT test report(s)
- A reviewed and approved SAT document
- System test setup (as much actual hardware as possible)
- Test applications and tooling (e.g. for performance testing)

2.2.1 Integration of the tested and validated system components

The system components that make up this system should all be tested and validated. The units that make up the components should be unit tested and validated on interface level, normal use and boundary checking, error handling, performance, etc. After integration into components the components should be integration tested the same way on interface level.

This applies for the pilot specific components as well as for the used IoT components. For each component a test/validation report for the used version should be available stating that the component has been tested and validated and passed this test.

2.2.2 Validation and Test reports of the system components

The test and validation reports of the system components used in this project should provide an overview of the tests done on the used system components and the outcome of these

tests. Each component should have passed the tests and validation before used in this SAT. The version in these reports shall match the version used in the system release to be SAT tested. An overview of the used version can be found in Table 78: Component version overview.

2.2.3 Executed FAT test report(s)

The FAT report or reports of the executed FAT tests which prove that the system has been tested and validated before integration on site.

2.2.4 SAT document

A printed version of this document which should be checked before start of the test. Any issue found should be manually corrected. The outcome and remarks of each test as well as the final outcome should be written in the copy of this document during testing and be signed at the end of the SAT. The signed copy of this document will serve as an acceptance on the system to start field integration.

2.2.5 System test setup, Test applications and tooling

The test setup as described in this document should be present and checked for completeness. See 0 Test environment for the system setup.

2.3 Acceptance Criteria

The SAT acceptance criteria is a signed copy of this SAT document as this contains all the needed deliverables and tests to complete the Site acceptance testing. After a successful SAT the system can be integrated in the field and proceed to SAT testing.

2.4 Testing types

The SAT document will define the testing types per project in detail. The following list provides an example of testing types one could think of:

- manual data load
- interface using scripted data
- interface bounds checks
- converted data load
- converted data inspection
- backup and recovery
- database auditing
- data archival
- security
- locking
- batch response time
- online response time
- network stress
- stress testing
- security
- live data
- live environment
- error handling

2.5 Suspension and resumption criteria

When one of the entrance criteria is not met at the start of the test the complete or part of the tests will be suspended until it/they are met.

When during testing one of the entrance criteria's is found to be unsatisfactory parts of the tests or the complete test can be suspended until it is met. In most cases though the test will be executed completely to prove the rest of the system.

When a test is suspended or after execution is not accepted the found issues shall be solved and a new SAT test shall be performed. The new SAT will then again run all tests to confirm the issues are solved and no new issues have been introduced.

2.6 Change Control Board

A Change Control Board (CCB) will be defined for each project.

The change control board will consist of the following persons:

- Carlos Palau (UPV)
- Eneko Olivares (UPV)
- Flavio Fuart (XLAB)
- Johan Schabbink (NEWAYS)
- Dennis Engbers (NEWAYS)
- Pablo Giménez (VPF)
- Gema Ibáñez (SABIEN)

2.7 Defect Reporting

Please see D6.1 System Integration Plan for defect reporting.

3 Factory Acceptance Test

3.1 INTER-LogP SAT

The goal of INTER-LogP pilot is to demonstrate the need for a system that allows the exchange of data and messages among the different actors of the port community. In this case, as can be seen in Figure 1, there are three main actors: the port, the terminal and the haulier company. INTER-IoT has to provide interoperability between the IoT platforms of the port and the terminal, and give access to devices from other companies, like trucks.

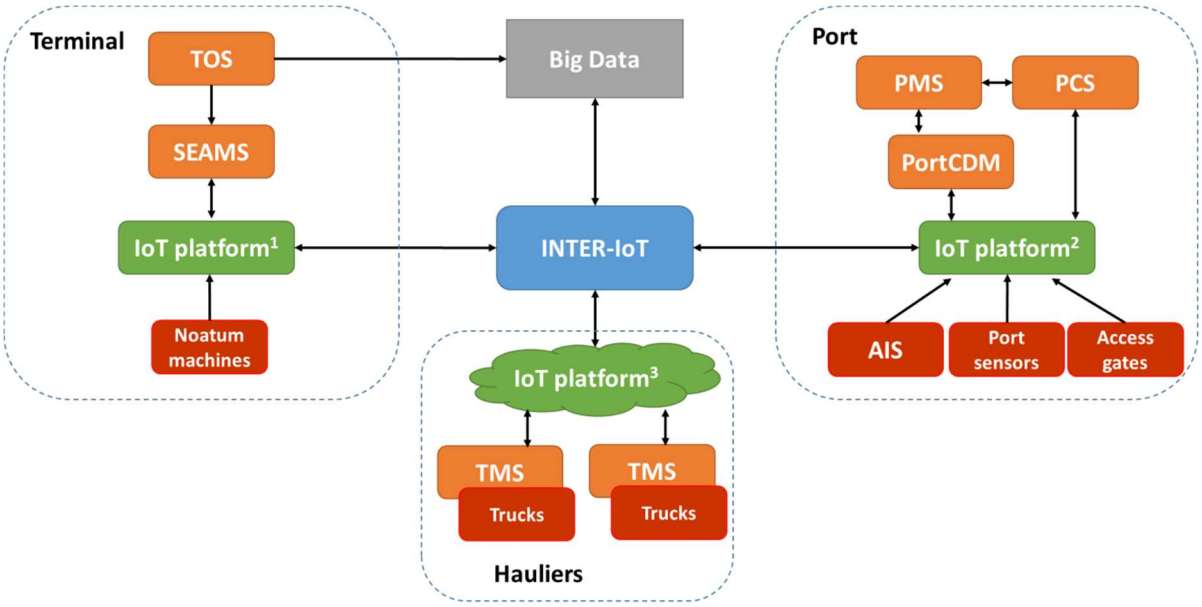


Figure 1: INTER-LogP high-level design.

Both the port and the terminal have a large number of sensors and devices that produce large amounts of data that can be interesting for other entities. Similarly, they need data from other companies to provide a better service to their clients.

3.1.1 Port authority

The port authority has several sensors distributed throughout the port that provide data for management and operation. Most of this data is confidential, but some can be shared, adding value to other companies.

The architecture for providing interoperability with the existing port infrastructure can be seen in Figure 2. Currently, the port authority has an industrial control bus infrastructure using SCADA and a common database to store data coming from different sources in an isolated way (in red). The new platform uses WSO2 to provide an IoT architecture in two ways: provide data in near real time through the Message broker and historic data through the Data services server and an Enterprise service bus (ESB).

Because the port has its own platform, the integration with the INTER-IoT is done through the INTER-MW. It needs a bridge in the middleware layer in order to interoperate with other platforms.

Additionally, new devices are also introduced (i.e. PIR sensors, dynamic lighting controllers) and interoperate through the gateway created in INTER-IoT.

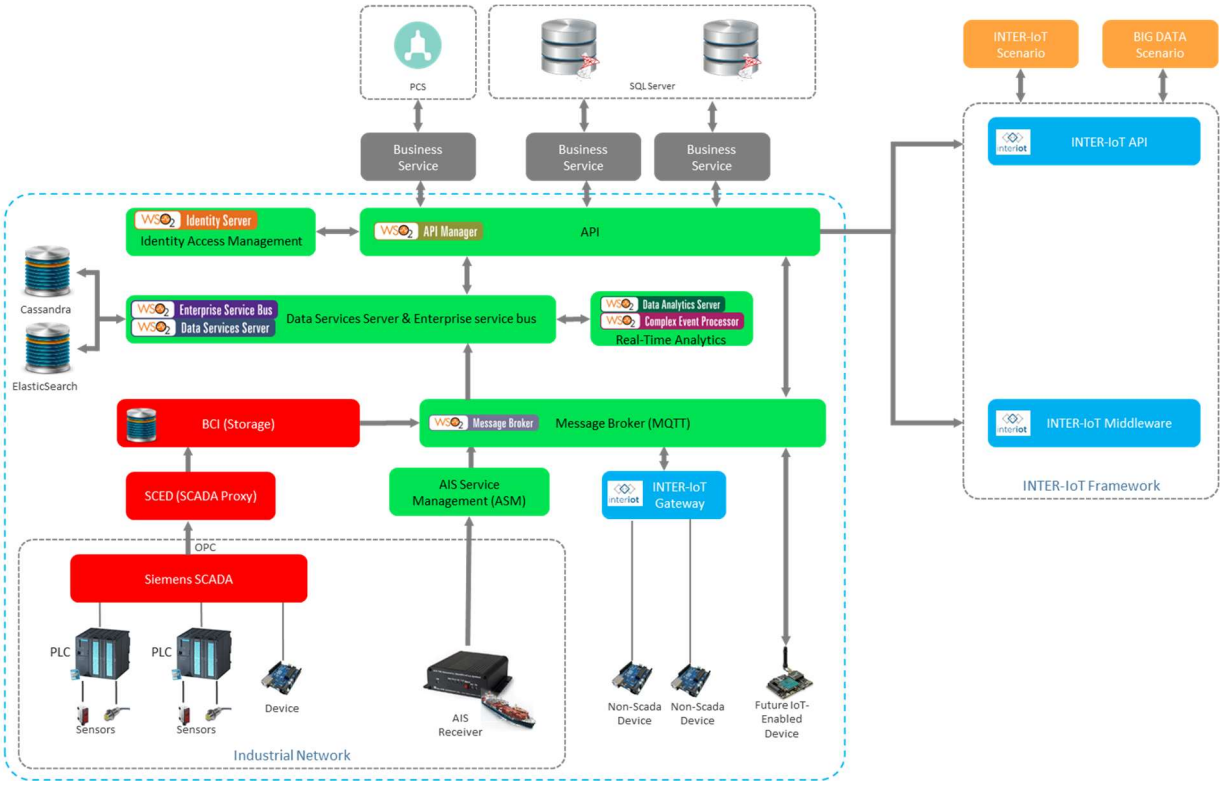


Figure 2: Port IoT platform and integration

3.1.2 Port data ontology

There are three different types of data from the port authority domain included in the INTER-IoT pilots: gate access, environmental, and lighting. For each one of them, we have defined a set of messages and included in the port ontology. These messages can be found below.

Gate access


```
{"valenciaPortData":{
        "entryVehicles": [
                {
                "accessId": 7089855,
                "date": "2018-01-
07T07:01:04.000Z",
                "lane": 7,
                "plate": "6345 GXZ",
                "reliability": 95,
                "containerNum1":
"MSKU9371630",
                "containerReliability1": 100,
                "containerNum2": "",
                "containerReliability2": 0
                }
]}}
```

```
{"valenciaPortData":{
        "exitVehicles":[
                {
                "accessId": 2156762,
                "lane": 1,
                "lprOk": 1,
                "lprDate": "2017-11-01T02:00:17. +02:00",
                "lprPlate": "7252EKT",
                "lprReliability": 99,
                "lprAlarmOk": 1,
                "lprAlarmDate": "2017-11-01T02:00:17. +02:00",
                "lprAlarmPlate": "7252EKT",
                "lprAlarmReliability": 99,
                "quercusLprOk": 1,
                "quercusLprDate": "2017-11-01T02:00:17. +02:00",
                "quercusLprPlate": "7252EKT",
                "quercusLprReliability": 99,
                "quercusLprAlarmOk": 1,
                "quercusLprAlarmDate": "2017-11-01T02:00:17.
+02:00",
                "quercusLprAlarmPlate": "7252EKT",
                "quercusLprAlarmReliability": 99,
                "gateOpen": "2017-11-01T02:00:17. +02:00",
                "spire": "2017-11-01T02:00:17. +02:00",
                "ocrOk": 1,
                "ocrDate": "2017-11-01T02:00:17. +02:00",
                "ocrContainerNum1": "PCIU9985660",
                "ocrContainerReliability1": 99,
                "ocrContainerNum2": "PCIU9985660",
```

```
                                        "ocrContainerReliability2": 99,
                                        "ocrAlarmOk": 1,
                                        "ocrAlarmDate": "2017-11-01T02:00:17. +02:00",
                                        "ocrAlarmContainerNum1": "PCIU9985660",
                                        "ocrAlarmContainerReliability1": 99,
                                        "ocrAlarmContainerNum2": "PCIU9985660",
                                        "ocrAlarmContainerReliability2": 99
                                        }
]}}
```


Environmental


```
{"valenciaPortData":{
        "meteoStations": [
                {
                "meteoStationId": 2,
                "name": "P.Felipe",
                "portId": 1,
                "active": 1,
                "location": "Muelle Felipe
Valencia",
                "latitude": 26.94442,
                "longitude": 19.29351
                }
]}}
```

```
{"valenciaPortData":{
        "weatherMeasurements": [
                {
                "measurementId": 3181710,
                "meteoStationId": 9,
                "date": "2018-02-15T09:50:01.000Z",
                "windSpeed": 2.797972,
                "windDirection": 177.4603,
                "averageTemperature": 13.23898,
                "humidity": 78.36698,
                "precipitation": 4.899998,
                "seaTemperature": 0.0,
                "solarRadiation": 0.0,
                "pressure": 1025.552
                }
]}}
```

```
{"valenciaPortData":{
        "soundMeters":[
                {
                "cabinId": 1,
                "soundMeterId": "3D0CF7BE-457A",
                "description": "Control Túnel",
                "portId": 1,
                "latitude": 26.027065,
                "longitude": 18.170195
                }
]}}
```

```
{"valenciaPortData":{
        "soundMeasurements":[
                {
                "measurementId": 3178660,
                "cabinId": 1,
                "startDate": "2017-11-01T02:00:17.
+02:00",
                "endDate": "2017-11-01T02:00:17.
+02:00",
                "maxSoundLevel": 706,
                "averageSoundLevel": 552,
                "minSoundLevel": 505
                }
]}}
```

```
{"valenciaPortData":{
        "emissionCabins":[
                {
                "emissionCabinId": 1,
                "name": "Cabina VR-004",
                "portId": 1,
                "description": "Caseta Ecoport",
                "latitude": 26.027065,
                "longitude": 18.170195
                }
]}}
```

```
{"valenciaPortData":{
        "emissionMeasurements":[
                {
                "measurementId": 3178660,
                "emissionCabinId": 1,
                "date": "2017-11-01T02:00:17. +02:00",
                "co": 0.2,
                "no": 12,
                "no2": 10,
                "nox": 18,
                "so2": 10,
                "particlesConcentration": 7.9
                }
]}}
```

Lighting


```
{"valenciaPortData":{
        "lights":[
                {
                "lightId": "L01",
                "lampPostId": "B01",
                "description": "",
                "latitude": 26.94442,
                "longitude": 19.29351,
                "powerState": 1,
                "isConnected": 1,
                "consumption": 7.9,
                "dimmer": 7.9,
                "lastUpdate": "2018-03-14T11:00:17. +02:00"
                }
```

```
{"valenciaPortData":{
        "presenceSensors":[
                {
                "presenceSensorId": "P01",
                "type": "PIR",
                "description": "",
                "latitude": 26.94442,
                "longitude": 19.29351
                }
]}}
```

```
{"valenciaPortData":{
        "presenceSensorObservations":[
                {
                "observationId": 1,
                "presenceSensorId": "P01",
                "detectionDate": "2017-11-01T02:00:17. +02:00"
                }
]}}
```


3.1.3 Port data services

The data coming from the port authority IoT platform is accessible in two different ways, depending on the type of data.

To access real-time data, a platform or application can subscribe to different topics at a broker (always through the MW), as long as they have access. An example of a topic for accessing meteorological data is *env/weather/stations/1.*

In the case of historical data, there are different APIs providing access to the data stored in the database. The port IoT platform has an API manager where all the published APIs can be found, as seen in Figure 3. Figure 4 shows an example of one API definition.

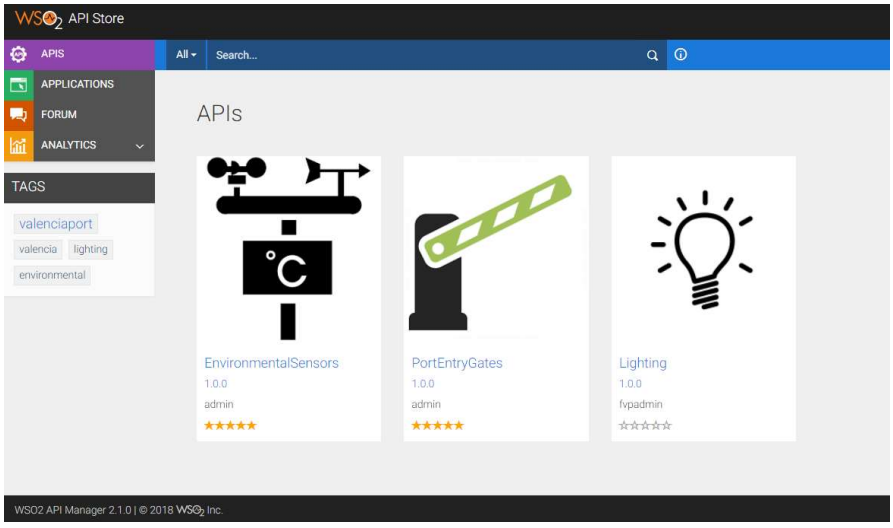


Figure 3: Port API manager interface

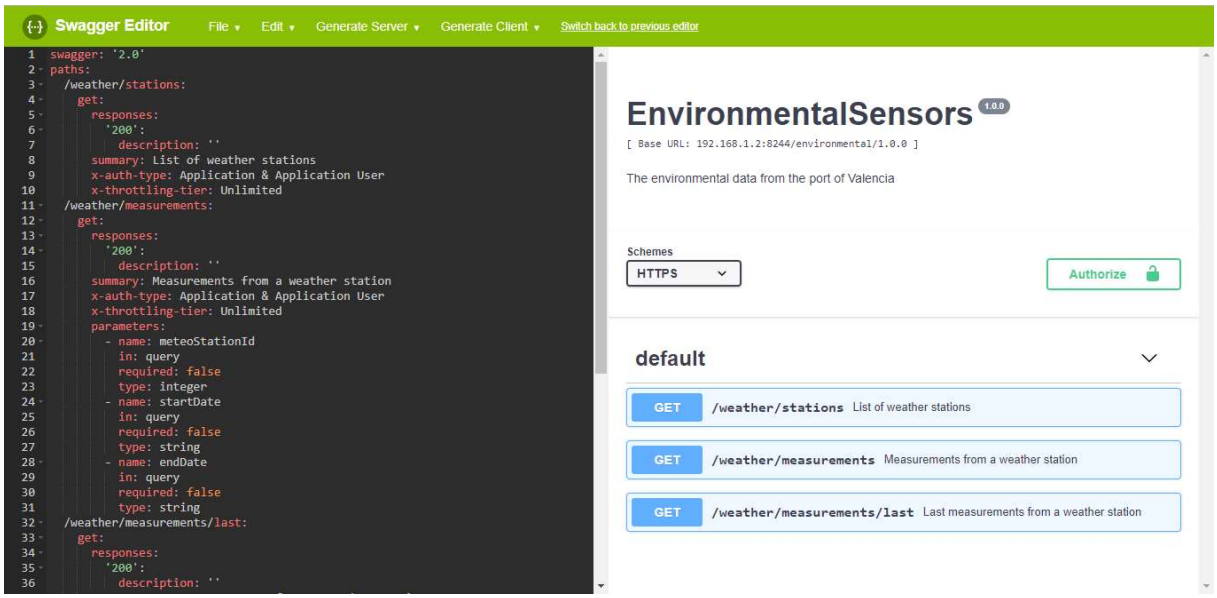


Figure 4: Environmental API definition

3.1.4 Port equipment deployed

The INTER-LogP pilots will reuse the existing systems and sensors in the port and the Noatum terminal as much as possible. However, the pilots required the deployment of some additional equipment.

In the case of the port, the main equipment added is a server hosting the port IoT. This server has access to the different port legacy systems needed to collect the data.

However, in the case of the lighting pilot new equipment had to be deployed in order to monitor the two areas involved in the pilot: the rail yard in Noatum and the railway switchgear area, which are shown in the Figure 5

Figure 5: Lighting pilot monitored areas. a) rail yard b) railway switchgear area

The new equipment distributed between the port and the terminal is:

- 28 light bulbs
- 34 controllers for lights and PIRs
- 8 PIRs
- 4 routers
- 3 WiFi access points
- 3 INTER-IoT gateways

In the following figures it is shown how this equipment is connected and controlled through the different IoT platforms. Namely, Figure 6 shows the layout connectivity, and Figure 7 the installation map.

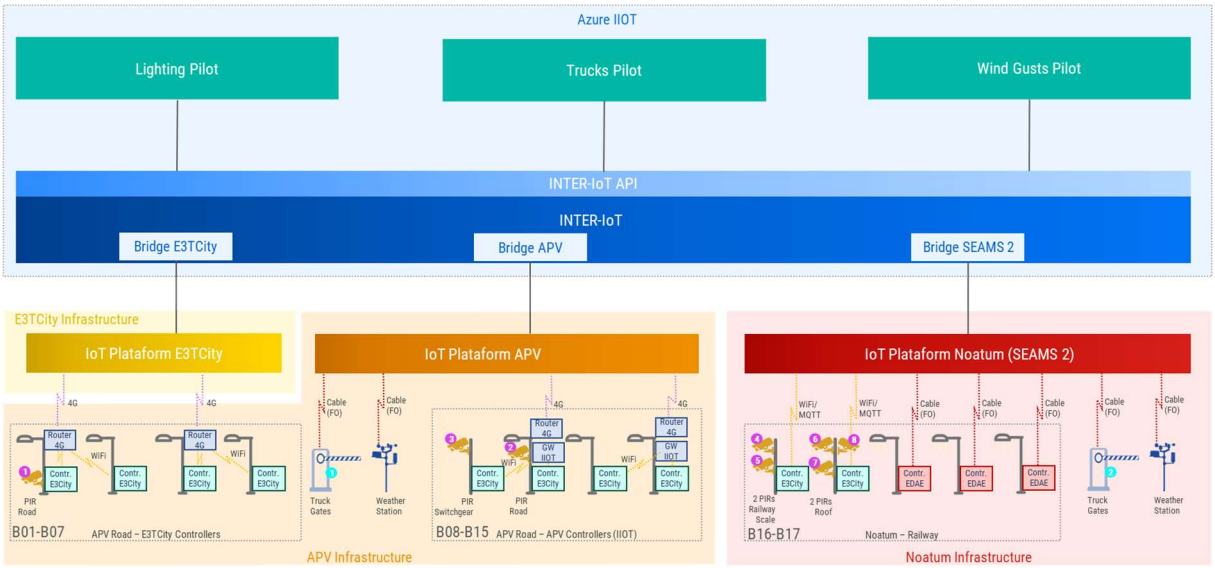


Figure 6: Lighting pilot connectivity layout

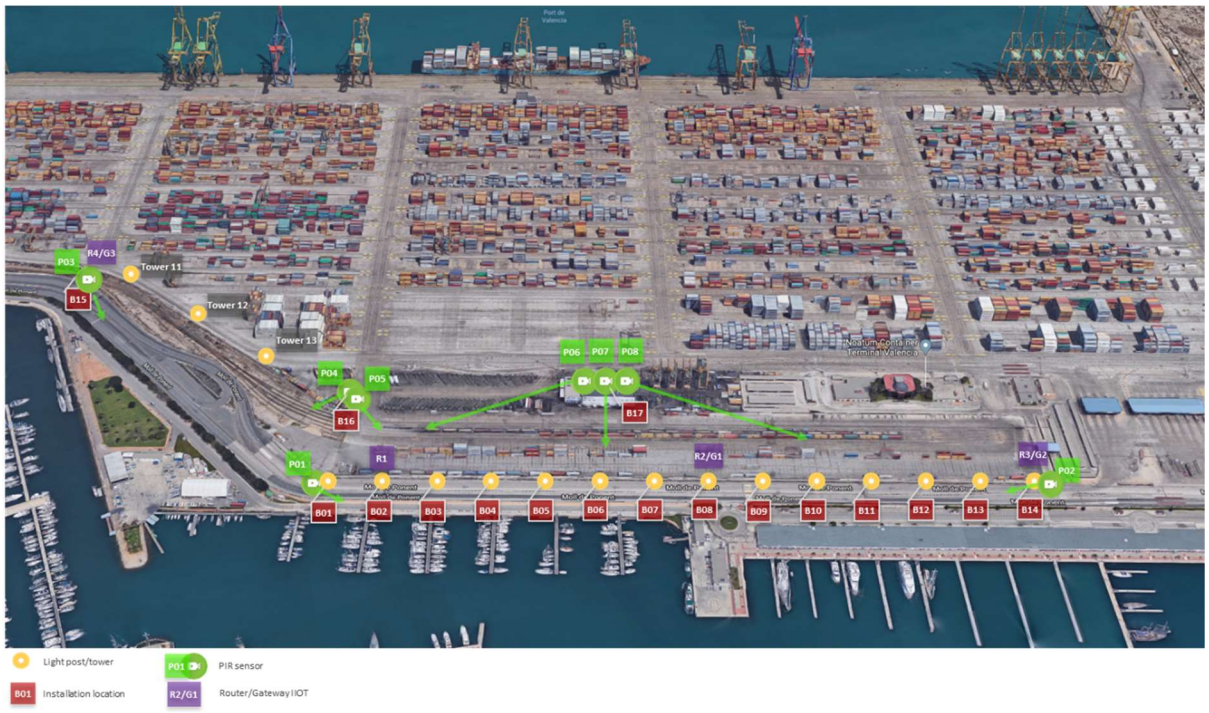


Figure 7: Lighting pilot installation map

3.1.5 Container terminal

The correct management of resources in a container terminal implies the monitoring of all the machinery. For that reason, in the Noatum terminal, each machine (vehicles, cranes, etc.) provides a massive amount of data, about to 80 sensors per machine reporting on a secondly basis. In total, there are around 300 monitored devices, including both machines and the dynamic lighting on lamp posts.

As can be seen in Figure 8, there are four different types of data sources reporting data from the machinery to the IoT Platform, legacy sensors/systems and IoT devices. Legacy sensors are read once per second and inserted in the IoT Platform. New IoT devices are configured to send their data directly through a MQTT bus or REST interfaces on real-time. In addition, the data is stored in a non-relational database, what speeds up the access to information.

As in the port, the Noatum container terminal IoT platform is to be integrated with INTER-IoT through the middleware layer and the API layer.

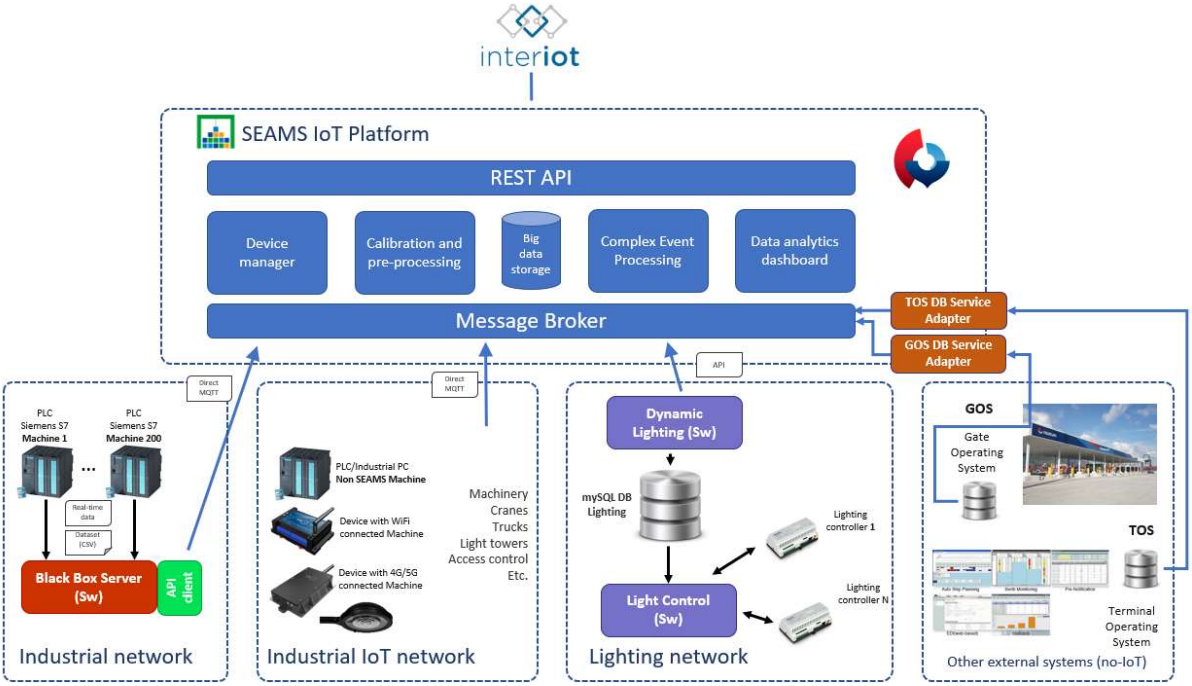


Figure 8: Terminal IoT platform and integration

The container terminal has its own server with its own IoT platform. The main interest of the container terminal is to know the estimated time of arrival of the trucks to the terminal to manage its resources adequately. Furthermore, the terminal grants access to other companies to some of their own data, such as the entry and exit of trucks by their access.

Example of a data from a terminal tractor (TT) as it is present in the SEAMS IoT Platform (which uses several components of WSO2):


```
{"id":169,"name":"TT0063","type":"TT_G5","family":"TT","msgTimestamp":"29/05/201
8
14:29:42","version":"1.0","attributes":{"GpsX":{"value":39.4341354370117},"GpsY"
:{"value":-
0.329963326454163},"GpsHDOP":{"value":1.06619203090668},"GpsSQ":{"value":0},"Gps
DU":{"value":2},"Gps":{"value":7},"GpsV":{"value":0},"Dist":{"value":0.0},"GpsR"
:{"value":296.85},"Nmov":{"value":true},"PlcOn":{"value":true},"DrOn":{"value":t
rue},"On":{"value":true},"Off":{"value":false},"Alarm":{"value":false},"WHours":
{"value":30451},"FRate":{"value":0.0},"PPos1":{"value":false},"PPos2":{"value":f
alse},"Reserva12":{"value":false},"Reserva11":{"value":false},"Reserva10":{"valu
e":false},"Reserva9":{"value":false},"Reserva8":{"value":false},"Reserva7":{"val
ue":false},"Reserva6":{"value":0},"Reserva5":{"value":0},"Reserva4":{"value":0},
"Reserva2":{"value":0.0},"Reserva1":{"value":0.0}},"metadata":[{"timestamp_BB":"
2018-05-29T14:29:43.161Z"},{"guid":"42491904-cfad-46ca-830c-
94060213c61a"},{"threadId":29}]}
```

Or, in the case of a Sea-to-Shore (STS) crane:


```
{"id":10,"name":"STS015","type":"STS_G7","family":"STS","msgTimestamp":"29/05/20
18
14:50:31","version":"1.0","attributes":{"GpsX":{"value":39.432975769043},"GpsY":
{"value":-
0.323958337306976},"GpsHDOP":{"value":1.066192},"GpsSQ":{"value":2},"GpsDU":{"va
lue":2},"Gps":{"value":8},"GpsV":{"value":0},"Dist":{"value":0.0},"GpsR":{"value
":292.46},"TwO":{"value":false},"TwC":{"value":true},"TrLane":{"value":true},"Nm
ov":{"value":false},"Sp20":{"value":false},"Sp40":{"value":true},"SpTwin20":{"va
lue":false},"SpTwin40":{"value":false},"SpTwin4x20":{"value":false},"FlippUp":{"
value":true},"FlippDown":{"value":false},"NetLoad":{"value":0.0},"ETrolley":{"va
lue":-
13525},"JoyTr":{"value":0},"TrSea":{"value":false},"TrLand":{"value":false},"TrP
ark":{"value":false},"EHoist":{"value":4635},"JoyHo":{"value":-
67},"HoistU":{"value":false},"HoistD":{"value":true},"BoDls":{"value":true},"BoU
lt":{"value":false},"BoDw":{"value":false},"BoUp":{"value":false},"Bo45":{"value
":false},"JoyGr":{"value":0},"GantR":{"value":false},"GantL":{"value":false},"Pl
cOn":{"value":true},"DrOn":{"value":true},"SprOn":{"value":true},"PrtOn":{"value
":false},"CabSen":{"value":false},"CtCab":{"value":true},"CtGrn":{"value":false}
,"CtBoo":{"value":false},"CtEro":{"value":false},"MMant":{"value":false},"HookMo
":{"value":false},"ManMo":{"value":false},"On":{"value":true},"EcoON":{"value":f
alse},"StbyON":{"value":false},"Off":{"value":false},"OnR":{"value":false},"EcoO
nR":{"value":false},"StbyOnR":{"value":false},"OffR":{"value":false},"Fault":{"v
alue":false},"Alarm":{"value":false},"Warning":{"value":false},"WHours":{"value"
:52527},"PwCons":{"value":-
62},"PwGen":{"value":0},"WindS":{"value":25.53125},"FRate":{"value":0.0},"HBrake
":{"value":false},"Oper":{"value":0},"Reserva12":{"value":false},"Reserva11":{"v
alue":false},"Reserva10":{"value":false},"Reserva9":{"value":false},"Reserva8":{
"value":false},"Reserva7":{"value":false},"Reserva6":{"value":0},"Reserva5":{"va
lue":0},"Reserva4":{"value":0},"Reserva2":{"value":0.0},"Reserva1":{"value":0.0}
},"metadata":[{"timestamp_BB":"2018-05-29T14:50:31.696Z"},{"guid":"6f63472a-
0c89-4ce5-8a77-6f5f4625b172"},{"threadId":13}]}
```


The data as it is delivered by the REST API is as follows

```
curl -X GET --header 'Accept: application/json' --header 'Content-Type: application/json'
'http://192.168.11.9:8081/seams2/SEAMS2-Platform/1.0.0/devices/context/STS015'
```


```
    {
        "name": "GantL",
        "value": "false",
        "metadata": null
    },
    {
        "name": "WindS",
        "value": "17.65625",
        "metadata": null
    },
    {
        "name": "PosArea",
        "value": "",
        "metadata": null
    },
    {
        "name": "PosZone",
        "value": "DOC",
        "metadata": null
    },
  […]
```

3.1.6 Haulier Company

Haulier companies have large fleets of trucks accessing the port daily. The trucks involved in the pilot will have a mobile app (MyDriving[1]) installed in a mobile or a tablet that acts as a bridge between the vehicle and the IoT platform of the company in Azure. All the devices in the truck and the driver send the data to the IoT platform through the mobile app via Bluetooth.

The haulier company has an Azure IoT platform in the cloud where their trucks send all their data. These data will be accessible to other companies as long as they are authorized and certain conditions are met, such as being within the port area.

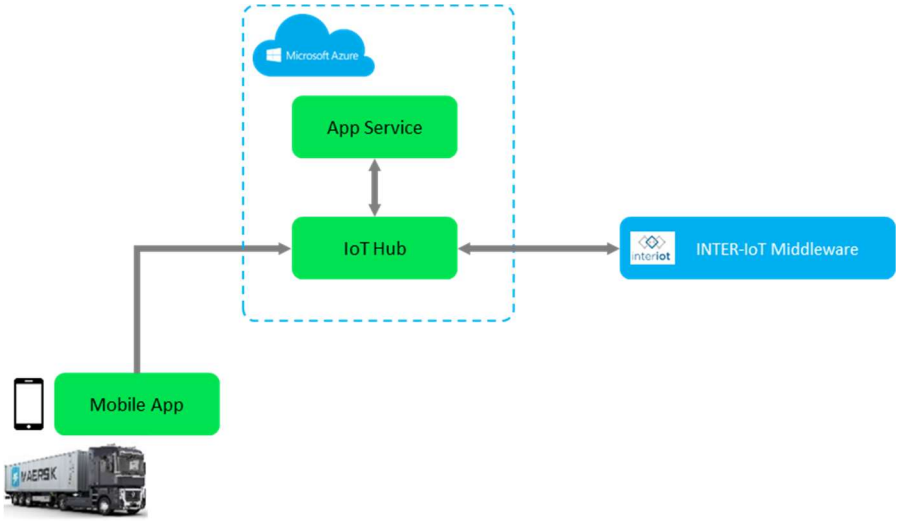


Figure 9: Haulier IoT platform and integration

3.1.7 Integration of INTER-IoT components

In the port domain there are many different companies with different data platforms. Hence, a tool that allows these companies to interoperate is needed. During the INTER-LogP pilot, two of the INTER-IoT components will be used to allow such interoperability.

The most important INTER-IoT component used is the INTER-MW. INTER-MW allows the interconnection of the three IoT platforms and the exchange of data and messages. Each of these platforms has to install or develop a bridge, which is integrated in the Middleware. Once the bridge is ready, the platform can share and receive data. It is also needed to develop a translator in the bridge, in order to allow the semantic translation.

In one of the INTER-LogP pilots, the INTER-IoT Gateway is also used. In the dynamic lighting pilot, several light posts have to send and receive data to/from the port authority IoT platform. Instead of sending the data from each light post, there are two gateways that gather all the data and they send it to the platform.

Similarly, the INTER-FW and the INTER-API are needed as well. INTER-FW allows to manage the previous INTER-IoT components. On its side, INTER-API is included in the framework to provide access to different applications.

[1] https://azure.microsoft.com/en-gb/campaigns/mydriving/

3.1.8 Deliverables and version overview

The following table contains a deliverable list that needs to be signed of before SAT testing commences.

ID	Description	Check
Documents		
1	INTER-IoT D6.1 System Integration Plan	
2	INTER-IoT D6.2 Factory Acceptance Test Plan	
3	INTER-IoT D6.3 Site Acceptance Test Plan	
Software		
4	WSO2 IoT platform	
5	Seams IoT platform	
6	Azure platform	
7	Spotlights and controllers	
Tools		
8	Wireshark	
9	SoapUI	
10	MQTT.fx	
11	Mosquito MQTT	

Table 1. Deliverable checklist

The following table shows the software components and version of the system release.

ID	Description	Version	Check
INTER-IoT Physical Gateway			
1	Physical Gateway		
INTER-IoT Virtual Gateway			
4	Virtual Gateway		
INTER-IoT Middleware			
7	MW		
8	IPSM		

Table 2. Component version overview

3.1.9 Requirements, scenarios and use cases

The following table provides an overview of the relation between the requirements and the test(s) that validate their implementation.

ID	Description	Covered by
1	Roaming across platforms	T1.1
20	Real time support	All
26	Remote device control	T1.3, T1.4
27	System security	All
28	System privacy	All
51	API for data publication	All
74	Ontology support	All
95	Robustness, resilience and availability	All
166	Detection of passive physical entities to start communication with other platforms	T1.1
194	Provide exchange of virtual objects between platforms	All
195	Provide the creation and monitoring of geofences	T1.1

234	Provide connectors to middleware standards	All
237	API Middleware for interoperability between different platforms	All
246	Identification of an object through multiple techniques	T1.1

Table 3. Requirements vs. test mapping

The following table provides an overview of the relation between the scenarios and the test(s) that validate their implementation.

ID	Scenario name	Covered by
6	Dynamic lighting in the port	T1.3, T1.4
7	SCADA port sensor system integration with IoT platforms	T1.1, T1.2
8	SEAMS integration with IoT platforms	T1.1, T1.2
30	IoT access control, traffic and operational assistance	T1.1

Table 4. Scenario vs test mapping

3.1.10 Test environment

3.1.10.1 Introduction

In this chapter the INTER-LogP test environment is described. In addition, there is a list of tools used to test the correct functioning of the different systems.

3.1.10.2 Test environment

This section describes the SAT test environment where INTER-LogP is deployed.

The TRL (Technology Readiness Level) objective of the INTER-IoT components is 6 or 7. So the final demonstration test will be done in a real operational environment, but in a controlled environment. It is important that a prototype never interfere with the daily operations of the company where it is deployed.

For that, the port authority IoT platform is located in the DMZ of the Valencia Port Authority network. It has access to real data, but with security measures that guarantee the correct functioning of the port systems. The mechanisms deployed to access the data can be removed quickly and easily in case of any incident. The system status is checked daily to assess if there is any type of incident.

Most of the data used is restricted, so security is also relevant. The port IoT platform is only accessible for IPs with permissions and for certain ports. The rest of the IPs are discarded. In the case of the Noatum terminal, the data is more sensitive. In this case, the IoT platform is only reachable through a VPN.

In the INTER-LogP pilot, other INTER-IoT components are used, such as the middleware and the gateway. These two components are deployed in an Azure cloud, and they are configured to access the different platforms.

3.1.11 Test tools, hooks and probes

Many of the test tools, hooks and probes listed here were used as well in the Factory Acceptance Test in the lab environment. They are already described in the Deliverable D6.2, to which the reader is referred for more detailed information. For the Site Acceptance Tests in the InterLogP pilot we plan to use the following tools, hooks and probes:

TS_01 Publication of data from a legacy data source

The first and second scenario require the information from the gate access control and environmental stations. The industrial system of the port collects information and registers them in a database. Our task is to use that database as an event generator, so the inserts and updates on the tables of interest are notified to the platform, manipulated, and made available to potential users following the format described in Section 3.1.2.

To do so, we devised the following tools:

- Event detector: we built an event detector to be used with MicroSoft (MS) SQL Server databases, the ones in use in the Port of Valencia. The detector makes use of the MS Service Broker (MSSB) tool, included in the MSSQL Server. The MSSB can implement procedures that detect when new inserts or updates are performed on a series of tables of interest. These events are captured by the broker and stored with a determined format in an internal queue. Additionally, we use the MS SQL Service Broker External Activator Service (SSBEAS) tool that, upon the occurrence of new events, detected as new messages in the queue, will trigger the execution of an external application, our event publisher.
- MQTT Event publisher (MEP): the event publisher is in charge of collecting the events captured by the MSSB, converting them to the agreed format, and publishing them to the target communication mean. The communication mean, in this case, is a MQTT broker.
- Data injector: we use the data injector to replicate a real environment in our lab. It loads past data from the database and injects it following the same time pattern. In this way we can test our system with a load similar to the real one, or using speed up factors, so the data is inserted faster. In this manner, we can test the response of our tools under stress.

Having these tools in place, the flow in both real and test environment (using the data injector) would be the following:

1. The MS SQL Server database receives a new insert or update statement, either from the real industrial system or from the data injector.
2. The MSSB captures the event associated to the new statement. The associated procedure stores data from this event into an internal queue.
3. Simultaneously, the MSSB triggers the SSBEAS.
4. The SSBEAS triggers the MEP.
5. The MEP connects to the SQL Server database and collects the data stored in the internal queue.
6. The MEP formats internally these data into the format agreed, shown in its Section and publishes the data to the MQTT broker hosted in our WSO2 system.

Figure 10 describes these steps mentioned above graphically.

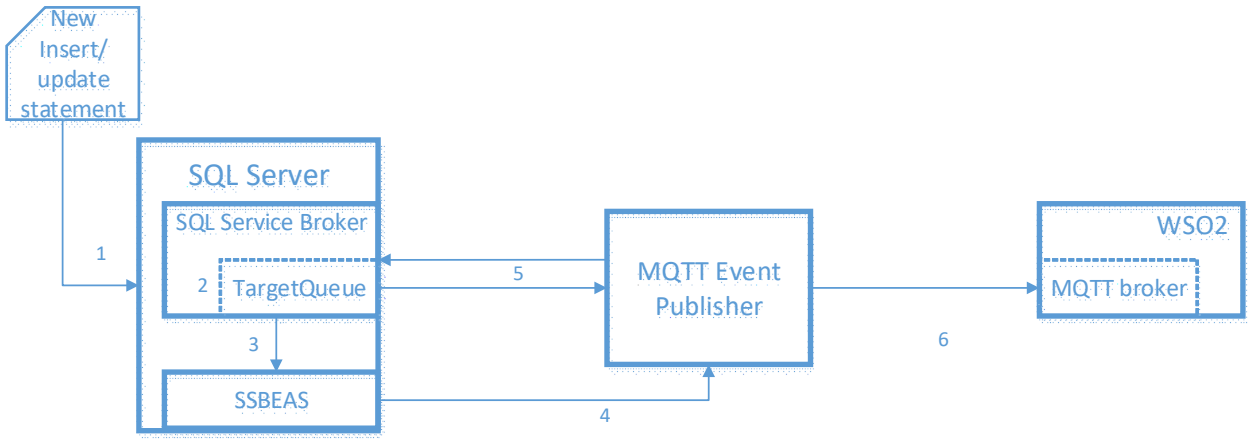


Figure 10: Events flow on detection of a new event until publication in the *MQTT broker.*

TP_01 MQTTFX

MQTTFX is a free MQTT client with a user friendly graphical interface in which users can create MQTT connections with various configurations (e.g. security), subscribe to several MQTT topics as well as to publish messages to an MQTT Broker. In the SAT tests we will use the MQTTFX tool to capture all the events generated by the devices involved in this pilot (gates' sensorization system, meteorological stations, presence sensors, etc.). It will act as a logging tool to check the correct functioning of the INTER-IoT platform as well as the port's platform. As an example, with the aim of checking the proper operation of port's platform, we will be able, for instance, to subscribe to events (via WSO2 MQTT Broker) like the ones thrown by the port's gates systems when trucks pass through. Likewise, we could validate the communication between IoT platforms by subscribing to information generated by Noatum's gates systems.

TP_02 TCPdump

Although TCPdump was already used for the FAT tests it is still worth to use it in SAT as well. TCPdump may be very helpful in diagnosing the status of communication channels between involved IoT platforms, especially in case of problems. By filtering packets per port, MAC address and IP address one can focus in monitoring just the communication flow between a selected pair of entities (e.g. device to platform).

TH_01 SOAP UI

Although SOAP UI was already used in FAT tests, it remains very helpful for SAT tests as well. Once the INTER-IoT platform will be deployed and connected with port's platform through the bridge it will be very important to double check, before the pilot's execution, the correct functioning of different parts of the whole pilot's system. This will be accomplished by injecting test messages into different points in the chain of entities involved in this pilot.

TT_01 Mosquito MQTT Broker

Mosquito MQTT Broker was also used in FAT tests. This test tool will be used as passive monitoring system that will sniff all the MQTT messages that comes into the port's platform and messages going out towards the INTER-IoT system. It will allow us to check that data is flowing between both applications.

3.1.12 Test description

This chapter describes the different tests performed in the INTER-LogP pilot.

3.1.12.1 IoT access control, traffic and operational assistance

The main objective of the defined pilot is providing a service to control access, monitor traffic and assist the operations in the port. Several systems will be able to identify trucks and drivers using different devices. This information can be shared under certain predefined rules due the interoperability between the platforms involved. This information can be used to monitor the truck inside the port by the Port Authority platform (security and safety purposes) and to manage the resources in the terminal more efficiently. This will also allow avoiding queues in the access gates to the port and the terminal.

Interoperability in this scenario is required to connect the port authority, the container terminals and the road hauliers IoT platforms. The resulting service will integrate the:

- Port Authority IoT platform
- Container terminal IoT platform
- Road haulier cloud IoT platform

Truck triggers information

The element starting the communication is the truck once it approaches to the port. At that moment, it starts to send its location information in real time to the haulier company IoT platform, which is the responsible for sharing it.

This use case involves these requirements: [27], [28], [166], [180], [188], [194], [195], [198], [245], [246], [248], [268].

T1.1 IoT access control, traffic and operational assistance

ID	**T1.1**
Test	Verify the integration of all the components in the IoT access control, traffic and operational assistance pilot. The main objective of the defined pilot is a service to control access, monitor traffic and assist the operations at the port.
Type	System testing
Setup	Deployment, installation and configuration of all the components.
Start	A truck arriving to the port.
Req.	[27], [28], [166], [180], [188], [194], [195], [198], [245], [246], [248], [268]
Input	Truck data
Output	Exchange of access data between the port and the terminal
Outcome	Pass / Fail

Test output: INTER-LogP1.1.log

The process that will be developed in the scenario is the following:

1. The truck continuously sends information to the haulier company. This information includes the position.
2. Upon arrival, the truck is detected by the port gates system and the associated data is sent to the port authority IoT platform.

3. The port IoT platform publishes the data to all the entities that are allowed to receive this data.
4. From this moment, the haulier IoT company starts to share the position of the truck with the port and the terminal.
5. When the truck is detected by the Noatum gates system, the data is sent to the terminal IoT platform.
6. The terminal IoT platform publishes the data to all the entities that are allowed to receive this data.
7. All the data is gathered, analysed and represented in a dashboard owned by the port authority.

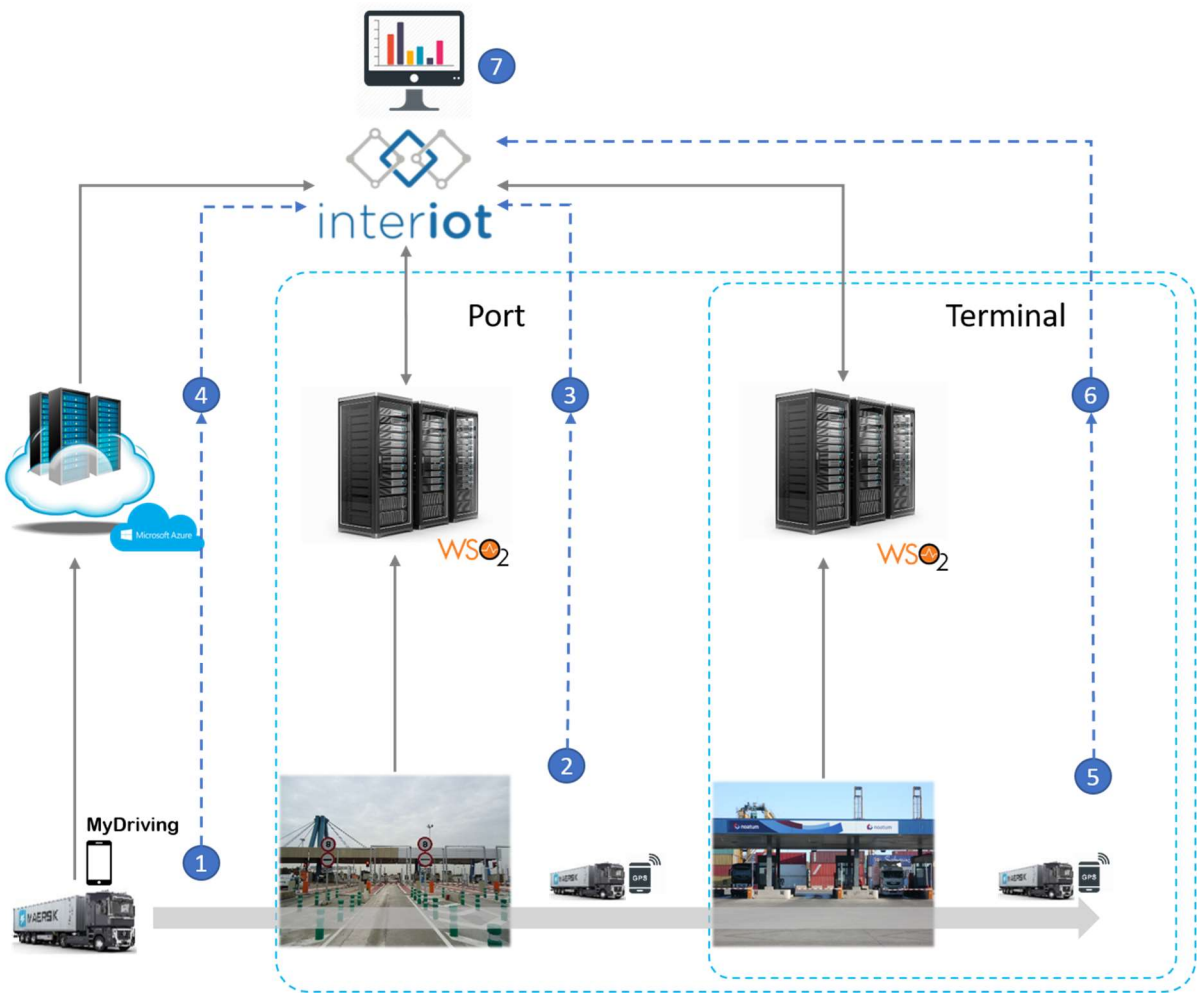


Figure 11: IoT access control, traffic and operational assistance pilot process

3.1.12.2 Pilot Wind gusts detection

The main objective of the defined pilot is to share data related to wind gusts in order to avoid accidents when they become dangerous. The port authority and each of the terminals have their own meteorological stations that detect different environmental data. However, in a dangerous situation, the wind gust may be detected in the terminal itself but it might not be detected early enough. If you could know that wind gusts may be arriving in advance by having them detected by other near meteorological stations, the operation could then be stopped safely.

Interoperability in this scenario is required to connect the port authority and the container terminal IoT platforms.

The resulting service will integrate the:

- Port Authority IoT platform
- Container terminal IoT platform

Wind triggers information

The element that starts the communication is the wind gust, in the different meteorological stations. The port and the terminal share the wind data to detect dangerous situations.

This use case involves these requirements: [27], [28], [180], [188], [194], [245], [268].

T1.2 Pilot Wind gusts detection

ID	T1.2
Test	Verify the integration of all the components in the wind gust detection pilot. The main objective of the defined pilot is providing a service to share meteorological data to improve the safety.
Type	System testing
Setup	Deployment, installation and configuration of all the components.
Start	A wind gust is detected.
Req.	[27], [28], [180], [188], [194], [245], [268].
Input	Wind data
Output	Exchange of wind data between the port and the terminal
Outcome	Pass / Fail

Test output: INTER-LogP1.2.log

The process that will be developed in the scenario is the following:

1. The port weather stations detect wind data that is stored in the port IoT platform.
2. When the wind gust exceeds a threshold, the event is published through INTER-IoT.
3. In the same way, Noatum has its own weather station that are storing metrological data.
4. When a dangerous wind gust is detected, is also published.
5. All the data is gathered, analysed and represented in a dashboard.

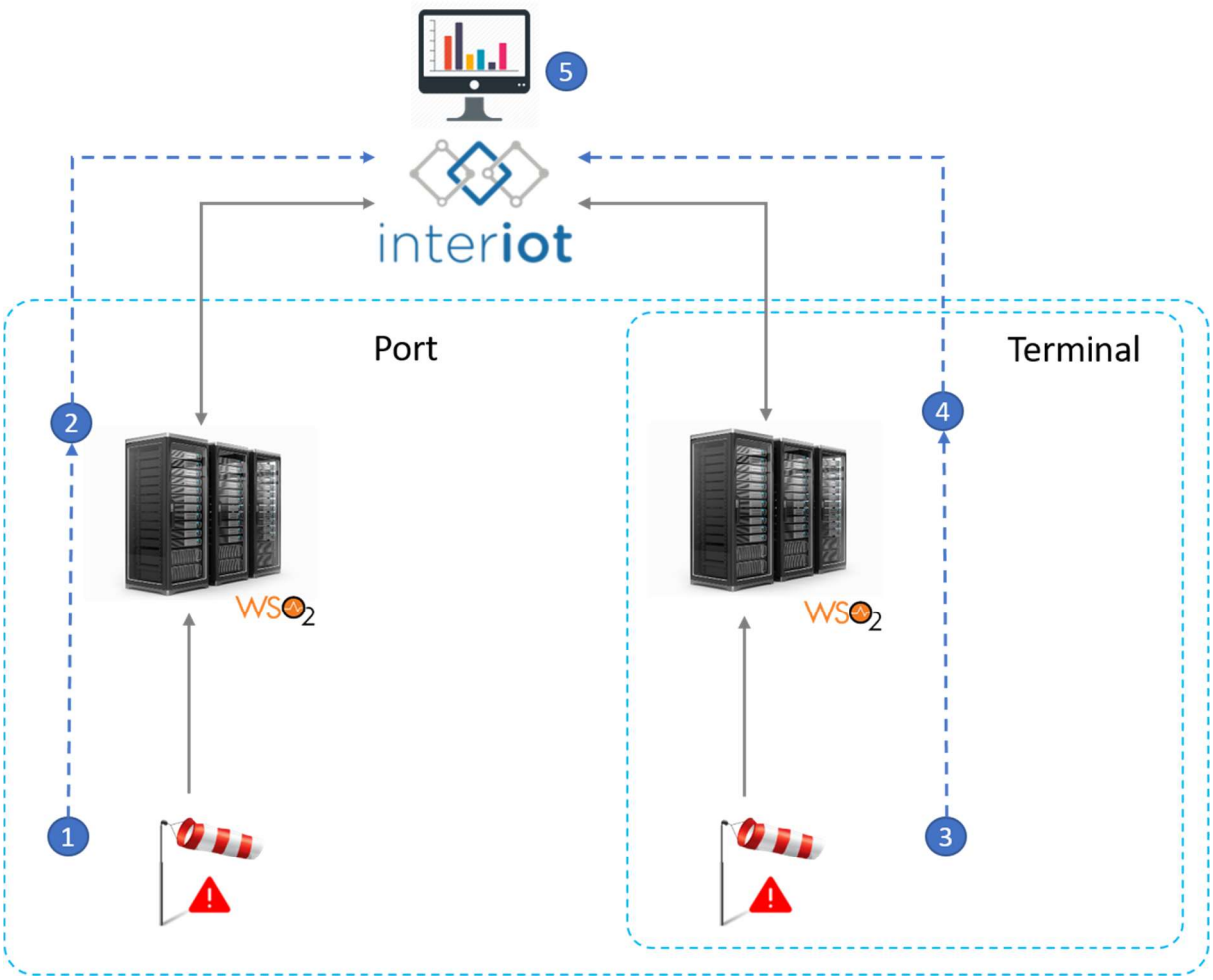


Figure 12: Wind gusts detection pilot process

3.1.12.3 Pilot Dynamic lighting

The goal of this pilot is to expand the smart illumination (dynamic Illumination) in the yard of Noatum for the rail yard and in the railway switchgear area (terminal access). In the first case, the lighting posts belong to the port authority of Valencia but the machinery belongs to Noatum. In the second case, the lighting posts belong to Noatum and the port authority manages the switchgear area. In both cases, an exchange of data between both companies is needed in order to illuminate the area properly during the operation.

Interoperability in this scenario is required in order to connect the port authority and the container terminals.

The resulting service will integrate the:

- Port Authority IoT platform
- Container terminal IoT platform

Lighting triggers information

The intensity of the lighting is reduced when there is no activity in the terminal. When a truck or a train access the railway area the intensity of lighting should be increased. Therefore, the truck or train are the elements that trigger an event by sending their position.

This use case involves these requirements: [27], [28], [168], [180], [198], [245].

T1.3 Pilot Dynamic lighting. Train arriving to Noatum's rail yard

ID	T1.3
Test	Verify the integration of all the components in the dynamic lighting pilot. The goal of this pilot is to develop a smart illumination (dynamic Illumination) system for the rail yard in the yard of Noatum.
Type	System testing.
Setup	Deployment, installation and configuration of all the components.
Start	A train or machinery accesses the rail yard area in the terminal.
Req.	[27], [28], [168], [180], [198], [245].
Input	Data from PIR sensors.
Output	The light intensity in the rail yard terminal is adjusted for the operation.
Outcome	Pass / Fail

Test output: INTER-LogP1.3.log

The process that will be developed in the scenario is the following:

1. Activity is detected in the train yard (train, truck, or person) by PIRs located on the left corner (truck access) or on top of the building.
2. Upon detection, the associated data is sent to the terminal IoT platform to be processed. The light posts that depend on the terminal raise their light intensity.
3. At the same time, the data is published through INTER-IoT and, hence, shared with the port.
4. The two platforms managing some of the port streetlights receive the data and raise the light intensity.

Figure 13: Dynamic lighting pilot 1 process

T1.4 Pilot Dynamic lighting. Train in the arriving to railway switchgear area

ID	T1.4
Test	Verify the integration of all the components in the dynamic lighting pilot. The goal of this pilot is to develop a smart illumination (dynamic Illumination) system for the railway switchgear area.
Type	System testing.
Setup	Deployment, installation and configuration of all the components.
Start	A train or machinery accesses the rail yard area in the terminal.
Req.	[27], [28], [168], [180], [198], [245].
Input	Data from PIR sensors.
Output	The light intensity in the railway switchgear area is adjusted for the operation.
Outcome	Pass / Fail

Test output: INTER-LogP1.4.log

The process that will be developed in the scenario is the following:

1. Activity is detected in the railway switchgear (train or person) by PIRs located at the beginning or at the end of the area.
2. Upon detection, the data is sent to the port IoT platform to be processed.
3. At the same time the data is published though INTER-IoT and, hence, shared with the terminal.
4. The terminal IoT platform receives the data and raises the light intensity in the area.

Figure 14. Dynamic lighting pilot 2 process

Reliability of components

In all the INTER-LogP pilots, the INTER-IoT products are the key components. The objective of these scenarios is to demonstrate the usefulness of the components and test all their features. In this case, the two INTER-IoT components used are the Middleware and the Gateway. For both components, we need to perform deep performance tests.

T2.1 MW Reliability

ID	T2.1
Test	Integrate and test the middleware in the port environment. For that, we will carry out communication tests, sending messages, forced shutdown, stress tests, long-term stability, bridges deployment tests, etc.
Type	Integration and testing.
Setup	Deployment, installation and configuration of the middleware.
Start	Bridges for each type of platform are installed and platforms start to exchange data with the middleware.
Req.	[2], [28], [43], [95], [234], [235], [236], [237], [238].
Input	Messages coming from the platforms through the MW. Clients requesting data.
Output	Messages arrive to their destination.
Outcome	Pass / Fail

Test output: INTER-LogP2.1.log

T2.2 Gateway Reliability

ID	T2.2
Test	Integrate and test the gateway in the port environment. For that, we will carry out communication tests, sending messages, forced shutdown, stress tests, long-term stability, etc.
Type	Integration and testing.
Setup	Deployment, installation and configuration of the gateway.
Start	A gateway is installed in the port and sensors start to send observations through the gateway.
Req.	[2], [28], [39], [95], [243], [244], [245], [237], [238].
Input	Observations coming from the sensors to the gateway. Actuations coming from the platform to the gateway.
Output	Messages arrive to their destination.
Outcome	Pass / Fail

Test output: INTER-LogP2.2.log

3.1.13 Test outcome overview

The following table will provide an overview of the test result of all the performed tests in this SAT.

Test	Description	Outcome
T1.1	IoT access control, traffic and operational assistance	Pass / Fail
T1.2	Pilot Wind gusts detection	Pass / Fail
T1.3	Pilot Dynamic lighting. Train arriving to Noatum's rail yard	Pass / Fail
T1.4	Pilot Dynamic lighting. Train in the arriving to railway switchgear area	Pass / Fail
T2.1	MW Reliability	Pass / Fail
T2.2	Gateway Reliability	Pass / Fail
SAT Outcome		**Pass / Fail**

Table 5. Test outcome overview

3.1.14 Integration ethics and security

Introduction

This section addresses recommendations and proposed solutions to the ethical and security points of the integration.

INTER-LogP

Privacy and confidential

The objective of this project is interoperability, so the exchange of data between components and platforms is mandatory. However, in the port sector this data could be sensitive to the owner's company, so agreements are necessary among the organizations to use the data only for the agreed purpose. Moreover, the information should not be shared with anyone else.

The data used in the INTER-LogP pilot is not critical or sensitive, oppositely to the health pilot, but it is confidential. The access has to be secure as this data could be used by competitors for market positioning.

Furthermore, all the INTER-IoT developments and platforms must adapt to the General Data Protection Regulation (GDPR) legislation that becomes enforceable on May 2018. For this reason, the data owner will always be able to give consent, see who is accessing their data and revoke the access permissions if necessary.

One of the lessons learnt regarding data sharing in a port environment is that several companies are reluctant to share their data. If these companies are not in the consortium agreement and they do not see a clear benefit to them, they do not want to participate.

Security

In addition to privacy and confidentiality, all these processes require a high level of security. The security must be guaranteed in communications and in each of the intermediate components. This must be done through the use of secure and encrypted communication channels and with high security in the middleware.

In the port IoT platform there is an Identity Access management module that manages the security. A token is necessary to access the data through any of the APIs or the real-time data in the broker. The protocol used for this purpose is OAuth version 2.0. In fact, the client has to go through two steps before getting a valid token: first, it must get the client id and the secret key, and second, the client uses the client id and secret key to request a token using the OAuth2 Client Credentials Grant type.

Apart from OAuth2 access protocol, each client is required to establish a connection using SSL/TSL encryption protocol. Clients need a CA signed certificate in order to connect with the port IoT platform and thus exchange information encrypted by both sides.

3.2 INTER-Health SAT

The following scheme shows the deployment of hardware equipment:

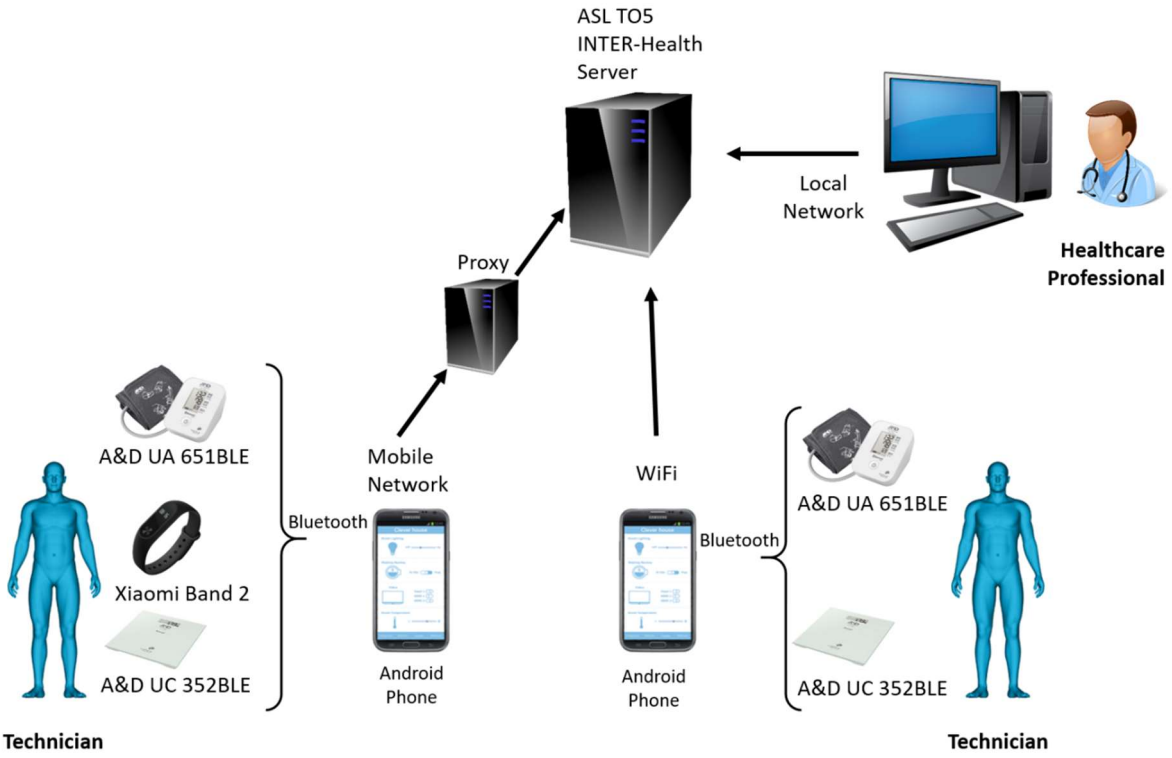


Figure 15: INTER-Health Hardware test overview

The network in ASLTO5 is pre-existent and should not be modified just to integrate this new system – or as little as possible.

The Local Server is a dedicated machine. It runs a Windows Server 2016 OS. Its main specifications are:

- Intel Xeon 3.8GHz
- 32GB Ram
- 4x1TB HDD + RAID controller

The Proxy shields the Local Server connection to the Internet and provides naming and HTTPS connections. It is part of the existing infrastructure of ASLTO5 network.

The PCs used to access the Professional Web Tool (PWT) will be the regular PCs used by the health professionals. Since access to the PWT is through a Web Browser their specifications will most likely not have any impact, but it is still worth taking into account.

The only specification regarding the Android Phones used by health professionals to take measurements inside ASLTO5 premises is that they support Bluetooth Low Energy: The sensor devices used at ASLTO5 as well as those given to patients are Bluetooth Low Energy-compatible weight scales: A&D UC 352BLE and blood pressure sensors A&D UA 651BLE. The patients also have the wrist band Xiaomi Band 2.

The phones used by patients will be different models, as long as they are Android, have Bluetooth, and are compatible with the requirements of the BodyCloud app. The BodyCloud App is an existing software component that is not being tested in this SAT, we consider it to be stable in regards of its communication features. If BodyCloud fails the E-Health system does not fail: doctors can keep using the tool as usual, they just won't get info from BodyCloud devices. The interfaces between the mobile app and the server are part of the BodyCloud system itself, which is considered to be tested.

The following scheme shows the deployment of main software subsystems:

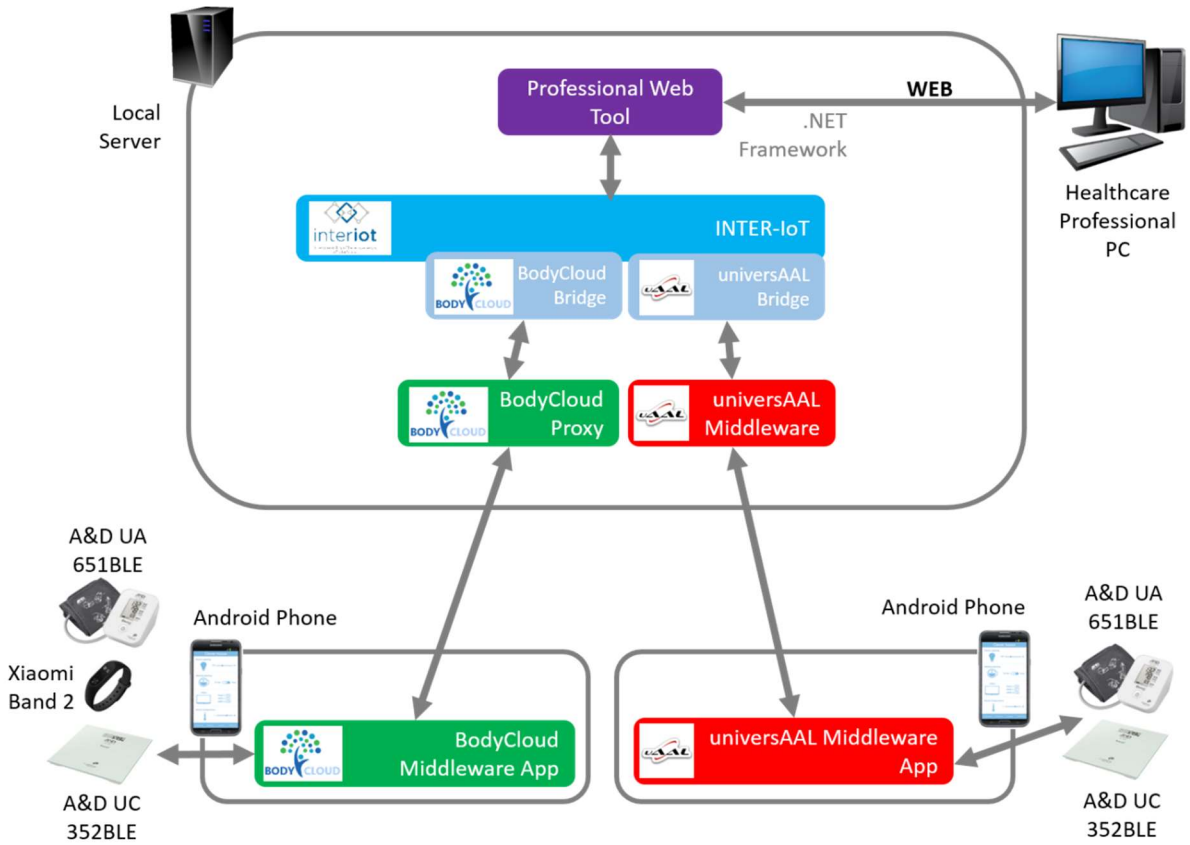


Figure 16: INTER-Health Software overview

The Local Server runs Windows Server 2016 OS with nested virtualization capabilities. The following table shows the installed software components that form each of the above subsystems inside the Local Server. Some have to be run on top of a container software (Docker) component that is also installed:

Component	Subsystem	Version	Container
INTER-MW	INTER-IoT	0.0.1-SNAPSHOT	Apache Tomcat 8.5.23, Java JDK 8
RabbitMQ	INTER-IoT	3.7	Docker 17.09.1-ce-win42
WSO2 API Manager	INTER-IoT	2.1.0	Docker 17.09.1-ce-win42
Kafka	INTER-IoT	1.0.0	Docker 17.09.1-ce-win42
IPSM	INTER-IoT	0.4.0.3	Docker 17.09.1-ce-win42
Parliament DB	INTER-IoT	-	Docker 17.09.1-ce-win42
Professional Web	Professional Web	1.0.0	IIS, .NET Framework 4.7

Tool Application	Tool		
MS SQL Server	Professional Web Tool	2014 Version 12.0.5204.0	Native (Windows)
MySQL	BodyCloud Proxy	15.1	XAMPP v7.1.10
BodyCloud Proxy	BodyCloud Proxy	1.0.2	Apache Tomcat 8.5.23
UniversAAL server	UniversAAL Middleware	3.4.1-SNAPSHOT	Karaf OSGi 3.0.8, Java JDK 8

Table 1: Local Server software components

**Docker 17.09.1-ce-win42: This version of Docker was tested prior to the SAT itself to be the one that works acceptably with the environment. Newer and previous versions were tested, showing different issues*

The phones will run either one of the applications in the following table. No phone will run both applications at the same time. This accounts for patients' phones and doctors' phones, which accomplish different use cases.

Application	Version
BodyCloud INTER-Health App	0.0.1-SNAPSHOT
UniversAAL INTER-Health App	0.0.1-SNAPSHOT

Table 2: Android Phones applications

The browsers used by the HealthCare Professional PCs used in the test are the following, although other browsers could be used after the deployment is tested:

Application	Version
Mozilla Firefox	52.2
Google Chrome (if needed)	Latest at the time of testing

Table 3: Healthcare Professional browsers

3.2.1 Integration of IoT framework

As depicted above, INTER-IoT, in particular INTER-Framework, is used to allow the Professional Web Tool to access data from two different IoT platforms. For this reason, only the INTER-Middleware (or MW2MW) layer of INTER-Layer is of interest for the pilot.

All modules required to run INTER-Middleware are installed in the server, as listed above, but no other layers are required. The goal is that the Professional Web Tool uses INTER-API to access the required data through INTER-Framework, but in the FAT it will use a mix of API and some hardcoded values, for simplicity and speed of development.

The following table lists the components of INTER-IoT that are installed in the FAT and, if they do, how they interface with external components. Not shown are the other core components that are always required to execute INTER-IoT (IPSM, Parliament, etc):

INTER-IoT Component	Interface	With component
INTER-Middleware (BodyCloud Bridge)	BodyCloud's local REST-API and callback	BodyCloud Proxy
INTER-Middleware (UniversAAL Bridge)	UniversAAL's local REST-API and callback	UniversAAL Middleware
INTER-API	INTER-IoT local REST-API	Professional Web tool

3.2.2 Test coverage

Certain subsets of tests cannot be performed properly at this point given the deployment configuration. Tests that alter the infrastructure of the site (like plugging or unplugging cables or shutting down other systems) cannot be performed by SABIEN, since we do not have physical access to it. It is not just the hardware is located in a different country, but also that we are not allowed to physically access it (the hardware was sent to the ASLTO5 staff already assembled and installed), we only have remote access via VPN and Remote Desktop. We also cannot consider making tests that could affect hardware or software alien to the project, or the network of the site itself, in order not to alter the normal operation of the outpatient clinic. Finally, the security of the "external" remote accesses was also not tested as they depend on the security configuration set up by ASLTO5 on their premises, which is already existing and considered to be "tested" outside the scope of this project

3.2.3 SAT execution remarks

During execution of the INTER-Health SAT a few issues where found while testing. All of them were fixed, except for one related to Docker, which relies on a workaround. In this case a series of patches have been made which makes it more maintainable to keep this workaround than trying to fix it as that would cost a considerable amount of time.

The issue with the version of Docker for Windows (Community Edition) that we use. It is a known issue, reported by the community:

https://github.com/docker/for-win/issues/573

https://github.com/docker/for-win/issues/426

It blocks ports opened by containers in Docker after an arbitrary amount of time or load (in the pilot we have identified it happening every 2 days on average). When it was first identified, it seemed to stem from the POSTGRESQL container. This in turn affected the IPSM, and so on in cascade, until the INTER-MW stopped working properly. We observed two things could happen: a) that the error happened only once (or some isolated times). If this happened, it seemed INTER-MW kept working. b) That it happens continuously, (every time IPSM tried to communicate with POSTGRESQL it failed). In this case, there is no other solution than restarting Docker completely.

Eventually, we tried a possible fix by removing the POSTGRESQL container and relying on other storage mechanisms. This removed the original problem from the POSTGRESQL container, but the problem still manifested overall at the same intervals. This is in line with the reports from the community stating there is actually no fix to this issue, other than restarting Docker when it happens.

We set up a watchdog routine that checks periodically if INTER-MW is still working, and restart it's when it doesn't. How this affects the users when the system is temporarily down, depends on the IoT platform in use: If INTER-MW is not running when taking measurements from UniversAAL, the doctors can realize that the system is not working properly because the measure does not appear. Then a restart can be requested to technicians. For user measures taken from BodyCloud, this is not perceivable. Measures simply will not arrive at the backend and will not be available in the PWT. This is solved with an updated BodyCloud server that holds measures if it cannot send them through INTER-MW, and re-sends them when it is back up.

Proper fixes

As many Docker users comment in the reported issues, restarting Docker is simply a workaround until the problem shows up again. The fix is to simply not use Docker for Windows CE, as it is not stable enough. The alternative could be using a stable release of Docker Enterprise Edition running on a Windows Server version.

INTER-MW uses Linux containers in Docker. They are supposed to run on a Linux host. On the other hand, Docker for Windows CE by default uses Windows containers. To run Linux containers, Docker for Windows CE relies on a virtual machine to run Linux containers. This ability however is not available in Docker Enterprise Edition until a particular later version with HyperV isolation, to be run with Windows Server version 1709 (Insider Preview build 16267). That Windows Server version is not a LTS (Long Time Support) version and was not available to UPV to install in the INTER-Health server. In theory, installing this Windows Server version with this Docker EE version could fix this issue, but there is no guarantee that it will, as the community is still reporting some other issues on that version (including POSTGRESQL, and RabbitMQ, still used by INTER-MW). That Windows Server version is also a Core edition, which has no GUI, which means some of the platforms and tools required for INTER-Health may not be compatible.

Another alternative solution would be to run a Linux virtual machine over Windows ourselves, then install Docker for Linux on it. Since interactions between INTER-MW components are made through mapped network ports and actual network interfaces, it should still work with the rest of the system. However, this will require additional resources (memory, processor…) and the server was not built with this in mind, and has not been tested. It would also be complicated and time-consuming to set up remotely, now that the server machine is already installed on pilot premises.

All these obstacles together are why we decided to continue with the current workaround while the pilot lasts.

3.2.4 Deliverables and version overview

The following table contains a deliverable list which need to be signed of before SAT testing commences.

ID	Description	Check
Documents		
1	Validation and Test reports of the IoT system components	Y
2	Validation and Test reports of the Professional Web Tool	Y
3	Validation and Test reports of UniversAAL	Y
4	Validation and Test reports of BodyCloud	Y
Hardware		
4	Local Server	Y
5	Healthcare Professional PC	Y
6	Android Phones (ASLTO5)	Y
Tools		
7	MS Windows Remote Desktop	Y

Table 4: Deliverable checklist

The following table shows the software components and version of which the system release version 1.0 consists of.

ID	Description	Version	Check
INTER-IoT			
1	Java JDK	1.8	Y
2	Apache Tomcat	8.5.23	Y
3	Docker	17.09.1-ce-win42	Y
4	Parliament DB		Y
5	IPSM	0.4.0.3	Y
6	Kafka	1.0.0	Y
7	WSO2 API Manager	2.1.0	Y
8	RabbitMQ	3.7	Y
9	INTER-MW	0.0.1-SNAPSHOT	Y
Professional Web Tool			
10	MS SQL Server	12.0.5204.0	Y
11	.NET Framework	4.7	Y
12	.NET Core Windows Server Hosting	1.0.4 & 1.1.1	Y
13	Professional Web Tool Application	1.0.0	Y
BodyCloud			
15	XAMPP	7.1.10	Y
16	MySQL	15.1	Y
17	BodyCloud Proxy	1.0.2	Y
UniversAAL			
18	Karaf OSGi	3.0.8	Y
19	UniversAAL Server (Middleware + REST API)	3.4.1-SNAPSHOT	Y

Table 5: Component version overview

3.2.5 Requirements, scenarios and use cases

The following table provides an overview of the relation between the requirements and the test(s) that validate their implementation.

ID	Description	Covered by
62	Constraints on processing of personal and health data	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.3.2, T1.3.3, T1.4.1, T1.4.2
71	Application response time	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.3.2, T1.3.3, T1.4.1, T1.4.2
101	Exchanging discrete medical measures across platforms	T1.3.1, T1.3.2, T1.3.3
102	Exchanging complex medical measures across platforms	T1.3.1, T1.3.2
103	User Authentication to access INTER-Health services	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.3.2, T1.3.3, T1.4.1, T1.4.2
104	Personal data and user profile management	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.3.2, T1.3.3, T1.4.1, T1.4.2
106	Definition of reference meaning for health information	(Theoretical, not testable practically)
107	Exchanging synthetic or statistical health information between platforms	T1.3.1, T1.3.2, T1.3.3
127	Availability of sensor data	T1.3.1, T1.3.2
145	Informed consent. Processing of personal data	T1.1.1, T1.1.2, T1.2.1
146	Information sheet. Processing of personal data	T1.1.1, T1.1.2, T1.2.1
157	Seamless patient monitoring	T1.3.1, T1.3.2, T1.3.3, T1.4.1, T1.4.2
158	National, regional and local Bioethic Committee	T1.1.1, T1.1.2, T1.2.1
164	Medical Device informatics	T1.3.1, T1.3.2
172	User Access Service for Patients	T1.1.3, T1.3.2, T1.3.3
173	User Access Service for Doctors	T1.1.1, T1.1.2, T1.2.1, T1.3.1, T1.4.1, T1.4.2
174	User Access Service for Administrators	T1.1.1, T1.1.2, T1.2.1
176	User Access Gateway for Patients	T1.1.3, T1.3.2
177	User Access Gateway for Caregivers	T1.3.1
188	Integration with legacy systems	T1.1.1, T1.1.2, T1.2.1, T1.4.1, T1.4.2
217	Wearable devices support	T1.3.2
218	Personal data collected on Computerized Nutritional Folder	T1.2.1, T1.3.3, T1.4.2

Table 6: Requirements vs. test mapping

The following table provides an overview of the relation between the scenarios and the test(s) that validate their implementation.

ID	Scenario name	Covered by
1	Chronic disease prevention	All
11	Primary prevention of cognitive decline	
12	Health failure disease and mild Alzheimer disease	
15	Surveillance systems for prevention programs	All
16	Elderly monitoring	All

21	Low risk of developing chronic diseases.	All
22	Increased risk of developing chronic diseases	All
23	High risk of developing chronic diseases	All
24	Very high risk of developing chronic diseases	All
25	Extremely high risk of developing chronic diseases	All
27	Vitamins intake analyser	
28	Calories / nutrition mixer / cookware counter	

Table 7: Scenario vs test mapping

3.2.6 Test environment

3.2.6.1 Introduction

To test the functionality of the integrated INTER-Health in combination with the IoT framework representative test hooks in the system are needed. This chapter will describe this environment and the used hardware, software, tools and platforms.

3.2.6.2 Test environment

For details, refer to system description in section 3. The environment for the test comprises:

The server machine: This computer is located in the premises of ASLTO5, with restricted physical access allowed only to ASLTO5 personnel.

The local network used in the test is the one at the Healthcare center that will be used in the real deployment. It is managed by ASLTO5 staff, as well as the proper sub-networks, firewalls and proxies. The performance and stability of the network is not under test in this SAT: We do not have the access privileges to test it, it would be disruptive for ASLTO5, and we can consider it to be stable enough given it is the one used for their daily operations.

The computers for web access to the PWT are the Healthcare professional's desktops, as well as, for some tests, the local server machine. They all run on different versions of Windows with either Firefox or Chrome installed to do the end-user tests of the Professional Web Tool.

All access to tools, hooks and probes by UPV-SABIEN developers are performed in the local server through Remote Desktop via VPN for the duration of the tests.

The mobile phones used are the ones from ASLTO5 for "local" tests, and a developer phone Motorola Moto G4 running Android 6.0 or 7.0 to perform tests from UPV. It is possible however that during the course of the test other phone models are tested, if available.

The sensor device models used in the test will be the same as those used by patients and ASLTO5: A&D UA 651BLE and A&D UC 352BLE, and Xiaomi Band 2.

The role of patients and healthcare providers will be performed indistinctively by any of the UPV-SABIEN and ASLTO5 staff at convenience.

3.2.7 Test tools, hooks and probes

This paragraph describes the test setups, tools, hooks and probes used in the by this document defined tests.

A test setup defines a setup used for a test, and describe the used system parts, interface etc.

Tools define the used COTS tooling that will be used during testing e.g. Logic analyzers, packet sniffer like e.g. Whireshark, developed scripts, etc.

Hooks are used to inject data or control parts of the system used to get the system to handle a needed scenario. This can be hidden, debug or service interfaces which contain the needed functionality.

Probes are used to gather logging from parts of the system under test to provide system feedback and to prove that the system is handling the scenario as it should.

The following paragraphs will define the used setups, tools, hooks and probes which will be referred to by the test descriptions.

TS_01 Overall test setup

This generic setup is used in all tests: A UPV-SABIEN/ASLTO5 tester plays the role of patient and has access to one mobile phone, already paired to each of the available sensor devices. The tester interfaces directly with the sensors to make measurements, and with the applications in the phone.

Another ASLTO5 tester, playing the role of Healthcare professional, has access, through his/her computer, to the Professional Web Tool. We take into account the fact that Healthcare professionals have had a minimum of training with the tool before using it, so we will not test random clicking (Monkey testing). During their training, it was proved that this was not an issue.

Finally, another UPV-SABIEN tester will act as observer and accesses the server through remote desktop to monitor the running programs and obtain the required results.

TT_01 Microsoft Windows Remote Desktop

Access to the software running in the server is performed through Remote Desktop. UPV-SABIEN testers have access to this server and monitor it in real time while the tests are performed, and obtain all necessary outputs.

TH_01 INTER-IoT Message Emulators

There are emulators that publish sensor data as if it came from either BodyCloud or UniversAAL. These are not officially part of any test, but may be used in case of issues with real data, in order to compare as a baseline or to test the overall system when needed.

TP_01 IoT Platforms consoles

Both UniversAAL and BodyCloud are installed in the system server. Some of their outputs can be observed in real time through their consoles. This can help identify crashes or problems as they happen.

TP_02 IoT Platforms output logs

Both UniversAAL and BodyCloud are installed in the system server. They generate output log files stored in the server that can be obtained after running, or even while the system is running, depending on the log editor used. This can help identify crashes or problems after they happen. For security reasons, file paths to logs shown in the tests descriptions below do not contain actual, full paths.

TP_03 Android Studio

Console output and logs from the mobile phones can be accessed through Android Studio. In order to enable this, the phones used in the tests have to enable their “developer options”, connect to a PC through USB and allow debugging. This is only available at the developer phone used by UPV-SABIEN.

3.2.8 Test description

3.2.8.1 Scenario 1, 15, 16, 21, 22, 23, 24, 25

U1 – Creates and operates users/services

T1.1.1 Professional creates user

ID	T1.1.1
Test	The Healthcare Professional enters the system and creates a new patient user
Type	Manual test
Setup	TS_01, TT_01
Start	First time run, empty records, Patient has been given phone and documentation
Req.	[62], [71], [103], [104], [145], [146], [158], [173], [174]
Input	Healthcare Prof. enters PWT, authenticates. Healthcare Prof. enters option to create new patient Healthcare Prof. enters patient data and confirms creation
Output	The new Patient is registered in the system The Patient can now use the system as intended
Logs	{PWT}\Daemons {InternetServer}\INTER-IoT\Logs
Outcome	Pass / Fail

T1.1.2 Professional modifies user

ID	T1.1.2
Test	The Healthcare Professional enters the system and updates a patient’s data
Type	Manual test
Setup	TS_01, TT_01
Start	Patient was already created in the system, Patient has been given phone and documentation
Req.	[62], [71], [103], [104], [145], [146], [158], [173], [174]
Input	Healthcare Prof. enters PWT, authenticates. Healthcare Prof. enters option to update patient Healthcare Prof. enters new patient data and confirms update
Output	The Patient’s new data is registered in the system The Patient can continue to use the system as usual
Logs	{PWT}\Daemons {InternetServer}\INTER-IoT\Logs
Outcome	Pass / Fail

T1.1.3 Patient logs with their profile

ID	T1.1.3
Test	Patient enters the system through his/her mobile phone apps to access data
Type	Manual test
Setup	TS_01, TT_01, TP_01, TP_02, TP_03
Start	Patient is in possession of mobile phone with installed app Patient has been registered in the system with proper data
Req.	[62], [71], [103], [104], [172], [176]
Input	Patient enters BC app, authenticates, checks data
Output	Patient successfully accesses app Patient can check up-to-date data
Logs	{PWT}\Daemons {InternetServer}\INTER-IoT\Logs
Outcome	Pass / Fail

U2 – Set patient protocol parameters

T1.2.1 Professional sets protocol

ID	T1.2.1
Test	The Healthcare Professional enters the system and updates a patient's protocol
Type	Manual test
Setup	TS_01, TT_01
Start	Patient was already created in the system, Patient has been given phone and documentation
Req.	[62], [71], [103], [104], [145], [146], [158], [173], [174], [218]
Input	Healthcare Prof. enters PWT, authenticates. Healthcare Prof. enters option to update patient Healthcare Prof. enters new patient protocol and confirms update
Output	The Patient's new data is registered in the system The Patient can continue to use the system as usual, according to new protocol
Logs	{PWT}\Daemons {InternetServer}\INTER-IoT\Logs
Outcome	Pass / Fail

U3 – Perform objective and subjective measures

T1.3.1 Professional collects measures (objective)

ID	T1.3.1
Test	Healthcare Professional takes measures from Patient at centre using devices
Type	Manual test
Setup	TS_01, TT_01, TH_01, TP_01, TP_02, TP_03

Start	Patient has been registered in the system with proper data Healthcare prof. is in possession of mobile phone with installed app Sensor devices are paired to mobile phone
Req.	[62], [71], [101], [102], [103], [104], [107], [127], [157], [164], [173], [177]
Input	Healthcare Prof. enters PWT, authenticates. Healthcare Prof. enters option to update patient measures Healthcare Prof. enters UniversAAL app, authenticates Patient takes measurement on centre sensor device
Output	Patient measure appears on uAAL app and PWT, allowing Healthcare Prof. to update patient measure data
Logs	{uAAL}\data\log
Outcome	Pass / Fail

T1.3.2 Patient performs measures (objective)

ID	T1.3.2
Test	Patient takes measures at home using devices
Type	Manual test
Setup	TS_01, TT_01, TH_01, TP_01, TP_02, TP_03
Start	Patient has been registered in the system with proper data Patient is in possession of mobile phone with installed app Sensor devices are paired to mobile phone
Req.	[62], [71], [101], [102], [103], [104], [107], [127], [157], [164], [172], [176], [217]
Input	Patient successfully accesses app Patient takes measurement on home sensor device
Output	The measure is registered in the system at the Healthcare centre and can be checked by Healthcare Prof. in PWT.
Logs	{tomcat}\logs
Outcome	Pass / Fail

T1.3.3 Patient performs measures (subjective)

ID	T1.3.3
Test	Patient answers questionnaire about habits
Type	Manual test
Setup	TS_01, TT_01, TP_01, TP_02, TP_03
Start	Patient has been registered in the system with proper data Patient is in possession of mobile phone with installed app Healthcare Prof. has set protocol
Req.	[62], [71], [101], [102], [103], [104], [107], [157], [172], [218]
Input	App notifies Patient about questionnaire Patient successfully accesses app Patient takes questionnaire

Output	The measure is registered in the system at the Healthcare centre and can be checked by Healthcare Prof. in PWT.
Logs	{tomcat}\logs
Outcome	Pass / Fail

U4 – Monitors subjective and objective parameters

T1.4.1 Professional monitors parameters (objective)

ID	T1.4.1
Test	Healthcare Professional accesses Patient data recorded through sensors
Type	Manual test
Setup	TS_01, TT_01, TH_01, TP_01, TP_02
Start	Patient has been registered in the system with proper data Patient has recorded objective measures Healthcare prof. has recorded objective measures of patient
Req.	[61], [71], [103], [104], [157], [173]
Input	Healthcare Prof. enters PWT, authenticates. Healthcare Prof. enters option to observe patient measures
Output	The PWT displays the measures taken with the sensors
Logs	{PWT}\Daemons {InternetServer}\INTER-IoT\Logs
Outcome	Pass / Fail

T1.4.2 Professional monitors parameters (subjective)

ID	T1.4.2
Test	Healthcare Professional accesses Patient data recorded through questionnaires
Type	Manual test
Setup	TS_01, TT_01, TH_01, TP_01, TP_02
Start	Patient has been registered in the system with proper data Patient has recorded objective measures Healthcare prof. has recorded objective measures of patient
Req.	[61], [71], [103], [104], [157], [173], [218]
Input	Healthcare Prof. enters PWT, authenticates. Healthcare Prof. enters option to observe patient measures
Output	The PWT displays the measures taken with the questionnaires
Logs	{PWT}\Daemons {InternetServer}\INTER-IoT\Logs
Outcome	Pass / Fail

3.2.9 Test outcome overview

The following table will provide an overview of the test result of all the performed tests in this SAT.

Test	Description	Outcome
T1.1.1	Professional creates user	Pass / Fail
T1.1.2	Professional modifies user	Pass / Fail
T1.1.3	Patient logs with their profile	Pass / Fail
T1.2.1	Professional sets protocol	Pass / Fail
T1.3.1	Professional collects measures (objective)	Pass / Fail
T1.3.2	Patient performs measures (objective)	Pass / Fail
T1.3.3	Patient performs measures (subjective)	Pass / Fail
T1.4.1	Professional monitors parameters (objective)	Pass / Fail
T1.4.2	Professional monitors parameters (subjective)	Pass / Fail
SAT Outcome		**Pass / Fail**

Table 8: Test outcome overview

3.2.10 Integration ethics and security

Introduction

This section addresses recommendations and proposed solutions to the ethical and security points of the integration.

INTER-Health

UPV-SABIEN and ASLTO5 have made the necessary legal arrangements, and UPV-SABIEN acts as the Data Processor. The Data Processor is the role in charge of managing the access to sensitive data in the Local Server in these tests. Access from UPV is through Remote Desktop via VPN, and the raw data of the (test) patients is not accessed directly. We trust the physical security of ASLTO5 infrastructure, as well as the security of the tools used to access the system remotely (the aforementioned VPN and Remote Desktop tools, and the HTTPS access and security mechanisms instilled by the Proxy for remote app data reporting and web access). Test patients have been set up to perform the tests. When the tests are finished, the test patients are not removed, but disabled, and that is taken into account for the remainder of the pilot operation. The Data Processor is the only one with credentials to access the underlying database of the Professional Web Tool holding the sensitive data. Still, some fields (like those related to authentication) of the data are encrypted and even the Data Processor cannot inspect them. All other remaining issues regarding security of the infrastructure are managed by ASLTO5 as the security of their own premises.

3.3 Open Call SAT's

3.3.1 Third Party: SensiNact

The SensiNact Gateway allows interconnection of different networks to achieve access and communication with embedded devices and/or cloud-based services. It is composed of a set of functional groups and their relative interfaces; both can be seen in the Figure 17. Below it is a non-exhaustive list of the components present in the platform.

- The *Device Protocol Adapter* abstracts the specific connectivity technology of wireless sensor networks. It is composed of the bridges associated to protocol stacks. All the bridges comply with a generic Device Access API used to interact with northbound SensiNact's services.
- The *Smart Object Access and Control* implements the core functionalities of SensiNact like discovering devices, resources and securing communication among devices services and their consumers.
- The *Consumer API* is protocol agnostic and exposes services of the Smart Object Access and Control functional to Consumers.
- The *Consumer Protocol Adapter* consists of a set of protocol bridges, translating the Consumer API interface into specific application protocols.
- The *Gateway Management* functional group includes all the components needed to ease management of devices connected to SensiNact, regardless of their underlying technologies. A Device Management API is used for this purpose. This functional group also contains the components managing cache, resource directory and security services. These management features are exposed by means of the *Manager API*.
- *Manager Protocol Adapter* allows adapting the *Gateway Management API* to the specific protocols used by different external management entities.

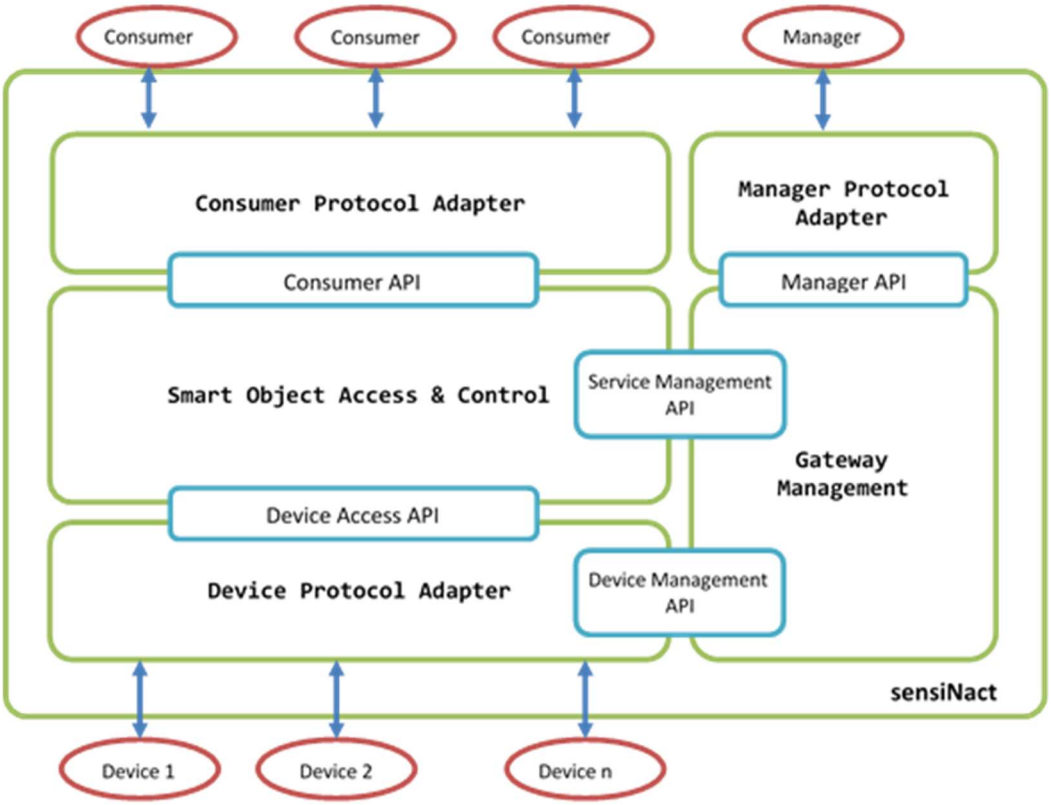


Figure 17: SensiNact Gateway overall architecture

In terms of connectivity:

On the southbound side SensiNact gateway allows to cope with *physical device* protocols and *virtual device*, allowing a uniform and transparent access to given protocol, for example ZigBee network, HTTP Restful web service. Below it is a list of supported protocols:

- **EnOcean**, concerting energy harvesting wireless sensor technology (ultra-low-power radio technology for free wireless sensors), and protocols in use to interact with those sensors;
- **BLE**, Bluetooth Low Energy , which is a WPAN, low power protocol designed mainly for healthcare or entertainment applications;
- **MQTT**, which is a machine-to-machine protocol, lightweight publish/subscribe messaging transport, useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium;
- **ZigBee** radio communication protocol for low consumption short range designed for WPAN (XBee for example);
- **CoAP** which is REST application protocol, designed to be “the HTTP for constrained networks and devices” whose concept originated from the idea that "the Internet Protocol could and should be applied even to the smallest devices," and that low-power devices with limited processing capabilities should be able to participate in the Internet of Things; it is usually used on top of a 6LoWPAN network, but it may travel regular IP networks as well (it is used by the OMA LWM2M protocol, for instance);
- **EchoNet,** Japanese communication protocol designed to create the “smart houses” of the future. Today, with Wi-Fi and other wireless networks readily available in ordinary homes, there is a growing demand for air-conditioning, lighting and other equipment inside the home to be controlled using smartphones or controllers, or for electricity usage to be monitored in order to avoid wasting energy.

On the northbound side the SensiNact gateway provides both client/server and publish/subscribe access protocols:

- **MQTT**;
- **JSON-RPC** (1.0 and 2.0);
- **HTTP RESTful**
- **CDMI**

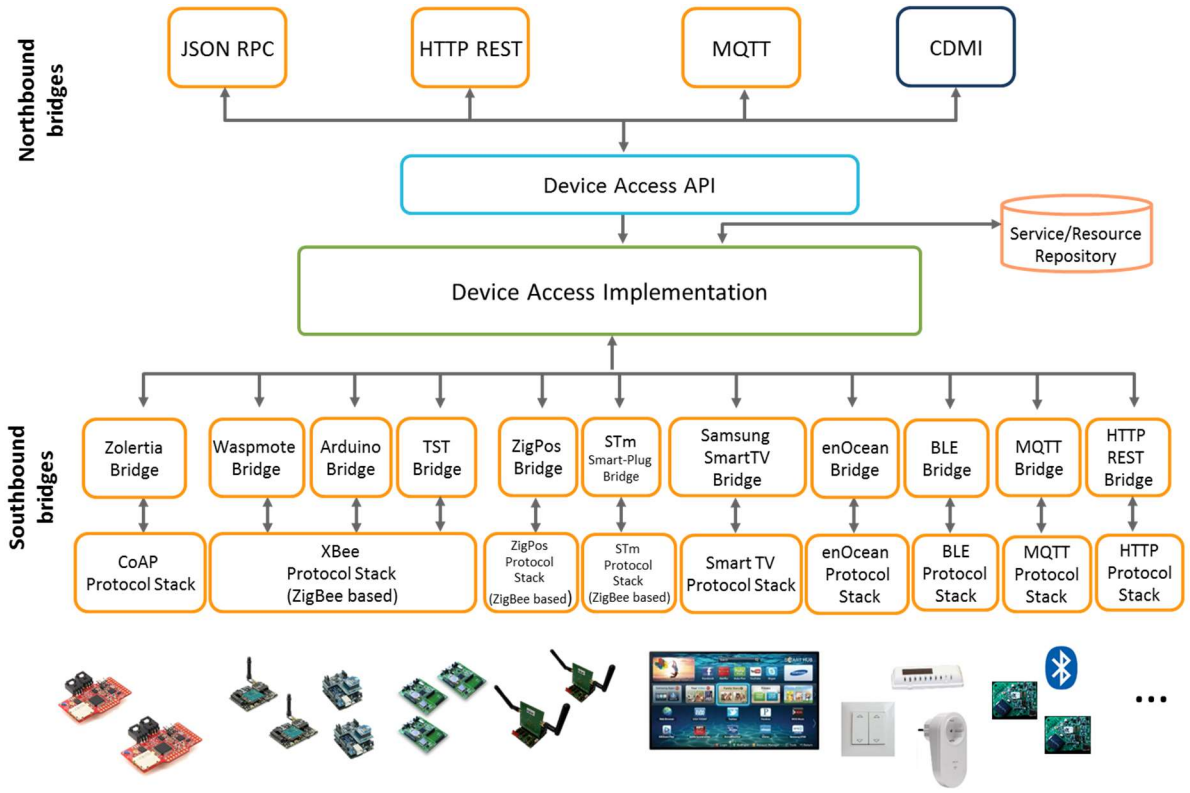


Figure 18: SensiNact Southbound and Northbound bridges

The **Smart Object Access and Control** functional group described in this previous section includes a large number of functionalities, among them:

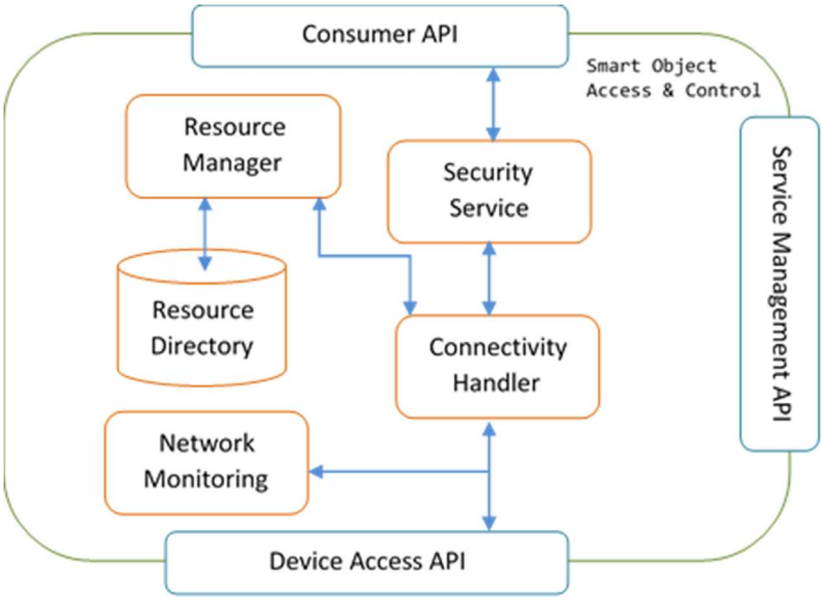


Figure 19: SensiNact Gateway internal architecture

- It handles the communication with the Consumer Protocol Adapter (REST API, JSON RPC, etc.) and IoT (and non-IoT) devices, providing URI mapping, incoming data/messages translation in an internal format and outgoing data/messages translation in Consumer format. Whenever a Consumer tries to access a resource via Consumer API, the requested URI is forwarded to the Resource Manager in order to check if a

specific resource descriptor exists or not inside the Resource Directory and to verify its accessibility status. If a resource descriptor doesn't exist, a message response with error code is returned to the Consumer API. Otherwise, the request is forwarded to the right interface. At the same time whenever response is originated from IoT device (or abstract IoT device), it will be also forwarded to its logical counterpart in order to update the resource representation in the gateway.

- It manages the subscription/notification phases towards the Consumer, if it is not handled by the targeted device (service) itself
- It supports Devices and Resource Discovery and Resource Management capabilities, to keep track of IoT Resource descriptions that reflect those resources that are reachable via the gateway. These can be both IoT Resources, or resources hosted by legacy devices that are exposed as abstracted IoT Resources. Moreover, resources can be hosted on the gateway itself. The Resource Management functionality enables to publish resources in SensiNact, and also for the Consumer to discover what resources are actually available from the gateway; SensiNact Service and Resource model allows exposing the resources provided by an individual service. The latter, characterized by a service identifier, represents a concrete physical device or a logical entity not directly bound to any device. Each service exposes resources and could use resources provided by other services. Figure 20 below depicts the Service and Resource model:

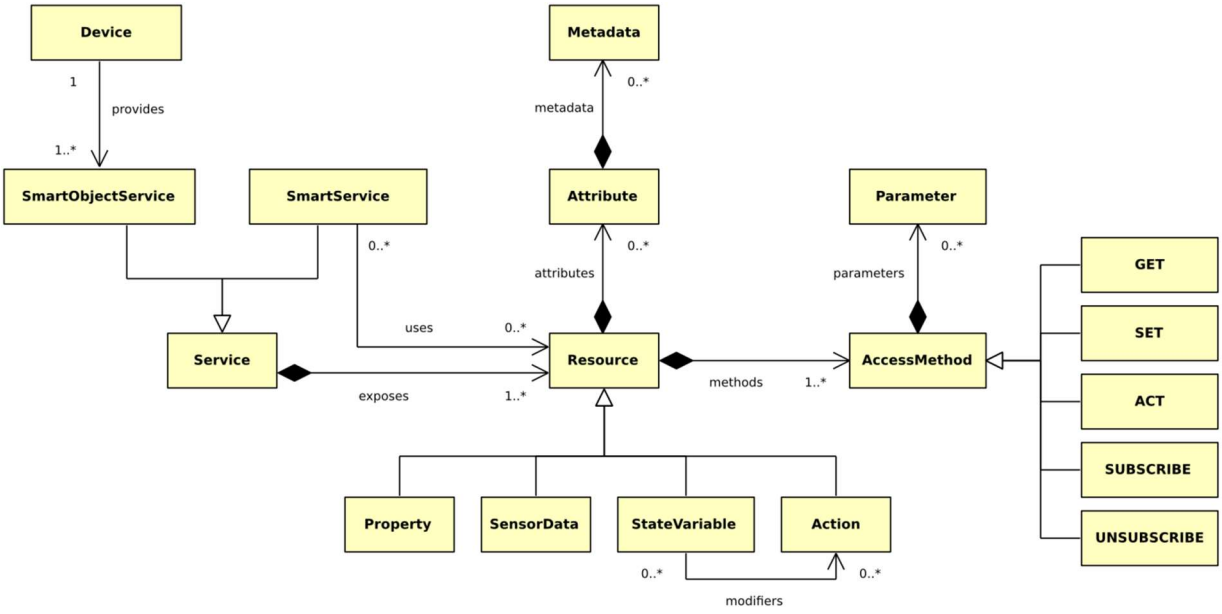


Figure 20: SensiNact Service and Resource model

The **Resource Directory** allows storing information, *i.e.* resource descriptions, about the resources provided by individual devices connected to SensiNact. It also supports resource description look up, as well as publishing, updating and removing resource descriptions.

Discovering and using resources exposed by Services is the preferred approach for avoiding using static service interfaces, thus increasing interoperability. Therefore, SensiNact Services and their exposed resources are registered into Service/Resource repositories. The gateway uses the OSGi service registry as Service/Resource repository, where resources are registered as service properties. Clients ask the Service/Resource repository for resources fulfilling a set of specified properties (defined by LDAP filters). In response, the Service/Resource repository sends clients the list of service references that expose the

requested and authorized resources. Clients can then access/manipulate the resources exposed by their selected service objects.

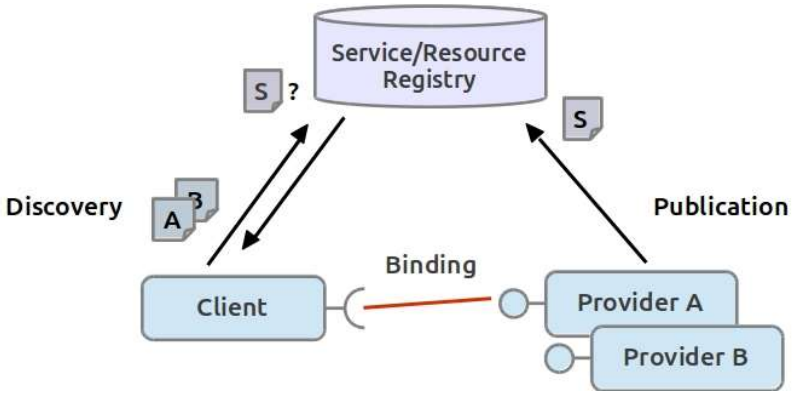


Figure 21: SensiNact's service oriented approach

Resources and services can be available for remote discovery and access using different communication protocols, such as HTTP REST, JSON-RPC, etc. Advanced features may also be supported (as semantic-based lookup). Resources can be classified as shown in Table 9, while the access methods are described in Table 10.

Table 9 Resource types

Type	Description
SensorData	Sensory data provided by a service. This is real-time information provided, for example, by the SmartObject that measures physical quantities.
Action	Functionality provided by a service. This is mostly an action on the physical environment via a SmartObject actuator supporting this functionality (turn on light, open door, etc.) but can also be a request to do a virtual action (play a multimedia on a TV, make a parking space reservation, etc.)
StateVariable	Information representing a SmartObject state variable of the service. This variable is most likely to be modified by an action (turn on light modifies the light state, opening door changes the door state, etc.) but also to intrinsic conditions associated to the working procedure of the service
Property	Property exposed by a service. This is information which is likely to be static (owner, model, vendor, static location, etc.). In some cases, this property can be allowed to be modified.

Table 10 Resource's access methods

Type	Description
GET	Get the value attribute of the resource
SET	Sets a given new value as the data value of the resource
ACT	Invokes the resource (method execution) with a set of defined parameters
SUBSCRIBE	Subscribes to the resource with optional condition and periodicity
UNSUBSCRIBE	Remove an existing subscription

The access methods that can be associated to a resource depend on the resource type, for example, a GET method can only be associated to resources of type Property, StateVariable and SensorData. A SET method can only be associated to StateVariable and modifiable Property resources. An ACT method can only be associated to Action resources. SUBSCRIBE and UNSUBSCRIBE methods can be associated to any resource type.

Protocols and data formats

The SensiNact gateway uses JSON formatted data. The resource model is a hierarchical five-tiered tree: A Device owns Services which in turn own Resources, which hold Attributes, and its *metadata*. To describe one element of this tree there is no restriction about how many sub-elements it can contain. The description of a resource and its value (result of an access method execution) are distinct from one to the other. The choice of this separation is to lighten the work of components whose work is to process the result of an access method execution, by avoiding the reification of high level data structures to only extract the content of one (or two) attribute(s).

Device Description

As only the resources are the containers of information, those which target the device are grouped in a specific service which is the administration one (« AdminService » prefixed). Those resources can be one specifying the location, or the vendor of the device, or any other data that are common to all provided services (and so resources). Formally, a device is a JSON object containing an array of services. The list of the services a device provides can be summarized or detailed. If it is summarized, only the name of the services are part of the description (otherwise each service is completely described).

```
{
  "serial-number":"fake-1234",
  "services":[
    {
      "ID":"AdminService_f1To4"
    },
    {
      "ID":"temperature_f1To4"
    }
  ]
}
```

Service Description

It gathers resources, and it references the unique identifier of the device holding it. It represents the entry point to access to resources through the OSGi context. The list of the resources a service provides can be summarized or detailed. If it is summarized only the name and the type of the resource are part of the description (otherwise each resource is completely described).

```
{
  "ID":"AdminService_f1To4",
  "properties":[
    { "device.serial-number":"fake-1234"}
  ],
  "resources":[
    {"name":"location","type":"property"},
    {"name":"owner","type":"property"},
    {"name":"vendor","type":"property"},
    {"name":"SLEEP","type":"action"}
  ]
}
```


Resource description

- The data structures are mainly nested in triplets : name, type and value;
- The type of the resource itself can be : property, variable, sensor, or action;
- The type key of a 'name-type-value' data structure (embedded in the resource description) can have a primitive as value (byte, short, int, long, double, char, boolean, [string]) or the canonical name of the java class used to reify it in the gateway;
- For each resource access method signatures are also described in a JSON array. Some of them can be shortcuts to other ones: a GET method without parameter is a shortcut to the GET method whose unique parameter "attributeName" has for value "value", for example. A parameter of an access method can be completed with the constraints which apply on it (« min », « max », « fixed », regular expression « pattern », « enumeration » of allowed values) or the JSON schema of the expected JSON object from which to reify the appropriate Java object in the gateway;
- At least two metadata exist for each attribute: the "hidden" one defining whether the attribute has to be specified in the description of the resource, and the "modifiable" one defining whether the value of the attribute can be modified by the client. By default, the « hidden » attribute is not visible in the description (if the attribute is visible that's mean that this metadata value is set to false, and if it is set to true the client is, at the end, not aware of that);
- A metadata specified as « dynamic » will be added to the result of an access method execution
- Timestamps are « epoch » formatted (number of seconds since 1970 January the first) ; To avoid the reification of high level objects to make calculations (that are at least as easy with this format). High level programming languages handle this format. It is also possible to multiply it by 1000 if handling of milliseconds is needed (what is done natively by java for example).

```
{
  "name":"temperature",
  "type":"sensor",
  "attributes":[
    {  "name":"value","type":"int",
     "metadata":[
        {"name":"modifiable",“type":"boolean","value":false,"dynamic":false},
        {"name":"timestamp","type":"long","value":1418541626,"dynamic":true},
        { "name":"description","type":"string","value":"temperature measure","dynamic":false},
        {  "name":"unit","type":"string","value":"celsius degree","dynamic":false}
      ]
    }
  ],
  "accessMethods":[
    {
      "name":"GET",
      "parameters":[
      ]  },
    {
      "name":"GET",
      "parameters":[
        { "name":"attributeName","type":"string"}
      ]  },
    {
      "name":"SUBSCRIBE",
      "parameters":[
        {  "name":"listener","type":"object",
          "schema-id":"http://fr.cea.SensiNact/subscription/listener",
          "description":"parameter value example: ‘{\"callback\":\"<uri>\"}’ "
        },
        {
          "name":"condition",
          "type":"object",
          "schema-id":"http://fr.cea.SensiNact/subscription/condition",
          "description":"parameter value example: ‘{ \"condition\":\"less\", \"value\":\"5\"}’"
        },
        {
          "name":"lifetime",
          "type":"long"
        }
```

```
        ]
      },
      {
        "name":"SUBSCRIBE",
        "parameters":[
          {
            "name":"listener",
            "type":"object",
            "schema-id":"http://fr.cea.SensiNact/subscription/listener",
            "description":"parameter value example: '{\"callback\":\"<uri>\"}' "
          } ] },
      {
        "name":"UNSUBSCRIBE",
        "parameters":[
          {
            "name":"subscriptionID",
            "type":"string"
          } ] }
  ]
}
```


Access Method result


```
{
  "name":"temperature",
  "type":"int",
  "value":22,
  "timestamp":1418541626
}
```


As it is the « default » attribute, asking for the value of a resource providing one returns a JSON formatted data structure in which the "name" key has the name of the resource as value (instead of "value" as it could have been expected).

Location resource

The location of a device (service, resource) is frequently a needed context information. By default a device always contains one (its administration service in fact), and a link to it is created in all services it provides. If needed, a link to this resource could also be created as an attribute of all resources (mainly if this location is supposed to change frequently and so to avoid to require the complete device description to update the information). Its content is not restricted (as it is the case for the others) and can so contain attributes defining longitude, latitude, altitude, a friendly name or whatever is needed to specify it (for now we are using « <latitude>,<longitude> » formatted string as value)

Device Access Control

This section explains the security definition and method into SensiNact architecture.

Security and access policies

A first level of security is reached by the way of some of available security "tools" in the *OSGi* environment: *ServicePermission* and *ConditionalPermissionAdmin.*

The ServicePermission is a module's authority to register or use a service.

- The register action allows a module to register a service on the specified names.
- The get action allows a module to detect a service and use it.

Permission to use a service is required in order to detect events regarding the service. Untrusted modules should not be able to detect the presence of certain services unless they have the appropriate *ServicePermission* to use the specific one.

The *ConditionalPermissionAdmin* is framework service to administer conditional permissions that can be added to, retrieved from, and removed from the framework.

The SensiNact gateway defines service permissions in such a way that access to the ones it provides is forbidden excepted if a specific condition is met (a SensiNact specific conditional permission). This condition being that the requirer is the SensiNact *SecuredAccess* service. Even SensiNact services have to use the *SecuredAccess* one to be able to “talk” to each other’s; Modalities of such exchanges depend on the UserProfile of the user of these services (the user can be the system itself). A *UserProfile* can be defined at each level of the hierarchical SensiNact resource model: *ServiceProvider*, Service, and Resource. Five *UserProfiles* exist for which predefined access rights are defined: Owner, Administrator, Authenticated, Anonymous, and Unauthorized.

When asking for a data structure of the SensiNact resource model, the access rights of the user are retrieved; the set of this user's accessible *AccessMethods* for the specific data structure is built and returned as part of the description object. Each future potential interaction of the user on the data structure will be made by the way of this description object. For a remote access a security token is also generated and transmitted to the user, to avoid repeating the security policy processing. A token is defined for a user and a data structure (and so it previously created description object).

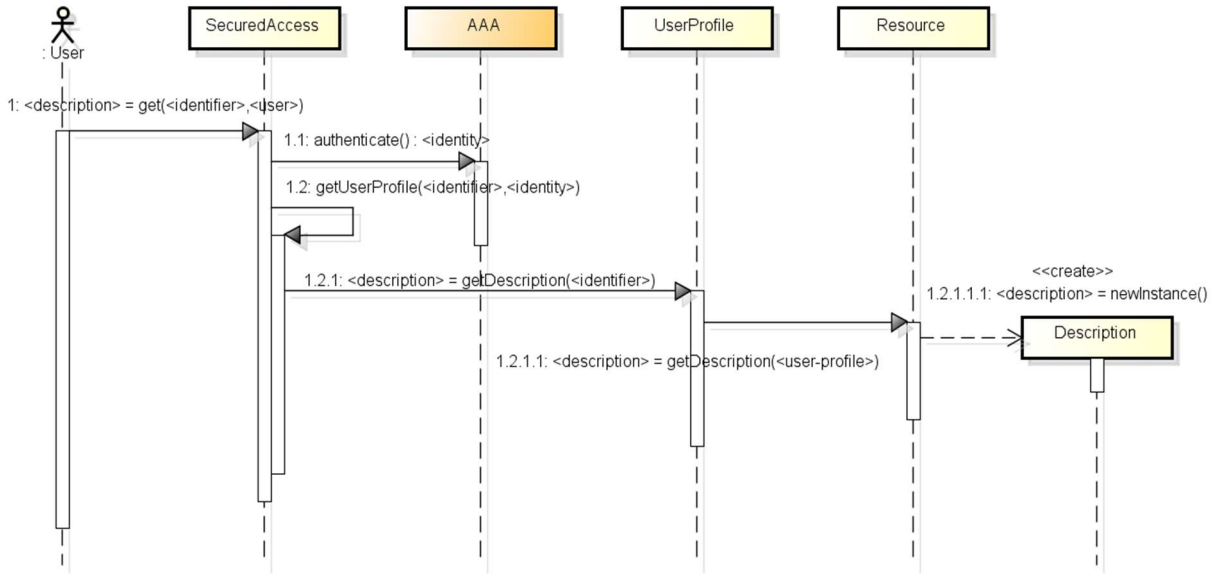


Figure 22: SecuredAccess Sequence Diagram

The Authenticaton Authorization Access service can be externalized; It is used to retrieve identity material from which it is be possible to associate a user and a SensiNact resource model data structure to a *UserProfile* (the SensiNact platform manages a database linking this identity to a *UserProfile* for a specific data structure). For all data structures for which the user has not been registered the Anonymous *UserProfile* is used by default (except if the owner of a resource has defined this default profile to another one). The internal database also gathers information relative to the minimum required *UserProfile* to access to data structures. This definition can be made at each level of the resource model, knowing that if no *UserProfile* is defined for a data structure, the one specified for its parent is used.

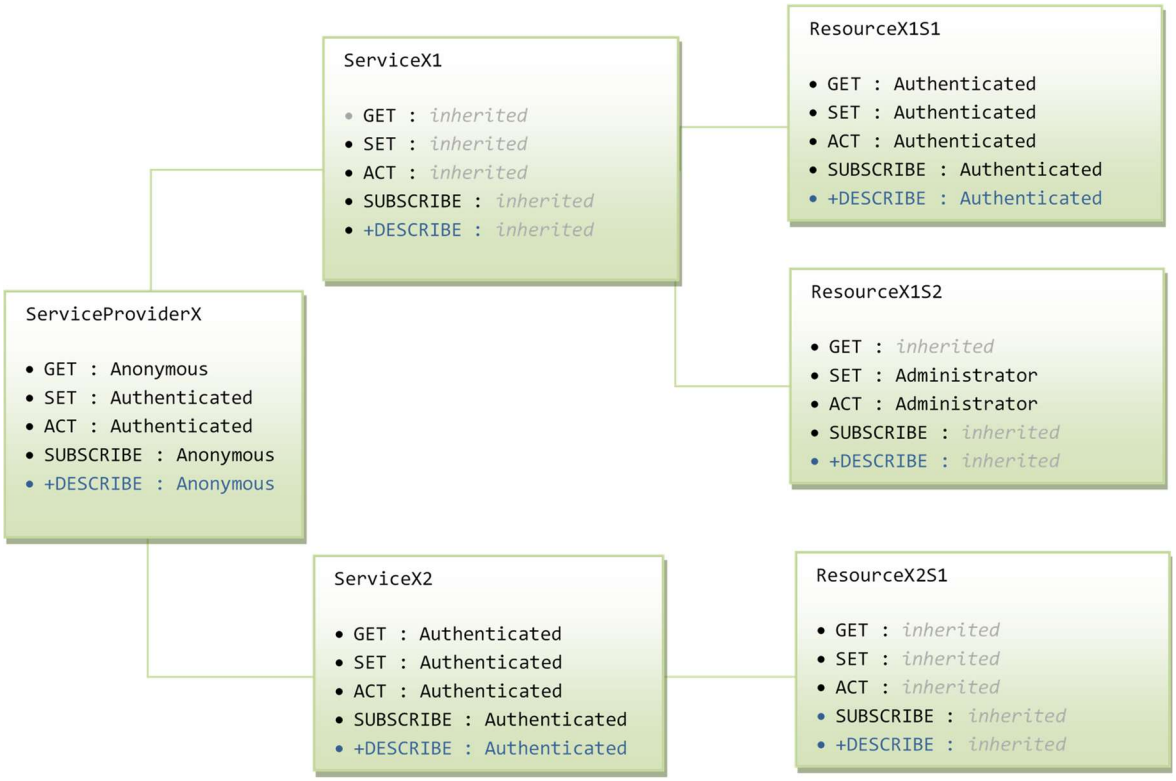


Figure 23: Access right inheritance diagram example

For example, according to the diagram shown above, a user trying to access to the ServiceProviderX for which its UserProfile is Anonymous will receive a description object in which only one Service will be referenced (ServiceX1), containing a single Resource (ResourceX1S2) providing two AccessMethods, GET and SUBSCRIBE.

Federation approach

SensiNact is based on a service-oriented approach where its functionalities are exported in terms of services, which allows easy integration of those features within the federated FESTIVAL's Experimentation Testbeds as a Service.

Application Manager

SensiNact has a component named AppManager, this component aims to create higher level applications based on the resources provided by the SensiNact gateway and which the life-cycle can controlled by the SensiNact gateway.

AppManager provides a way to develop event driven applications, i.e., based on the Event-Condition-Actions (ECA) axiom. Thus, the application is only triggered when the required events occur. Then, if all conditions are satisfied, the actions are done. Events are created from sNa sensors and the actions are performed using the sNa actuators available in the environment.

Data model and JSON format

The AppManager assumes that an application is a set of bound components. Each component processes a single function (e.g., addition, comparison, action). The result of this function is stored in a variable in the current instance of the application. The components using this result as input listen to the corresponding variable. When the variable changes, they are notified and can process their own function, leading to a new result.

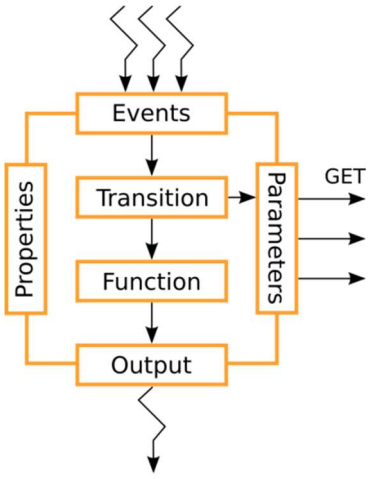


Figure 24: Architecture of a sNa component.

The component is the atomic element of an application, thus an application can consider a single component to perform an action. It holds the minimal requirements to create an ECA application:

- Events: events that trigger the process of a component. Trigger can be conditioned to a specific event or a specific value of the event (e.g., when the value of the sensor reach a threshold) ;

- Function: function wrapped in the component (e.g., addition, comparison, action). The acquisition of the parameters is realized in the transition block before the function block;
- Parameters: parameters of the function that are not available in the event (e.g., static value, sensors values).
- Output: result of the function that is stored in a variable and that triggers a new event.
- Properties: non-functional properties of the component (e.g., register the result as a new resource in sNa).

The AppManager is a sNa service provider. Thus like any other resource it provides a set of resources; in this specific case an INSTALL and an UNINSTALL resources, enabling a client to install/uninstall an application.

A sNa application is described using a JSON file. We developed a specific Architecture Description Language (ADL) to describe the components used in an application and the bindings between the components.

The following JSON code example corresponds to the code of a single component:

```
{
"events": [{
"value": "resource1",
"type": "resource",
"condition": {
"operator": ">=",
"value": 100,
"type": "integer"
}
}],
"function": "function_name",
"parameters": [{
"value": "ON",
"type": "string"
}],
```

This component specifies that when the resource1 is greater or equals to 100, the function_name is called with the string parameter "ON". The result of the function is stored in the output_name variable and triggers a new event that may be used by others components.

The supported types are:

- Primitives types: integer, boolean, long, double, float, string. This is used to described a static variable;
- Resource type: resource. This is used to refer to a resource. If this is set in the JSON Event section of the JSON, a SUBSCRIBE is done on the resource. If this is done in

any JSON Parameters section, a GET is done on the resource and returns the current value;

- Variable type: variable. This is used to refer to the output of a previously processed component;
- Event type: event. This is used to refer to the value of the event that triggers the function. This type is never used in the condition of the JSON Event section.

Here after is a synthesis of the type that can be used in the different parts of the JSON file.

	Primitive types	Resource type	Variable type	Event type
In event type	No	Yes	Yes	No
In event/condition type	Yes	Yes	Yes	No
In parameters type	Yes	Yes	Yes	Yes

Table 11: Types used in the JSON component

The AppManager supports the validation of the JSON files against a JSON schema. Schemas exist in the plugins and may be used by the developers of the applications.

Architecture

The AppManager is designed to be used as any sNa service provider. Thus it provides an "Install" and an "Uninstall" resource, enabling a client to install/uninstall an application. These resources are accessible using different bridges, such as any actuators.

The AppManager architecture is also designed to easily add new functions and to handle the lifecycle of applications in order to perform checks.

Plugins

Plugins enable to add new function to the AppManager. New plugins require to implements the mandatory interfaces Java interface to be found in the OSGi registry and thus be used by the AppManager. The AppManager is currently supporting the following functions.

Table 12: Functions supported by the plugins of the AppManager

PLUGIN	FUNCTIONS SUPPORTED
BASIC PLUGIN	various operators (e.g., equals, greater than, lesser than, different), addition, subtraction, division, multiplication, modulo, concatenation, substring, ACT and SET methods on resources
CEP PLUGIN	after, before, coincides, average, average deviation, count, max, min, median, standard deviation, sum

Lifecycle

The AppManager provides a lifecycle to manage the applications. It enables to process various checks during different steps of the lifecycle of the application (e.g., ADL consistency, resource permissions). The first step is to install the application, i.e., send the ADL of the application. If there is a problem, the AppManager returns an error. Once the application is installed, it can be started and its state changes to "Resolving". If there is a problem during this step, the application enters in the "Unresolved" state. Otherwise, the application is active until it is stopped or an exception occurs.

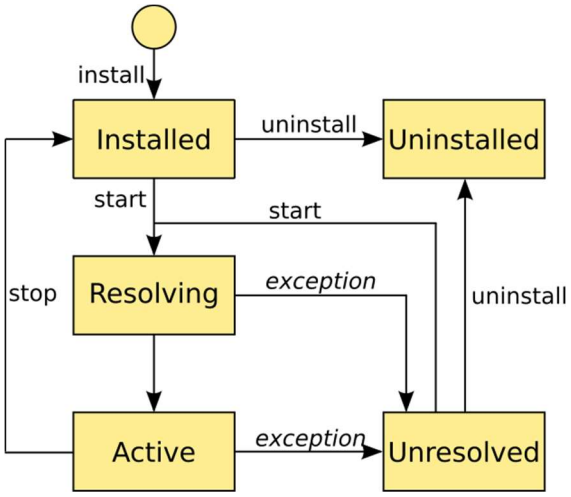


Figure 25: Lifecycle of an application.

Instances

The AppManager allows multiple instances of the same application to run in parallel. When an event occurs, the InstanceFactory of the application instantiates a new set of components and passes the event to the first component. The number of instances can be set in the application properties. If, there is more events than available instances, events are stored and processed when an instance ends.

SensiNact Studio

SensiNact Studio allows an easy interaction with the IoT devices and the creation of applications. The Studio is based on the Eclipse platform and built as a rich client platform application. The Graphical User Interface (GUI) is developed using the views mechanism from Eclipse. Thus, it proposes views for browsing devices, locating devices on a map and interacting with them, i.e., getting value from sensors or performing actions on actuators. The Studio is also targeted to ease the creation of IoT application following the Event-Condition-Action (ECA) pattern.

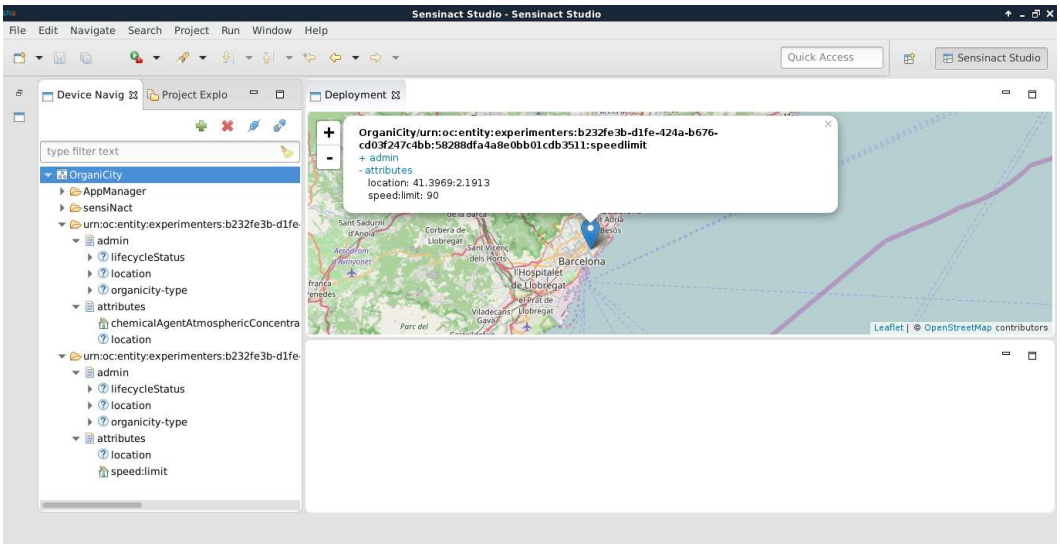


Figure 26: SensiNact Studio Graphical User Interface.

The GUI (Figure 26) includes different views: navigator, deployment, properties views, as well as a Domain Specific Language (DSL) editor.

Browsing devices

Before users can use the studio for managing devices and applications, they need to connect a SensiNact gateway. This action is performed by clicking on the plus sign icon on the device navigator. Then, gateway information have to be provided (Figure 27).

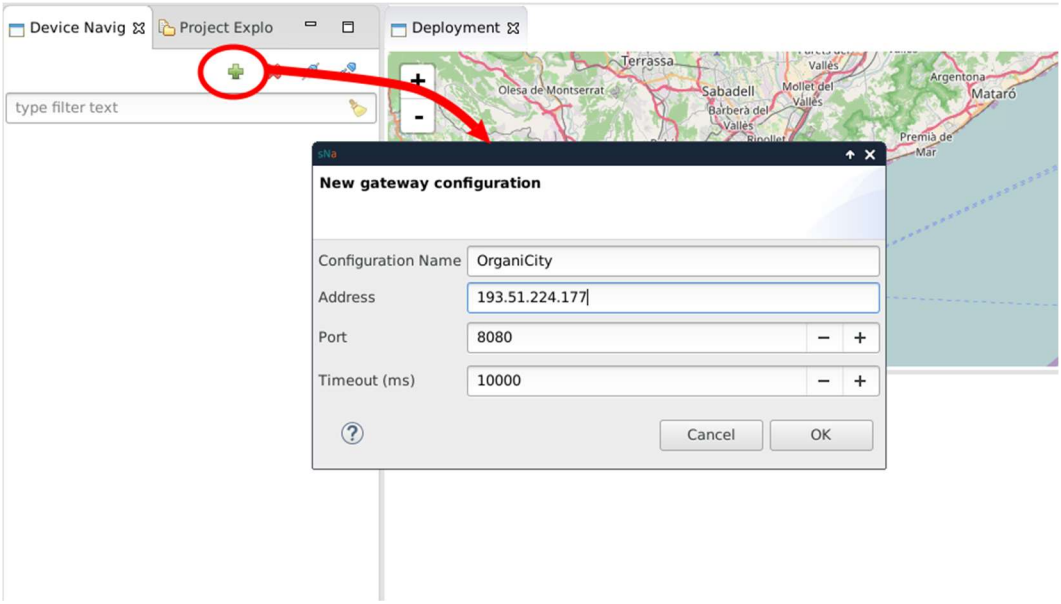


Figure 27: Gateway configuration.

Once the information have been provided and the dialog validated, the Gateway is added to the Navigator View. To display and browse the available devices imported by this gateway, connecting to it is needed. This action is performed using the connect button (Figure 28).

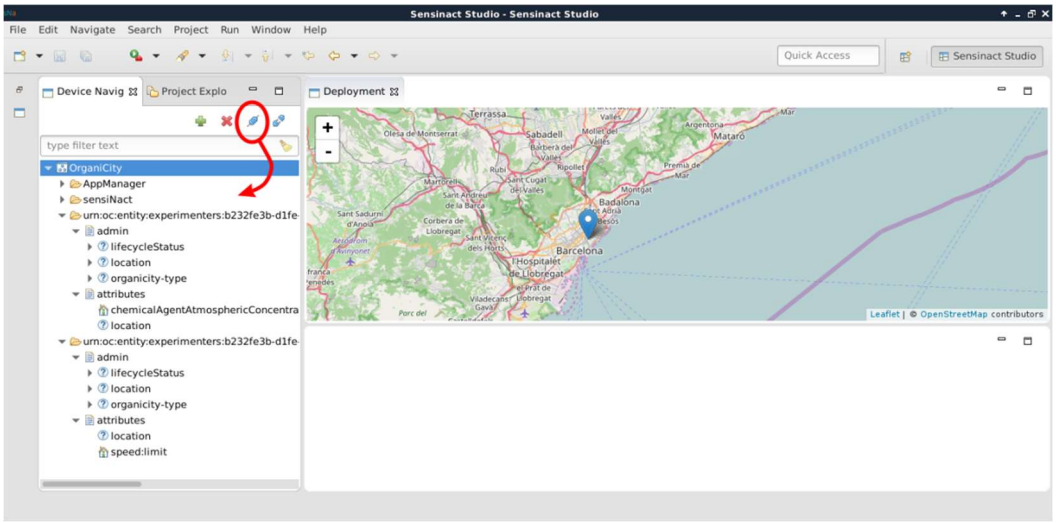


Figure 28: Gateway connection.

The device Navigator View is then populated, and pin points are displayed on the map. By clicking on attributes names, it is possible to get the current value for the considered

attribute. It is also possible to see attributes values on the map, clicking on the pin points (see Figure 28).

Application creation

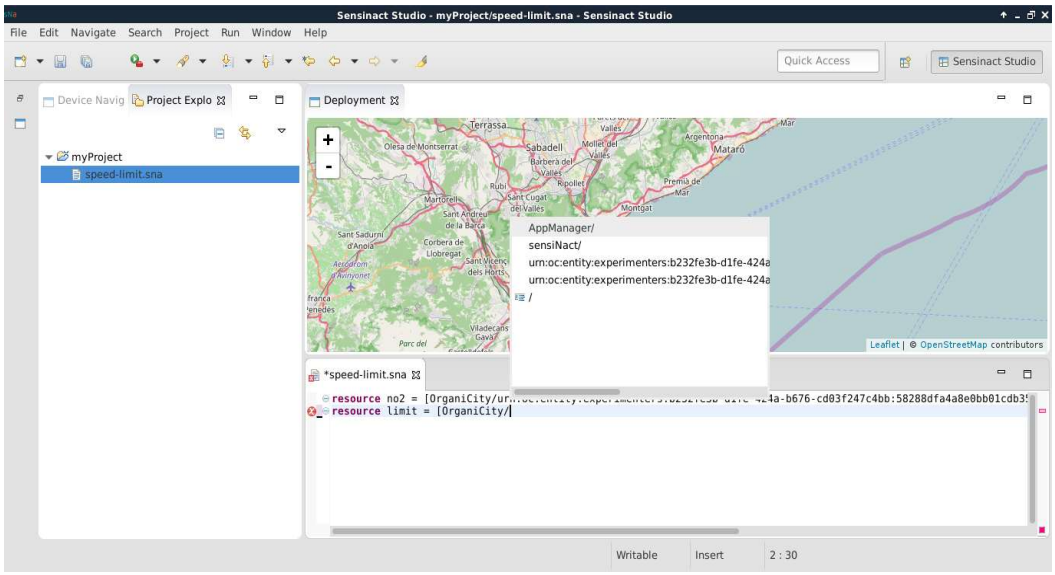


Figure 29: Application creation.

The SensiNact Studio allows the creation of applications to be executed on the gateway. Creating an application is performed by writing a script using a dedicated syntax, and deploying this script to the gateway.

On Figure 29, a project has been created on the project explorer view (on the left). In this project, a script named *speed-limit.sna* has been created, and is being edited. As the figures shows, the editor provides code highlighting (some keywords are displayed in a special font), code completion (with popups) and a syntax validator which displays red crosses on the script margin in case of error.

The dedicated syntax, a Domain Specific Language, is composed by the following blocks:

- The shortcut block: each resource is accessible through a unique URI, which can be quite long. This block aims at creating shortcut for the next blocks.
- The event block: the developer defines on which resources the application is triggered. When an event is thrown and is a valid trigger, the conditional block is executed.
- The conditional block: once the application has been triggered, and before any action can be executed, the data from the resource has to satisfy the conditional block. The keyword for this block is if followed by the conditions to be validated.
- The actions statements: if the conditional block is satisfied, actions are performed in the order that they are listed. The actions can be physical actions on actuators or virtual actions such as changing the format of a data using a mathematical function. The available actions, also named functions, are listed below:
- Basic functions: addition, subtraction, division, multiplication, modulo, string concatenation, substring, various operators (e.g., equals, greater than, lesser than, different), ACT and SET methods on SensiNact resources.
- Complex Event Processing functions: after, before, coincides, average, average deviation, count, max, min, median, standard deviation and sum.

Table 13 shows the basic structure for writing a script.

[**resource**<resource>]+	Shortcut block, which must contains at least one statement.
on<events>+	The event block, lists the events triggering the script. At least one event must be provided.
[**if**<condition>**do**]+ [<actions>]+ [**else do**]? [<actions>]?+ **end if;**	The conditional block, which lists actions to be performed based on conditions.

Table 13: SensiNact Domain specific language basic syntax

Once the script has been written, it can be deployed to the gateway where it will be executed. This is performed using a right click on the script file (see Figure 30).

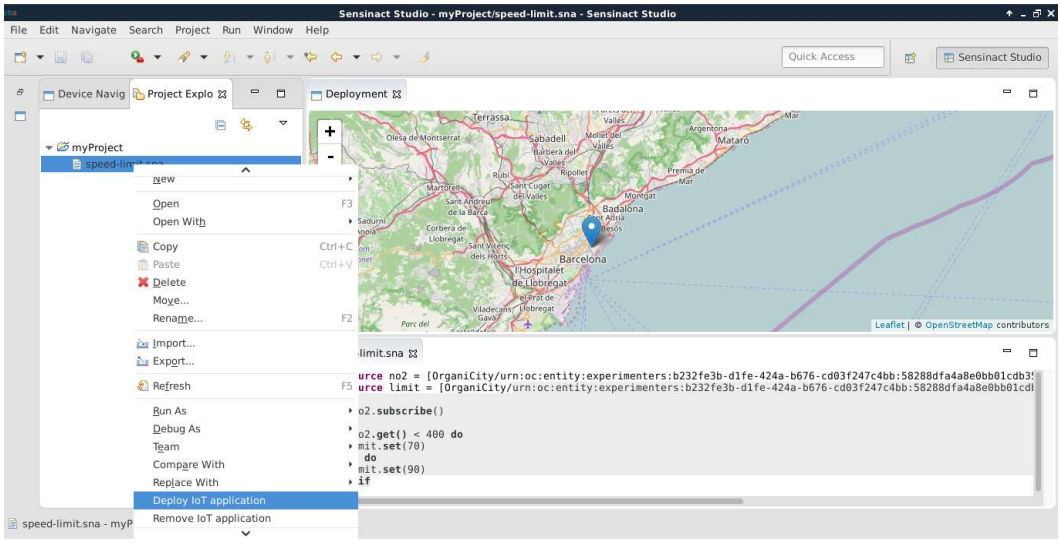


Figure 30: Application deployment.

Application monitoring

After the application has been deployed, a new set of resources is automatically created under the AppManager device. You can browse those resources into the Device Navigator View (Figure 31).

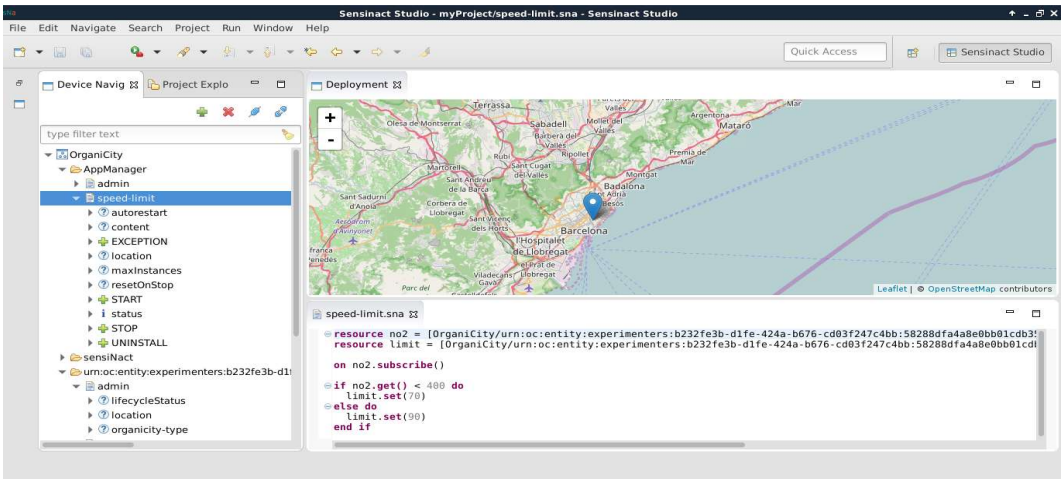


Figure 31: Application management resources.

First of all, a new service is created with the name of the sNa file (without the extension). In our example, it is speed-limit. This service representing the application always contains a standard set of resources (Table 14).

Table 14: Application management resources

Resource	Type	Description
autorestart	property	In case of failure, decides if the application should be automatically started again
content	property	Script file content
EXCEPTION	action	Deprecated
location	property	GSP location which can be used if it makes sense
maxinstances	property	Number of parallel instances which should be started
resetOnStop	property	On Stop, decides if the generated resources by the application should be destroyed or kept
START	action	Starts the application
status	state variable	Current status of the application: START/STOP/...
STOP	action	Stops the application
UNINSTALL	action	Removes the application

To start the application, simply double click on the START resource. This will launch the start action, which will run the script.

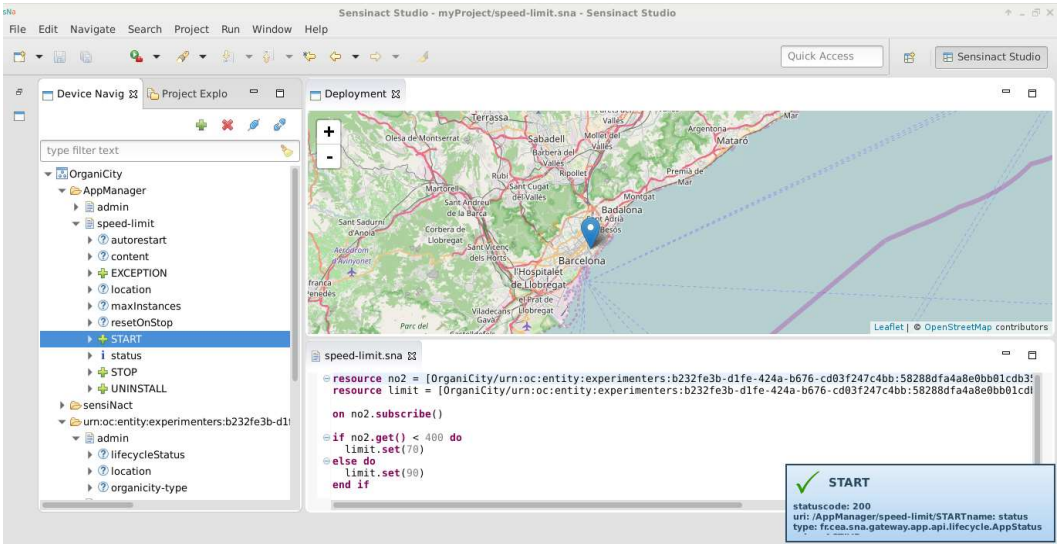


Figure 32: Application start-up.

Figure 32 shows that the application is up and running on the server. The studio can be used to check if the application has the expected behaviour, by querying the resources. The studio can also be shut down, since the applications are executed on the gateway.

Reusable components

SensiNact is easily portable to any hardware platform and supports a large number of protocols that allows its easy integration and reuse in various IoT environments.

SensiNact Studio Web

Presenting the information that is absorbed by the gateway is as important as organizing and aligning the concepts on the backend of gateway itself.

Thus in order to provide a simple interface to watch over the sensors integrated in the SensiNact Gateway, each instance of SensiNact offers a mobile-compatible interface that enables the end user to read the information of the gateway and execute standard device calls in the platform.

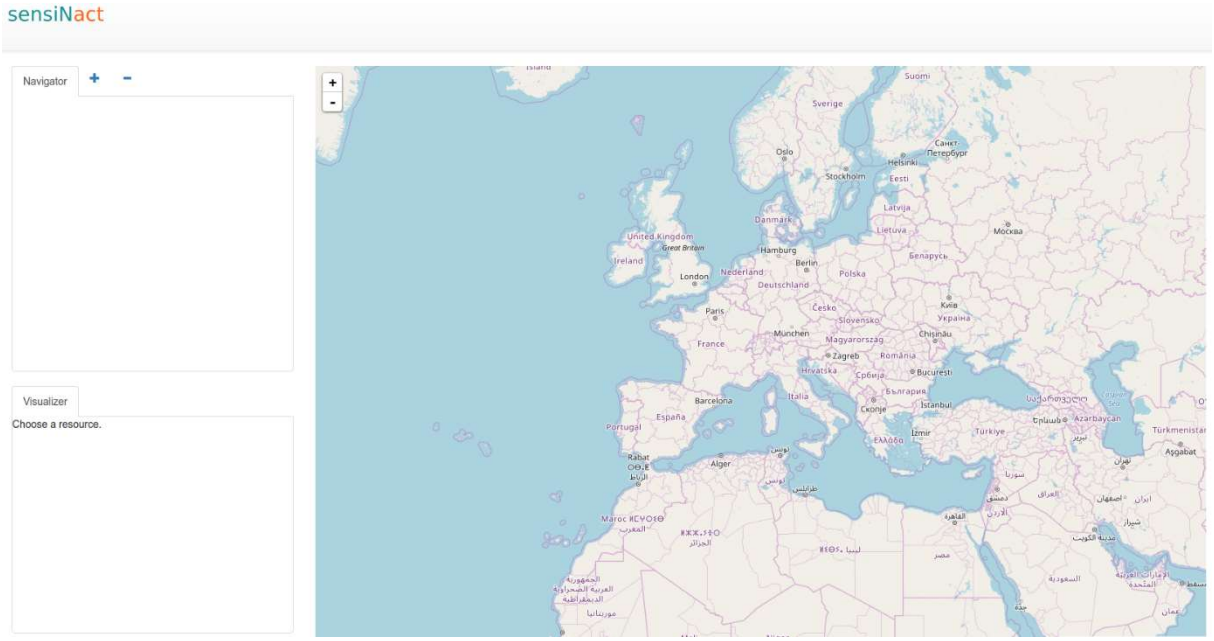


Figure 33 StudioWeb initial screen

The initial screen Figure 33 is composed of 3 areas: *Navigator, Map* and *Visualizer*, they are situated on upper-left, right and bottom-left respectively. Navigator is where the available gateway will be display along with the entire resource hierarchy, meaning displaying the gateway above all the sensors (known as providers), the sensors with all the services available and each service with their respective resource, all display in a tree in which the top most element is the selected gateway.

Within the *Navigator* area it is possible either add a new gateway using the “+” sign, or disconnect (and remove) the gateway using the “-” sign, noticing that this function only make sense if you are connected to one. The connect options can be seen in Figure 34.

This is where the client will input the information about the address of the gateway he/she wishes to be connected to and the actual port.

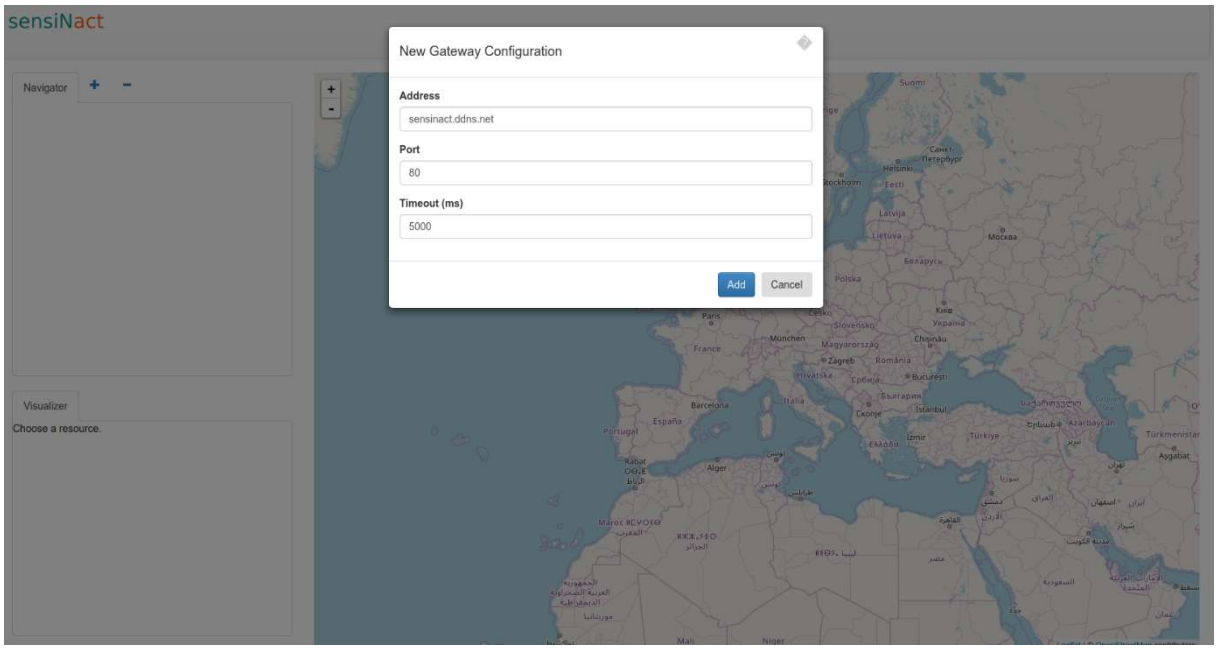


Figure 34: StudioWeb connect

After a successful connection, you will be able to see the gateway along with the tree of devices/services/resource attached to the gateway Figure 35.

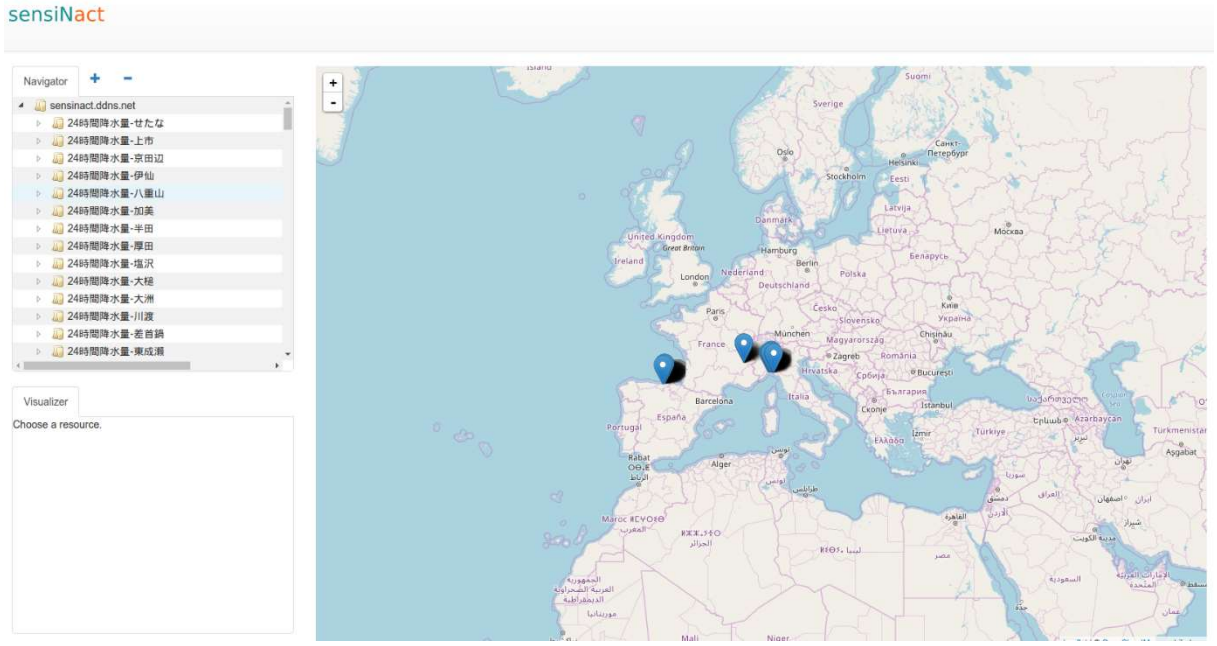


Figure 35: StudioWeb gateway content

All sensors, known as well as providers, are display on the upper-left part of the interface, if a specific sensor is selected, the Map area will lead to the geographic position of that sensor (if that information is available) and all the information of services and resources contained in that sensor will be display directly on the Map area, see Figure 36.

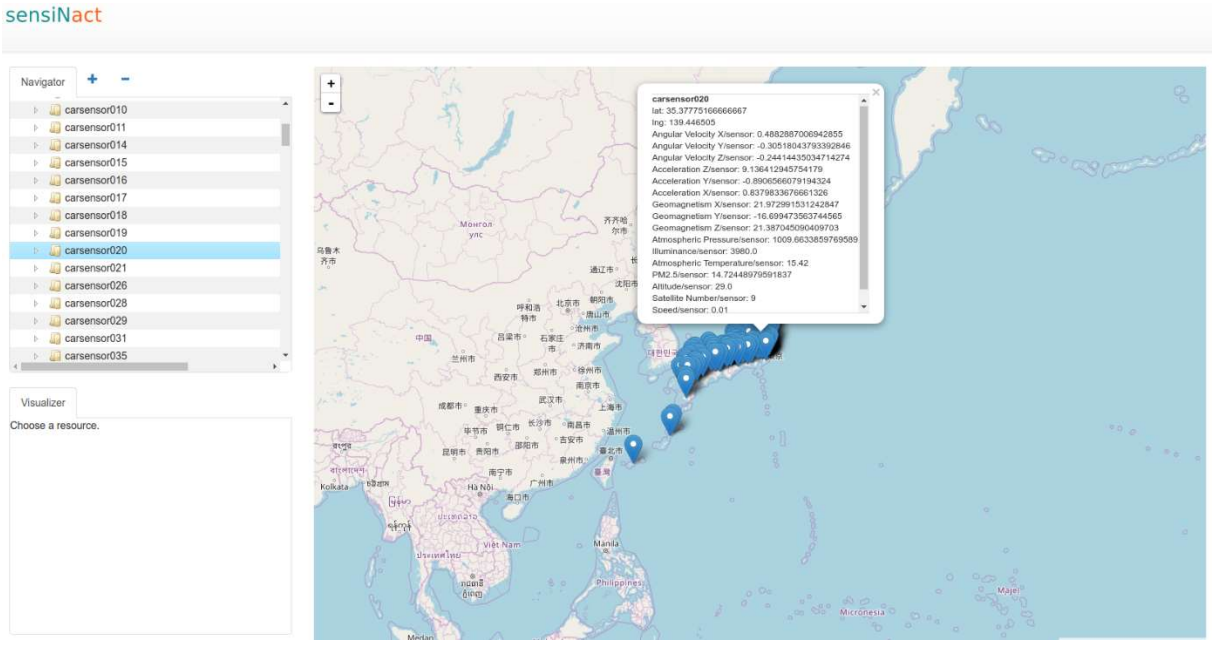


Figure 36 StudioWeb: Sensor data

Once the connection is established with an active gateway, the StudioWeb is receiving updates notifications from the gateway for the new data, which may include new devices attached to the platform, disconnected devices or updated sensor data notifications. According to the capability of the gateway (processing capability) you may be disconnected in order to preserve the resource consumption on the back-end.

After been disconnect voluntarily from the gateway a message will be display on the upper-right part of the screen, see Figure 37

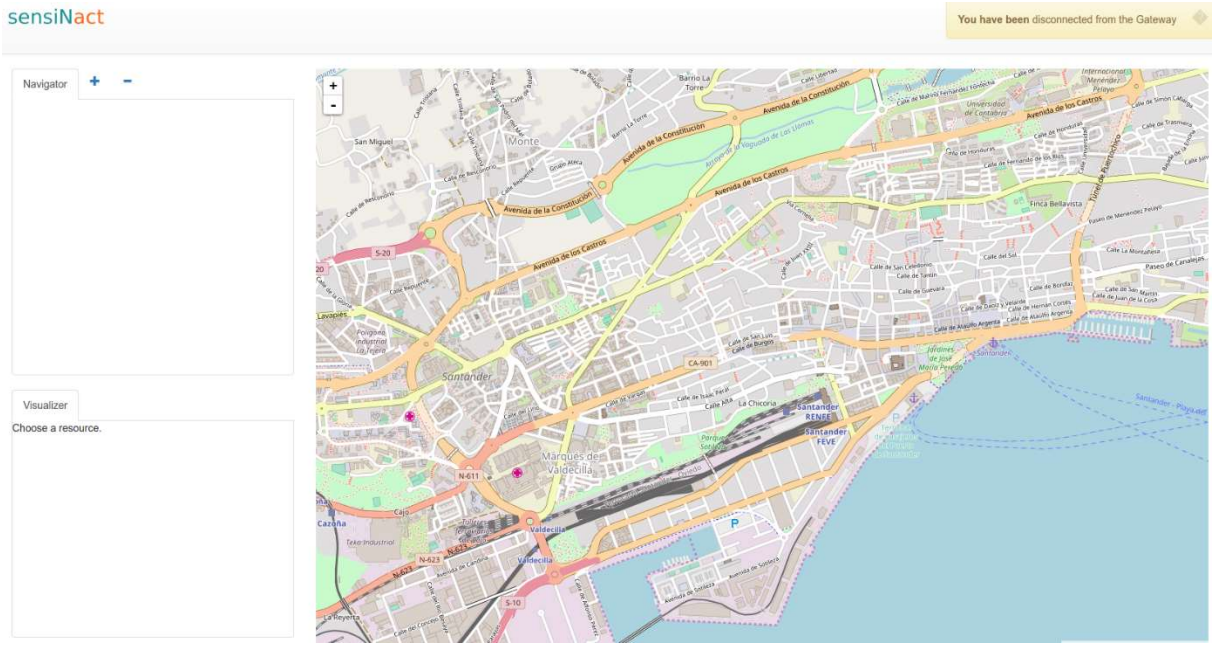


Figure 37: StudioWeb gateway disconnection

3.3.1.1 Integration of IoT framework

In order to establish a Site Acceptance Test Plan for Eclipse SensiNact platform it is important to understand how the INTER-IoT platform integration was done through the bridge implementation.

INTER-IoT Eclipse SensiNact support was done using the Bridge layer, The bridge layer was conceived to allow external IoT platforms to create a generic adaptation schema designed to support a large number of architectures, enabling this bridge coordinate and create a pattern to access the MW2MW component. The bridge layer can be seen below

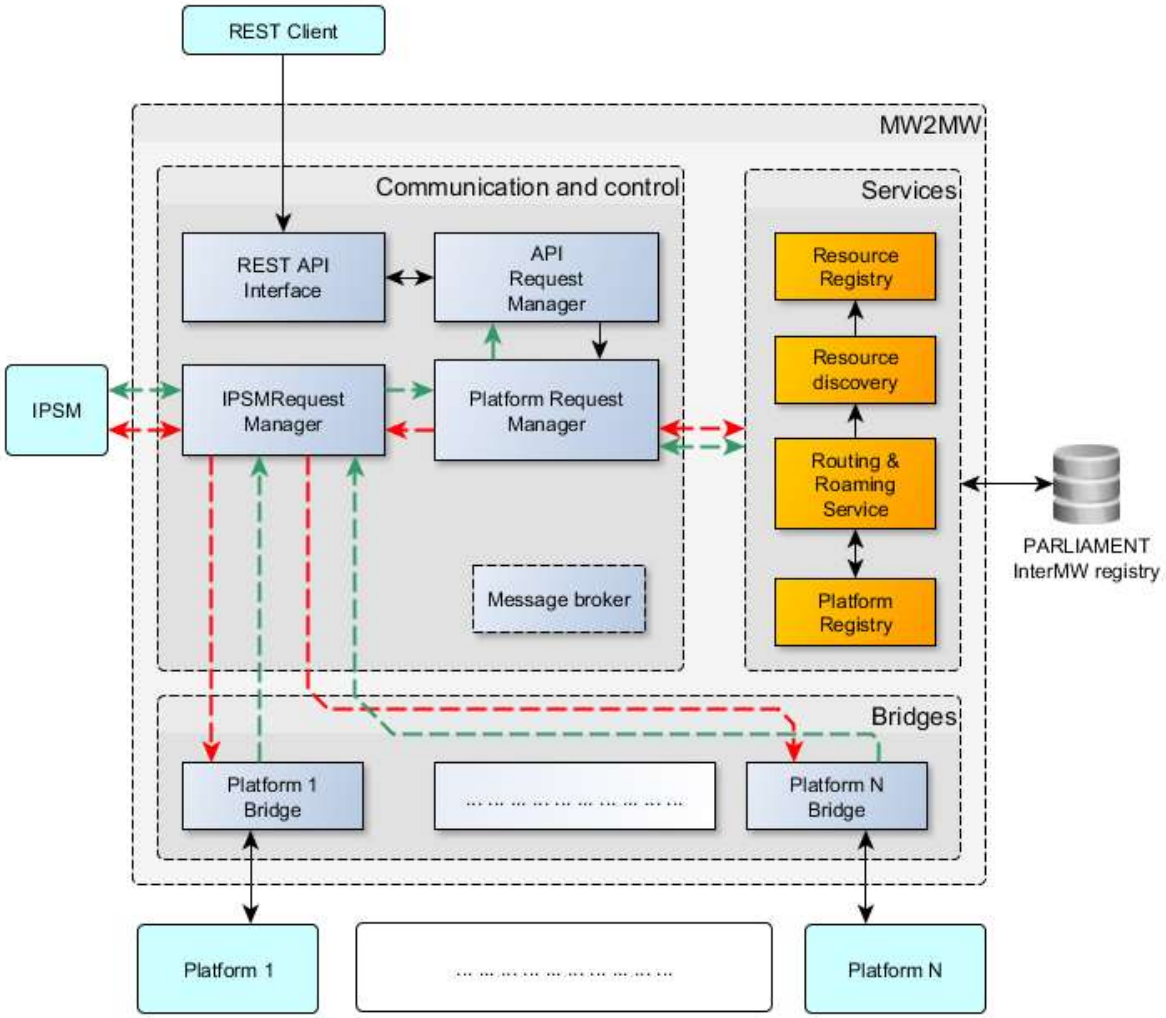


Figure 38: INTER-IoT: Middleware to Midldeware communication architecture

As seen, the bridge uses mainly the IPSM Request manager to communicate with INTER-IoT platform. IPSM will instantiate, initialize and receive and retransmit the information sent by the external supported platforms into the IPSM.

All the bridges are conceived based in an ontological model, this model can be any format supported by Jena. Although the Jena supports several ontological representation formats, Jena is only the framework that will encloses the instance of a chosen ontology, this choice will be done independently by each platform, which means that mainly three approaches can be followed, either adopt an existing ontology that is the most convenient for the platform

targeted domain, adopt Generic Ontology for IoT Platform GOIoTP [https://docs.INTER-IoT.eu/ontology/owl/GOIoTP.owl] or using a custom ontology.

The Eclipse SensiNact is situated in among other bridges in the Bridge layer, in order to enable INTER-IoT client to be able to use a supported platform, he must specify some parameters that will be use by the bridge to give an appropriate data source for the platform client.

The SensiNact bridge communication was designed according to the sequence diagram shown in the Figure 40.

Once the bridge is installed in the INTER-IoT platform, the SensiNactBridge constructor is called by the IPSM; the SensiNact bridge receives its configuration data and allows the bridge to receive the instantiation configuration info like the endpoint address of SensiNact, version of SensiNact, the port in which it should be reached and other parameters, all the options available for the bridge configuration can be seen in https://git.INTER-IoT.eu/INTER-IoT/INTER-MW_bridge_SensiNact/

At this point the bridge will call a factory Figure 39 that will instantiate the proper SensiNactAPI according to the parameters sent to the bridge. Until this point the bridge do not establish any communication with the endpoint sent to the bridge.

Once the INTER-MW receives a platform registration request from the INTER-MW REST API (represented in the Figure 41) the SensiNactAPI instance will establish the connection with the WebSocket channels in SensiNact.

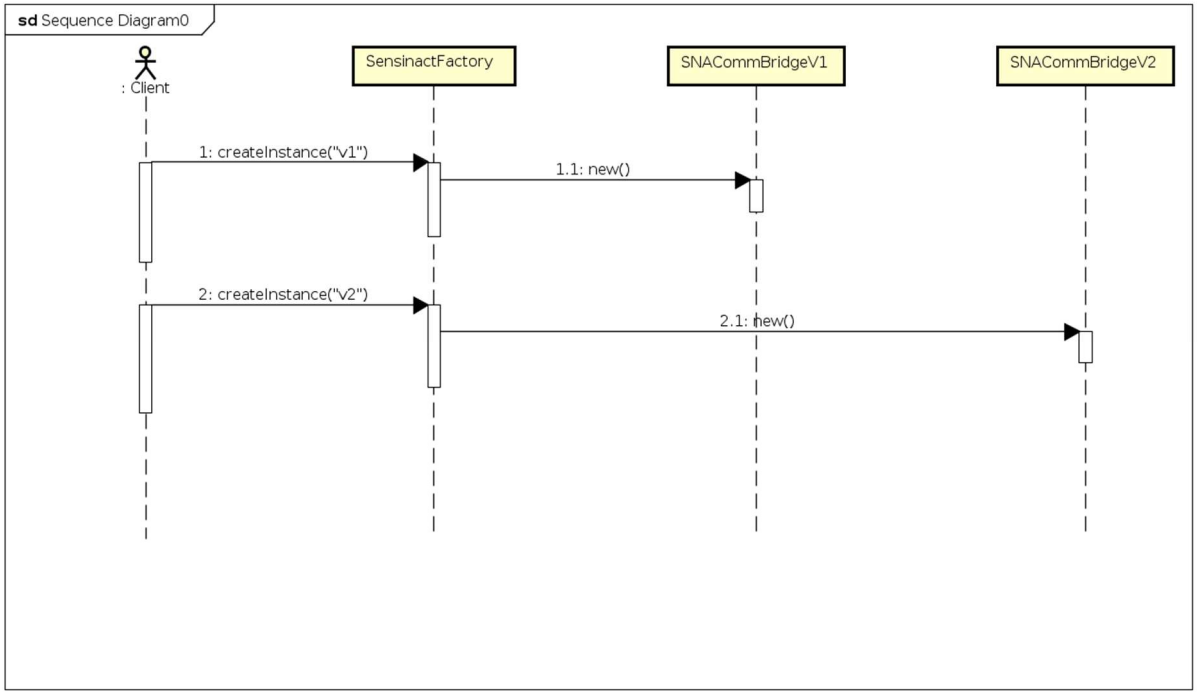


Figure 39: SensiNactAPI factory for multiple SensiNact versions

Depending on the version requested on the configuration file, there may be one or two websocket communication established by the bridge. If the version requested is the version “v1” (pre-eclipse legacy version), only one WebSocket channel will be opened, case the version “v2” (current eclipse version) is requested two Websocket channels will be required, one for the regular data update notification (callback), and the other channel is used to create virtual device into SensiNact.

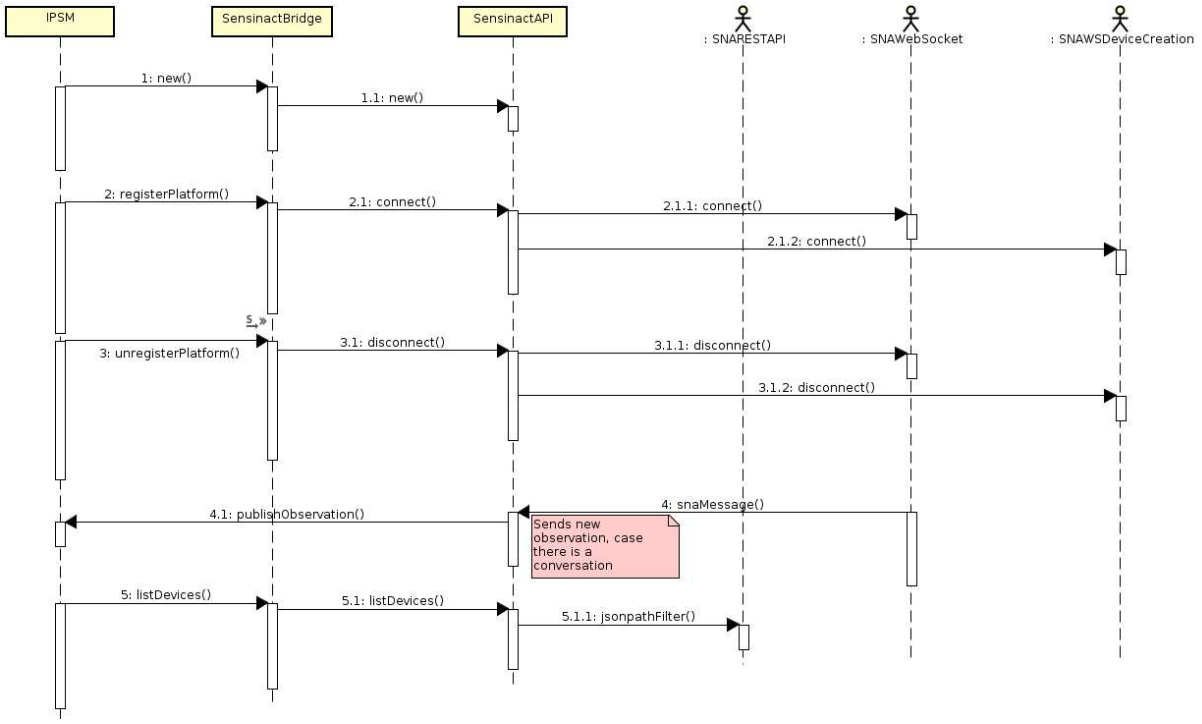


Figure 40: SensiNact INTER-IoT bridge design

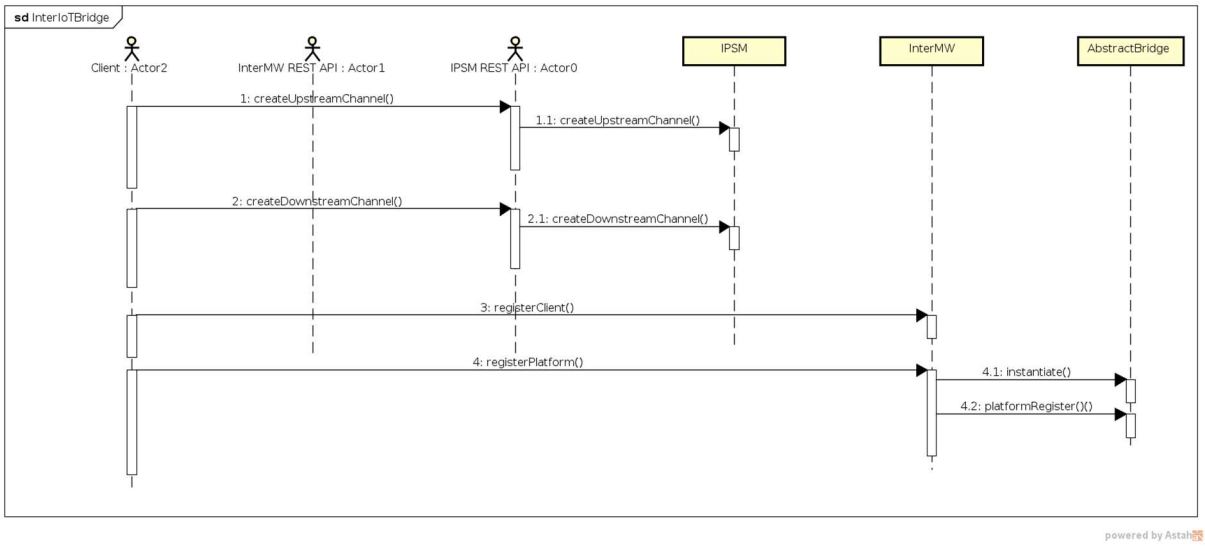


Figure 41: INTER-IoT: Sequence diagram for generic INTER-IoT MW2M bridge activation

Above as we can see, once the bridge is supported in the platform the INTER-IoT client must perform INTER-IoT REST calls in order to configure, initialize and instantiate his access to the targeted bridge.

The client first will need to access IPSM RESTful interface in order to create downstream and upstream communication channels, along with the channel creation, according to the architectural decision taken by the responsible for the bridge platform the user can as well specify the alignments that will be necessary for the requested bridge.

Once the channels are create in two direction, now the user must specify a Client access for the RESTful interface that will *de facto* access the data provided by the bridge.

Thus the Client will now register a client via INTER-MW client, the client register will be used on further calls to the API in order to represent the properties expressed by the interface client.

Once the client is registered successfully the next step is to call the procedure on the INTER-MW that will instantiate the platform for a given client. As input it is necessary for the RESTful interface client to indicate the ClientID created earlier.

Once those steps were successfully executed, now the user can finally have access to the data provided by bridge requested by using the proper calls, in order to have more information about those calls, the REST API documentation must be checked.

3.3.1.2 Deliverables and version overview

The following table contains a deliverable list which need to be signed of before SAT testing commences.

ID	Description	Check
Documents		
1	Validation and Test reports of the IoT system components	
2	Validation and Test reports of the Pilot system components	
Tools		
7	Wireshark	
8	CURL	
9	Java SDK	

Table 15: Deliverable checklist

The following table shows the software components and version of which the system release version 1.0 consists of.

ID	Description	Version	Check
SensiNact			
1	Eclipse SensiNact Gateway	v2.0	

Table 16: Component version overview

3.3.1.3 Requirements, scenarios and use cases

The following table provides an overview of the relation between the requirements and the test(s) that validate their implementation.

ID	Description	Covered by
1	Extensibility (feature evolution)	T1.1, T2.1
2	API for third-party developers	T1.1, T2.1
3	API REST	T3.1, T3.2
4	Open Source	T2.1
5	Documentation	T1.1

Table 17: Requirements vs. test mapping

3.3.1.4 Test environment

Introduction

To test the functionality of the integrated SensiNact in combination with the IoT framework representative test hooks in the system are needed. This chapter will describe this environment and the used hardware, software, tools and platforms.

Test environment

This paragraph describes the test setups, hooks and probes used in the system used during this SAT.

Test tools, hooks and probes

This paragraph describes the test setups, tools, hooks and probes used in the by this document defined tests.

A test setup defines a setup used for a test, and describe the used system parts, interface etc.

Tools define the used COTS tooling that will be used during testing e.g. Logic analyzers, packet sniffer like e.g. Whireshark, developed scripts, etc.

Hooks are used to inject data or control parts of the system used to get the system to handle a needed scenario. This can be hidden, debug or service interfaces which contain the needed functionality.

Probes are used to gather logging from parts of the system under test to provide system feedback and to prove that the system is handling the scenario as it should.

The following paragraphs will define the used setups, tools, hooks and probes which will be referred to by the test descriptions.

Test description

Documentation is available online

The platform **documentation** MUST be available publicly on the internet regardless of subscription or particular registering process.

T1 Check documentation available

ID	T1.1
Test	Verify that the platform documentation is available online on http://wiki.eclipse.org/SensiNact
Type	Manual verification
Setup	No particular setup is required
Start	System does not need to be started
Req.	Not Applicable
Input	Not Applicable
Output	Check that the documentation is available
Logs	Not Applicable
Outcome	Pass / Fail

Source code MUST be available online

The platform source code MUST be available publicly on the internet regardless of subscription or particular registering process.

T2 Check that the code source is available

ID	T2.1
Test	Check that it is possible to obtain the source code of the platform
Type	Manual verification
Setup	No particular setup is required
Start	System does not need to be started
Req.	Not Applicable
Input	Not Applicable
Output	Not Applicable
Logs	Not Applicable
Outcome	Pass / Fail

Check that the platform has a RESTful API

The platform MUST provide a RESTful API to reach its sensors, allowing external developers to easily use the platform information on their application.

T3.1 Check that the RESTful service is available

ID	T3.1
Test	Check that it is possible to obtain the source code of the platform
Type	Manual verification
Setup	• Point your browser to http://projects.eclipse.org/proposals/eclipse-SensiNact • Download the code-source • Compile the code-source • Activate bridges HTTP, Rest and Swagger
Start	Start SensiNact
Req.	Not Applicable
Input	Not Applicable
Output	Reach the interface http://localhost:8080/SensiNact/providers, at list one provider named “SensiNact” should be seen.
Logs	Not Applicable
Outcome	Pass / Fail

T3.1 Check that the swagger interface is available

ID	T3.2
Test	Check that it is possible to access swagger interface of the platform
Type	Manual verification
Setup	• Download SensiNact Binary • Activate bridges HTTP, Rest and Swagger • Start SensiNact • Reach the interface http://localhost:8080/swagger/index.html
Start	System does not need to be started
Req.	Not Applicable
Input	Not Applicable
Output	Swagger interface should be reachable
Logs	Not Applicable
Outcome	Pass / Fail

3.3.1.5 Test outcome overview

The following table will provide an overview of the test result of all the performed tests in this SAT.

Test	Description	Outcome
T1.1	Documentation is available online	Pass / Fail
T2.1	Source code MUST be available online	Pass / Fail
T3.1	Check RESTful service is available	Pass / Fail
T3.2	Check Swagger interface is available	Pass / Fail
SAT Outcome		**Pass / Fail**

Table 18: Test outcome overview

3.3.1.6 Integration ethics and security

The security, trust and privacy component of SensiNact is in charge of reifying and maintaining the data structures, holding access rights to existing resources as well their level of trust according to who is providing them. The main principle to ensure trust and privacy is first to keep as little private data as possible in the system. It is also to anonymize those that will be used for analysis. For example, the anonymization starts by breaking the link between the identifier of the user in the system and the identifier to which analysed data records are linked to, in manner of preserving all possible statistic calculations but disallowing the simple association between a data record to a unique person. Whatever the anonymization mechanisms are, some private data will remain into the system because of their use in the case of social features. In manner of being compliant with the new European rules relative to private data known as GDPR, we apply its seven principles in the private data management:

- *Loyalty and lawfulness of treatment:* the processing of data must be carried out for legitimate reasons and its use must conform to the regulations and transparent.
- *Collection of consent:* the consent of the individual whose data is collected and processed must be express, that is, it must result from a positive act. It cannot be the result of a silence or a checkbox that is checked by default.
- *Purpose of treatment*: The data that is being processed must be treated for a specific, explicit and legitimate purpose. This purpose cannot be altered: The data collected for a treatment cannot be processed for a purpose other than that for which the persons have given their consent.
- *Proportionality*: It is only possible to collect data that is adequate, relevant and not excessive to the purpose of the treatment.
- *Data security*: The controller has the obligation to undertake all the necessary measures to ensure the security of the data and to avoid their disclosure to unauthorised third parties.
- *Accuracy of data*: the data collected and processed must be accurate. To do this, it is better to promote a regular update of the data in question. Measures must be put in place to ensure that inaccurate data is erased or rectified. The most appropriate way to comply with this principle is to set up an application that allows users to modify their own data.
- *Deleting data*: The data that is no longer needed should be deleted. The shelf life is variable and depends on the nature of the data and the purposes pursued.

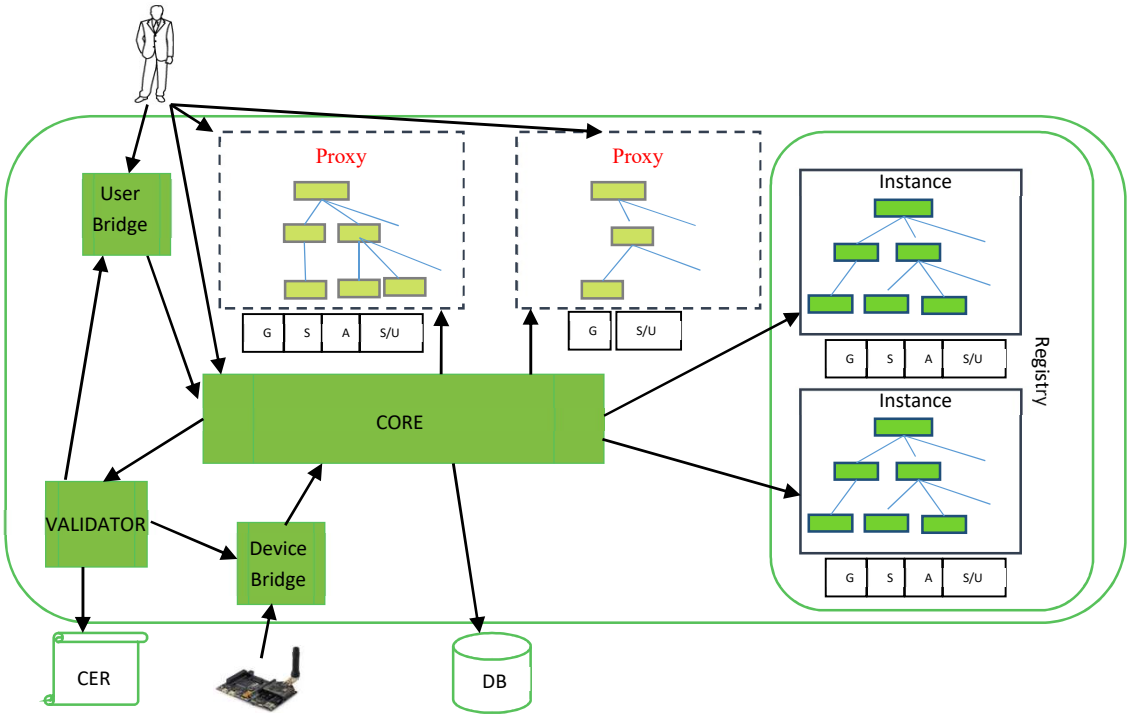


Figure 42: Securing private data

In particular to respect the data security principle, we use the same model that we use for city entities: we reify an instance of the SensiNact inner system's service model to host the private data, only accessible by the user. It means that sharing personal data result only in an explicit sharing request coming from the user itself.

Permission to use a service is required in order to access resources regarding the service. Untrusted clients should not be able to use the services, event not aware of their presence. The resource access layer defines service permissions in such a way that access to the ones it provides is forbidden except if a specific condition is met (a specific conditional

permission). The condition is that the client is an instance of “Secured Access Service”. The clients can be one of the following five profiles:

- Owner,
- Administrator,
- Authenticated,
- Anonymous,
- Unauthorized.

A UserProfile can be defined at each level of the hierarchical resource model: ServiceProvider, Service, and Resource.

When asking for a data structure of the resource model, the access rights of the user are retrieved; the set of this user's accessible access methods for the specific data structure is built and returned as part of the description object. Each future potential interaction of the user on the data structure will be made by the way of this description object. For a remote access, a security token is also generated and transmitted to the user, to avoid repeating the security policy processing. A token is defined for a user and a data structure (and so it previously created description object).

Implementation

Two levels of security are implemented. A first security level handled by the underlying service framework that hosts the SensiNact platform, which is the OSGI security framework, to secure installation and activation of software modules. The second security level tackles the user accesses to registered resources with respect to authentication (login/password authentication) and thus is provided as a service to the application layer.

OSGi[2] Framework which provides a first level of with its ServicePermission and ConditionalPermissionAdmin services. The ServicePermission is a module's authority to register or use a service:

- The register action allows a module to register a service on the specified names.
- The get action allows a module to detect a service and use it.

Permission to use a service is required in order to detect events regarding the service. Untrusted modules should not be able to detect the presence of certain services unless they have the appropriate ServicePermission to use the specific one.
The ConditionalPermissionAdmin is framework service to administer conditional permissions that can be added to, retrieved from, and removed from the framework.

In addition to the database managed by the Security & Dependability functional block, used to authenticate a user and to retrieve its identity in the system, the SensiNact platform manages an internal database allowing to link this identity to a UserProfile for a specific data structure. For all data structures for which the user has not been registered the Anonymous user profile is used by default (except if the owner of a resource has defined this default profile to another one). The internal database also gathers information relative to the minimum required UserProfile to access to data structures. This definition can be made at each level of the resource model, knowing that if no UserProfile is defined for a data structure, the one specified for its parent is used.

[2] https://www.osgi.org/

For example, according to the figure below, a user trying to access to the ServiceProviderX for which its UserProfile is Anonymous will receive a description object in which only one Service will be referenced (ServiceX1), containing a single Resource (ResourceX1S2) providing two AccessMethods: GET and SUBSCRIBE.

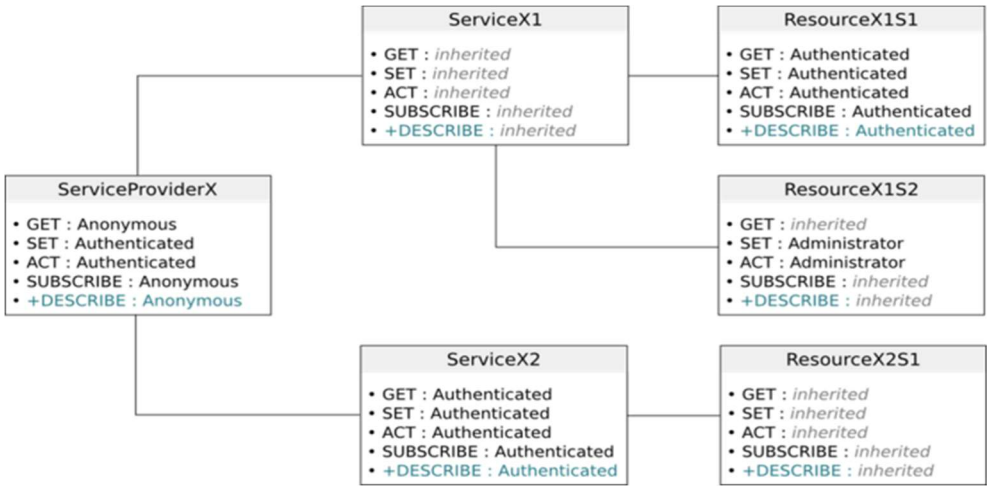


Figure 43: Example of security inheritance of service provider

More details about the implementation of the security mechanism implemented for the City resource access layer (aka SensiNact platform) can be found at:
https://wiki.eclipse.org/SensiNact/Gateway_Security

3.3.2 Third Party: OM2M

The system under test has four main components (see figure 1):

- the Middleware to Middleware (MW2MW) layer, which includes our bridge,
- the Inter-Platform Semantic Mediator (IPSM), which includes our new alignments),
- the Eclipse open source platform called Open Machine To Machine (OM2M)
- the Sensolus company's cloud, to which four "stickntrack" tracers are connected.

MW2MW or Middleware to Middleware, also called INTER-MW (inter-middleware) is the layer with which any IoT platform can communicate, provided a bridge is available for that platform type. The OM2M Bridge is our contribution to MW2MW. The Java interface, between the OM2MBridge class and the MWM2W layer, allows for the creation of any number of bridges connected to different OM2M platforms and allows the selection of the right bridge when sending a message to a particular platform.

The IPSM or Inter-Platform Semantic Mediator is a translation block able to modify the payloads of messages from MW2MW according to a so-called alignment to make them understandable for the platform they need to reach or for the MW2MW layer itself. The alignments developed in this large open call project are between the Generic Ontology for IoT Platforms (GOIoTP), being the central ontology developed for the INTER-IoT project by SRIPAS and a pseudo ontology based on XML (and oBIX for content instances) format of OM2M resources.

OM2M or Open Machine to Machine is an Eclipse open source project aiming at developing an implementation of the oneM2M standard. The core of the platform is the Common Service Layer that allows any node to be able to reach any other node or application via multiple protocol bindings (Hyper Text Transfer Protocol (HTTP), Constrained Application Protocol (CoAP), Message Queueing Telemetry Transport (MQTT) protocol or via specialized proxies. The functions of the Service layer are represented by resources accessed through Uniform Resource Identifiers (URIs). The actual data exchanged between the applications and nodes can take any format. The version of OM2M used for our tests largely corresponds to the main branch of OM2M.

Sensolus is a company providing Sigfox enabled tracers that can be easily attached to any asset that needs to be localized. The data from registered tracers can be accessed through their cloud via Application Programmer Interface (API) calls. An Interworking Proxy Entity (IPE) has been added in our OM2M platform to retrieve the location data regularly and copy it in the OM2M Infrastructure Node.

The tested interfaces are:

-The "MW2MW to IPSM" interface, for the selection of the right alignment needed for the semantic translation of the messages. The IPSM can maintain channels capable of performing translation between two supported ontologies. Once the alignments "OM2M to GOIoTP" and "GOIoTP to OM2M" are uploaded, the MW2MW layer must be able to request the IPSM to translate the message payloads (if relevant for the message type) automatically.

- the "MW2MW to bridge" interface: once an instance is created to communicate with a given platform, outgoing messages destined for that platform must be routed through the bridge seamlessly for mapping the message to OM2M requests and for doing syntactic translation.

- The “bridge to OM2M” interface: an HTTP client and server handle all requests and responses, in both direction. Those components and the underlying protocol layers needed to connect the physical devices running OM2M and MW2MW must keep functioning correctly. The bridge also reports this type of errors differently from syntactic or semantic errors.

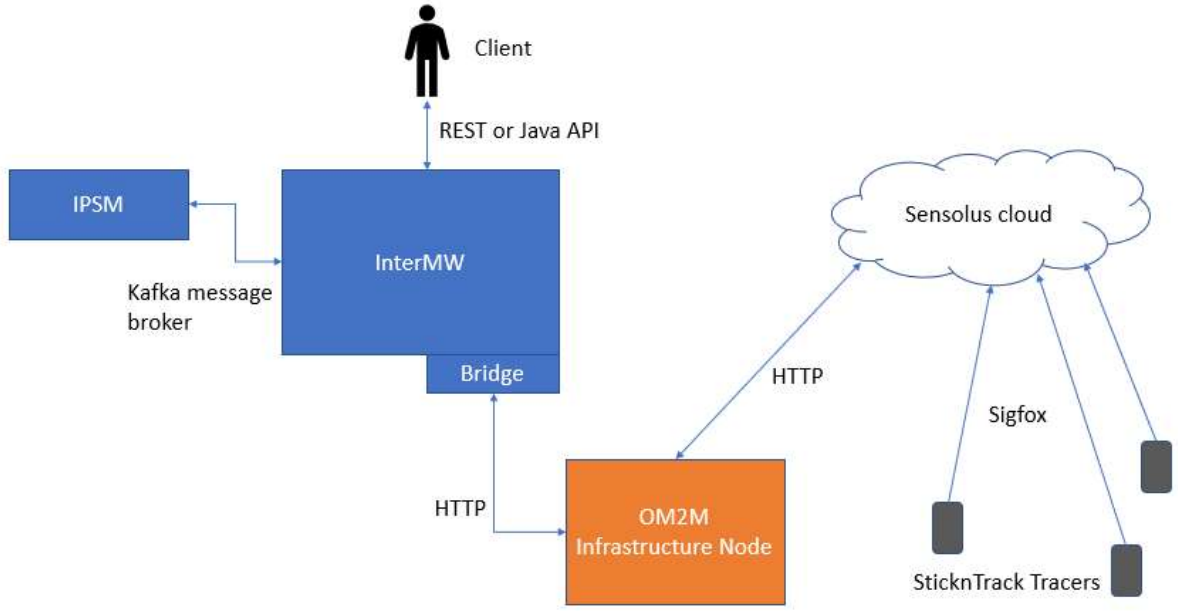


Figure 44: system overview

Integration of IoT framework

The integration in the pilots consist in the tracing of valuable assets in the port of Valencia thanks to the Sigfox-based tracers. Those tracers push their position to the cloud platform of the company Sensolus (later referred to simply as “Sensolus cloud”), which provides an API for data collection. That API is used by a server located in the Vrije Universiteit Brussel acting as Infrastructure Node of a oneM2M platform, which in turn makes some of its resources accessible through an Inter-Middleware bridge.

IoT component	Interfaces	tests
“Strickntrack” tracers	Sigfox connectivity	OK
Sensolus cloud	Proprietary API	T1.1.1
OM2M Infrastructure node	Proxy for Sensolus cloud	T1.1.1
	HTTP server/client	T1.1.1, T1.1.2
OM2M Bridge in MW2MW	HTTP server/client	T1.1.1, T1.1.2
	Java interface	T1.1.1, T1.1.2
MW2MW layer	Java interface	T1.1.1, T1.1.2
	REST API	T1.1.1, T1.1.2

Table 1: tested components and interfaces

3.3.2.1 Deliverables and version overview

The following table contains a deliverable list which need to be signed of before SAT testing commences.

ID	Description	Check
Documents		
1	Validation and Test reports of the IoT system components	
2	Validation and Test reports of the Pilot system components	
3		
Hardware		
4	Sensolus stickntracks	
5	Server	
6		
Tools		
7	Wireshark	
8		
9		

Table 2: Deliverable checklist

The following table shows the software components and version of which the system release version 1.0.0 consists of.

ID	Description	Version	Check
IoT back-end server			
1	INTER-MW layer	V1.7.0	
2	Inter Platform Semantic Mediator (IPSM)		
3	OM2M Infrastructure Node CSE	V1.1.0	

Table 3: Component version overview

3.3.2.2 Requirements, scenarios and use cases

The following table provides an overview of the relation between the requirements and the test(s) that validate their implementation.

ID	Description	Covered by
4	Alignment with other IoT architectures, especially with AIOTI	T1.1.1, T1.1.2
14	Platform independency	T1.1.1
15	Support of common IoT communication protocols	T1.1.1
16	Inter-connection support	T1.1.1
26	Remote device control	T1.1.1
42	Heterogeneous information representation	T1.1.1
45	Connectivity not based on HW identifiers	T1.1.1
51	API for data publication	T1.1.1
52	API REST	T1.1.1, T1.1.2
53	Location of sensor and measurement is included in semantic models	T1.1.1
127	Availability of sensor data	T1.1.1
154	Time stamped event recording	T1.1.1
178	Inter Platform Semantic Mediator provides data and semantic interoperability functionality	T1.1.1, T1.1.2

179	Inter Platform Semantic Mediator supports platform communication	T1.1.1, T1.1.2
180	Syntactic and semantics interoperability - Data format and semantics translation	T1.1.1, T1.1.2
183	IoT Platform Semantic Mediator does not store sensor data	T1.1.1
194	Provide exchange of virtual objects between platforms	T1.1.1
201	Monitoring and provision of subscription services between different platforms	T1.1.1
220	Ontology mapping among most prominent standards	T1.1.1, T1.1.2
224	Location semantic support for mobile smart objects	T1.1.1
234	Provide connectors to middleware standards	T1.1.1
237	API Middleware for interoperability between different platforms	T1.1.1, T1.1.2
238	Virtualization of common objects in middleware layer	T1.1.1
242	Object/Device virtualization	T1.1.1
254	Each data unit is identified univocally	T1.1.1
255	A common data model compatible with all platform-specific models is shared	T1.1.1
256	Each device has a unique INTER-IoT identifier	T1.1.1, T1.1.2
270	API allows subscription to data streams/queues	T1.1.1
272	Stores recent data for recovery	T1.1.1
283	Manage a sensor or actuator	T1.1.1

Table 4: Requirements vs. test mapping

The following table provides an overview of the relation between the scenarios and the test(s) that validate their implementation.

ID	Scenario name	Covered by
5	Monitoring of containers carrying sensitive goods	T1.1.1
33	Heterogeneous Platforms Methodology-driven Integration	T1.1.1, T1.1.2

Table 5: Scenario vs test mapping

3.3.2.3 Test environment

Introduction

To test the functionality of the integrated Inter-OM2M in combination with the IoT framework representative test hooks in the system are needed. This chapter will describe this environment and the used hardware, software, tools and platforms.

Test environment

This paragraph describes the test setups, hooks and probes used in the system used during this SAT.

Test tools, hooks and probes

This paragraph describes the test setups, tools, hooks and probes used in the by this document defined tests.

A test setup defines a setup used for a test, and describe the used system parts, interface etc.

Tools define the used COTS tooling that will be used during testing e.g. Logic analyzers, packet sniffer like e.g. Whireshark, developed scripts, etc.

Hooks are used to inject data or control parts of the system used to get the system to handle a needed scenario. This can be hidden, debug or service interfaces which contain the needed functionality.

Probes are used to gather logging from parts of the system under test to provide system feedback and to prove that the system is handling the scenario as it should.

The following paragraphs will define the used setups, tools, hooks and probes which will be referred to by the test descriptions.

TS_01 Asset Tracing test setup

The necessary components for this test setup are:

- A general purpose computer running the INTER-MW software. This software must include the OM2M bridge and both alignments related to that platform, namely “GOIoTP to OM2M” and “OM2M to GOIoTP”. It must be connected to the Internet to communicate with the OM2M server of VUB (also called Infrastructure Node or IN), reachable through the URL “https://onem2m.duckdns.org”. The OM2M bridge must be configured to use the right credentials to access the server with https basic authentication.
- At least one Sensolus tracer with a running Sensolus subscription. It must be used in an area within Sigfox coverage.

The tracers are registered in the OM2M Infrastructure Node, which already has a container per device to store incoming location data. The custom Interworking Proxy Entity retrieves that data from the Sensolus cloud every 15 min and converts it to oBIX format for storage on the Infrastructure Node. An Access Control Policy (ACP) resource including a rule for the bridge resource with permission to retrieve and create is already present in the Infrastructure Node (IN), and the containers of the trackers include it in their Access Control Policy Identifiers (ACPI).

The MW2MW layer is not yet connected to the OM2M platform. A client has been created with sufficient access rights to retrieve data via queries and create subscriptions.

TT_01 Test tool x

Test tools are not included in this setup.

TH_01 Test hook x

Test hooks are not included in this setup.

TP_01 Test probe 1

The bridge copies in its log the “protocol-less” request and response primitives exchanged with the OM2M platform, as well as the first observation message send from the bridge to the INTER-MW layer (or rather the Junit test playing its role) after a notification from the platform is received.

3.3.2.4 Test description

Scenario 1

This scenario focuses on the retrieval of the location of a device. This can be achieved either using a “query” message from the MW2MW layer or by subscribing to the device. In both case, the message with the location data coming from the OM2M will be syntactically and semantically translated to comply with the Generic Ontology for Internet of Things Platforms (GOIoTP). At that point, it could be sent further to any other supported platform, but this test will not focus on that part.

Due to the limitation present in the 868MHz band, that Sigfox uses, the tracers publish their position each 15 minutes, and additional data cannot be requested from the devices. The query messages retrieve the last available content instance in the virtual device corresponding to the tracker of interest. The location data acquisition by the tracers is not guaranteed to work in a closed environment, so it is advised to place them outside for the test. The latitude and longitudes of the positions of the tracers should be known to validate the correctness of the observations.

Registration

The INTER-MW layer needs to register to the VUB oneM2M server (called the Infrastructure Node), before any other message can be sent through the bridge.

The error messages encountered when sending the Platform_register message are the following:

“Failed to create(/register as) Application Entity for the bridge, status code:”

Followed by an oneM2M status code, which are listed in the oneM2M standard (TS004, section 6.6).

Check that the oneM2M Infrastructure node can be reached by using the web interface.

At this point, the INTER-MW log should be checked for the errors:

“Error: No URI list in response” or “Error: No ACP found for the bridge”

That error will not stop the program right away and registration will succeed, but further interaction with oneM2M resources will fail. The errors both mean the Access Control Policy (ACP) required for the bridge to communicate with oneM2M is not present or cannot be found in the oneM2M Infrastructure node. That ACP must include in its Privilege attribute a rule for the Identifiers of the bridge (f.i. /in-cse/in-name/ae_iiot_bridge and in-name/ae_iiot_bridge) the full access, meaning the possibility to create, retrieve, update and delete oneM2M (acop value of 63 or 31) resources.

```
<pv>
        <acr>
                <acor>/in-cse/in-name/ae_iiot_bridge in-name/ae_iiot_bridge</acor>
                <acop>63</acop>
        </acr>
</pv>
```

The presence of that ACP resource can be checked directly in the web interface of the oneM2M Infrastructure Node at https:// onem2m.duckdns.org/webpage.

Subscritpion

A Subscribe message only contains the URI of the Container one wishes to subscribe to, for instance: http://onem2m.duckdns.org/in-cse/in-name/DEVICE_0/LOCATION_DATA (hierarchical identifier) of http://onem2m.duckdns.org/in-cse/cnt-1234567 (non-hierarchical identifier). Note that INTER-MW only uses HTTP URI although the connection to the IN actually uses HTTPS (this is normal and the bridge will use whatever is defined by the "protocol" property).

Error message encountered at the creation of a subscription:

"Failed to create 1 of the 1 subscriptions"

The INTER-MW log will contain this line for each of the failed subscriptions (in this case 1):

"Failed to create subscription to device: %EntityId% with status code: %StatusCode%"

If the status code is equal to 4005 (Operation not allowed) check if the oneM2M resource you try to subscribe to includes the bridge ACP. The non-hierarchical ID of /in-cse/in-name/acp_iiot_bridge (<ri> attribute) should appear in the <acpid> attribute of the resource, as shown on Figure 1.

If the status code is 5204, check in the INTER-MW log if the bridge was able to start a jetty server (error: Unable to start jetty server, error: …). This can be caused by the Operating System of the computer where INTER-MW is running. If the server is running, check the properties of the bridge, if the callBackURL property is followed by your computer public IP address.

Attribute	Value
rn	acp_iiot_bridge
ty	1
ri	/in-cse/acp-316049630
pi	/in-cse
ct	20180903T140433
lt	20180903T140433
pv	AccessControlOriginator: /in-cse, /in-cse/in-name/ae_iiot_bridge, in-name/ae_iiot_bridge; AccessControlOperation: 63

Attribute	Value
rn	temperature
ty	3
ri	/in-cse/cnt-931478309
pi	/in-cse
ct	20180903T143914
lt	20180903T143914
lbl	• iiot_bridge • temperature
acpi	AccessControlPolicyIDs: /in-cse/acp-805683336, /in-cse/acp-316049630

Figure 45: Container resource (right) including an ACP resource (left)

Retrieval of location data

The location of a sensor appears in a RDF graph in the form:

%DeviceId% <iiot:hasLocation> A
A <rdf:type> sosa:Result
A <rdf:type> iiot:Location
A <geosparql:asWKT> point[%latitude%, %longitude%]

That structure is the same if one pulls the location with a Query message or is notified of a new value for a subscribed-to device.

One type of error can occur when retrieving observation with a “Query” message or receiving an observation after having subscribed to a device (i.e. oneM2M Container resource):

“Failed to translate retrieved content instance to rdf”

The Syntactic Translator of the bridge only for resources serialized in xml (this is hard-coded, should not be an issue) and, if that resource is a Content Instance, its content needs to be plain text or oBIX (check the Content Info or <cnf> attribute of the Content Instance).

Another possible error message specific to responses to a Query message is:

“Failed to retrieve content instance %ContentInstanceId% status code: %StatusCode%

If the status is 4004 (not found), check if the container exists in the oneM2M Infrastructure Node. If that container has no Content Instance, this behavior is normal as the oneM2M request targets the last Content Instance of the Container, in this case nothing.

Creation of Devices

The Message with type “Platform_Create_Device” need to go through IPSM for GOIoTP to oneM2M semantic translation before being passed to the bridge. This step can fail if IPSM is not running, if the kafka message broker fails to deliver the message to IPSM or if the IPSM channel for downstream messages to oneM2M has not been properly initialized. For the two first errors, check the IPSM and kafka logs, respectively. The channel should be created automatically when INTER-MW loads all the bridge classes, which will be mentioned in the INTER-MW log:

“Upstream IPSM channel for platform http://onem2m.duckdns.org has been created successfully.”

“Downstream IPSM channel for platform http://onem2m.duckdns.org has been created successfully.”

Please note that the new Container will not necessary appear in oneM2M with the URI mentioned in the INTER-MW message. This is due to the fact oneM2M imposes the (non-hierarchical) identifier of new resources. This should not have a negative effect on using the device Id later on because the bridge will store it in a map.

Sending a large number of Platform_Create_Device messages (see next section) should not cause issues in the INTER-MW to IPSM interface as Kafka is built to scale efficiently, but the semantic and syntactic translation of the messages can introduce delays.

Use case 1

T1.1.1 Location of tracers

ID	T1.1.1
Test	Retrieval of location data through the INTER-MW API
Type	System testing
Setup	TS_01
Start	The OM2M platform is paired to the tracers
Req.	4 ,14,15, 16, 26, 42, 45, 51, 52, 53, 127, 154, 167, 178, 179, 180, 183, 194, 201, 220, 224, 234, 237, 238, 242, 254, 255, 256, 270, 272, 283
Input	Send a “platform register” message through the INTER-MW REST API. Send a “query” message” through the INTER-MW REST API with destination the full hierarchical Id of the container of a tracker. Send a “subscribe” message” through the INTER-MW REST API with destination the full hierarchical Id of the container of a tracker.
Output	Check the creation of an Application Entity (AE) resource representing the bridge inside the Infrastructure Node. Check the OM2M primitive response sent to the bridge after the query: a Content Instance with oBIX content, and the corresponding data received in the INTER-MW REST API: a jsonLD text containing a Well Known Text (WKT) representation of the location data. Check the creation of a OM2M subscription as a child resource of the Container of interest
Logs	INTER-MW.log
Outcome	Pass / Fail

T1.1.2 Fast device creation

ID	T1.1.2
Test	Creation of large number of oneM2M container in a short period of time
Type	System testing/ stress testing
Setup	TS_01
Start	The OM2M platform is register within INTER-MW
Req.	4, 53, 178, 179, 180, 220, 237, 256
Input	Use the INTER-MW layer to create 10 new devices per second for 20 seconds, with “Platform_Create_Device” messages in an already registered OM2M Infrastructure Node.
Output	200 new Container should appear as child resources of the bridge’s Application Entity within the OM2M Infrastructure Node. The translation of 200 messages should appear in the IPSM log, since the payload of Platform_Create_Device is intended for the platform (not for the bridge itself).
Logs	INTER-MW.log, ipsm.log
Outcome	Pass / Fail

3.3.2.5 Test outcome overview

The following table will provide an overview of the test result of all the performed tests in this SAT.

Test	Description	Outcome
T1.1.1	Location of tracers	Pass / Fail
T1.1.2	Fast creation of devices	Pass / Fail
SAT Outcome		**Pass / Fail**

Table 6: Test outcome overview

3.3.2.6 Integration ethics and security

Introduction

This section addresses recommendations and proposed solutions to the ethical and security points of the integration.

Inter-OM2M

Our research team is not collecting sensitive data related to people such that we are not under the law of the protection of the privacy of individuals.

3.3.3 Third Party: INTER-HARE

The INTER-HARE project is intended to design a new LPWAN technology flexible enough to transparently encompass both LPWAN devices and multiple so-called *low-power local area networks* (LPLANs) while ensuring overall system's reliability. A cluster-tree network (Varga, 2015) is created, where the LPWAN acts not only as data collector, but also as backhaul network for several LPLANs, as shown in Figure 46.

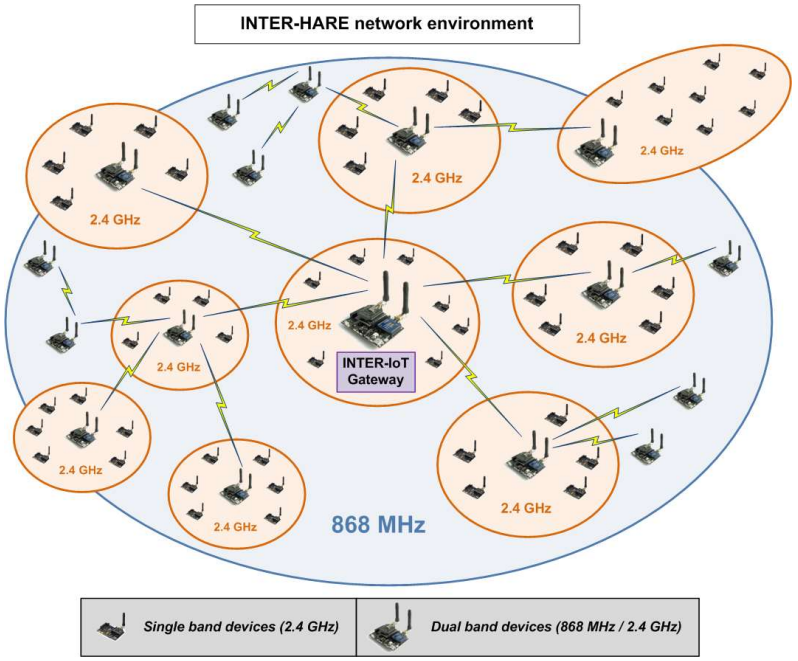


Figure 46: INTER-HARE transport network

Communication within the LPWAN is based on the HARE protocol stack (Adame, Barrachina, Bellalta, & Bel, 2018), ensuring transmission reliability, low energy consumption by adopting uplink multi-hop communication, self-organization, and resilience. Under these premises, LPWAN boundaries are extended beyond typical 868 MHz coverage range and easily integrate devices coming from adjacent/overlapping 2.4 GHz LPLANs. Use of separated frequency bands in overlapping networks results in an overall reduction of interferences. Lastly, thanks to the hierarchic system proposed, scalability is enforced by a management based on sub-networking techniques.

General considerations

The INTER-HARE platform is conceived as an innovative evolution of HARE protocol stack and can be considered as a dynamic multiprotocol. As it can be seen in Figure 47: Example of INTER-HARE transport network, in INTER-HARE, the INTER-IoT gateway (GW) and the cluster-heads (CH) of each LPLAN share the same protocol stack operating at 868 MHz and, at the same time, these CHs and the corresponding data acquisition devices (DADs) also share the same protocol stack, in this case operating at 2.4 GHz.

The INTER-HARE transport network conceives end devices as elements controlled by the GW by means of beacons, so that these are first received by CHs and then immediately retransmitted at a different frequency band in order to be listened by DADs. This centralized approach allows DADs to remain asleep the majority of the time, so that their single concern is to be awake enough in advance to listen to the next beacon. Network synchronization is thus achieved and allows the GW to ask for specific data and/or distribute configuration changes easily.

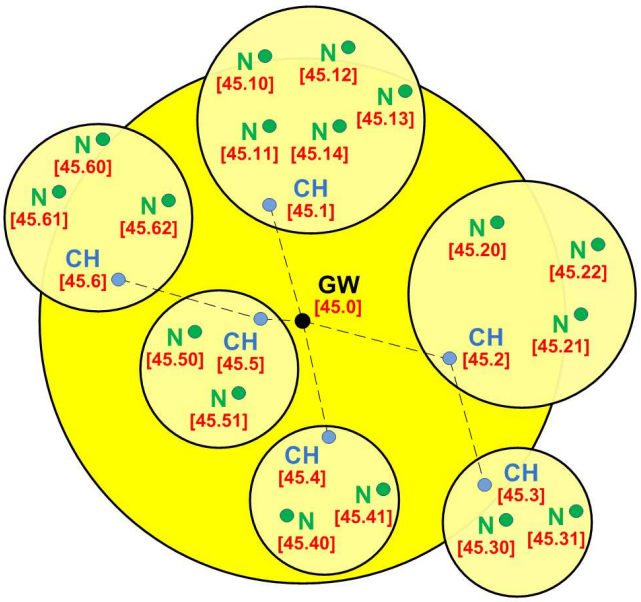


Figure 47: Example of INTER-HARE transport network

The GW is considered to be appropriately placed close to a power source or an energy harvesting solution. Therefore, it may always stay in an active state and is provided with the ability to directly communicate (i.e., via single-hop communications) with any CH of the network through unicast and/or broadcast messages as well as to redirect gathered data to other networks or the Internet.

Conversely, CHs can take advantage of their neighbours to create multi-hop paths over which data is transmitted to the GW by means of lower transmission power levels. Depending on their position within these paths, CHs of the LPWAN are ideally organized into rings, as shown in Figure 48: Ring structure of the LPWAN. The number of hops to reach the GW determines the ring number (i.e., CHs from ring 2 need two hops to reach the GW).

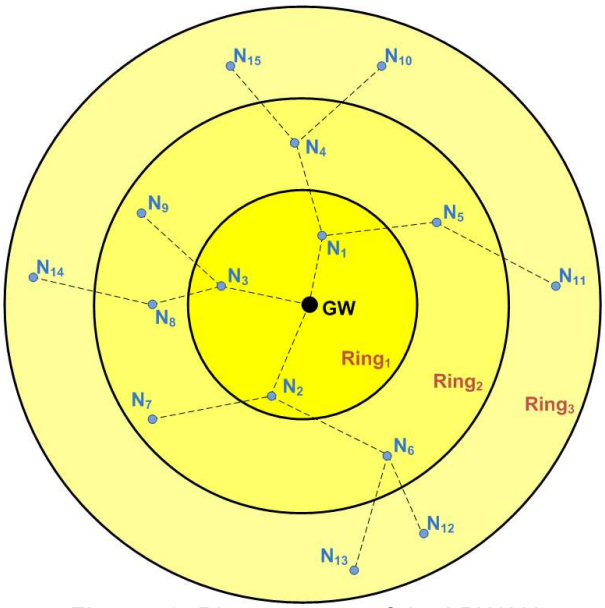


Figure 48: Ring structure of the LPWAN

Each uplink data transmission phase (consisting of one or more transmission windows) begins with a beacon signal from the GW. Transmission windows are in turn virtually split

into as many TDMA slots as network rings, so that CHs are only active during their own slot (for transmitting data) and the previous one belonging to their children[3] (for receiving data).

The first slot is allocated to the highest ring and the rest are scheduled consecutively. Data received by CHs is aggregated to that generated by themselves (i.e., data previously received from DADs of their LPLAN), and finally sent to the corresponding parent at the minimum power level which ensures reliable communications. This process is repeated as many times as rings the network has.

The correct reception of data transmissions at the GW is acknowledged with a broadcast message, so that CHs are not only aware of their own end-to-end reliability, but also of those CHs in the same path to the GW. These acknowledgment beacons, together with the information obtained from their adjacent nodes, allow CHs to decide whether they should remain awake to perform retransmissions of lost network packets.

Network association (also started by a beacon) remains stable until a change in the topology is detected or the mechanism is reset by the GW. Nevertheless, the agreed transmission power between adjacent nodes in the association phase is constantly monitored and adjusted in a decentralized way in order to reduce the energy consumption.

Architecture

The architecture of the INTER-HARE platform can be split into two networks with different purposes: the transport network and the integration network (as it can be seen in Figure 49: Proposed architecture for the SAT tests).

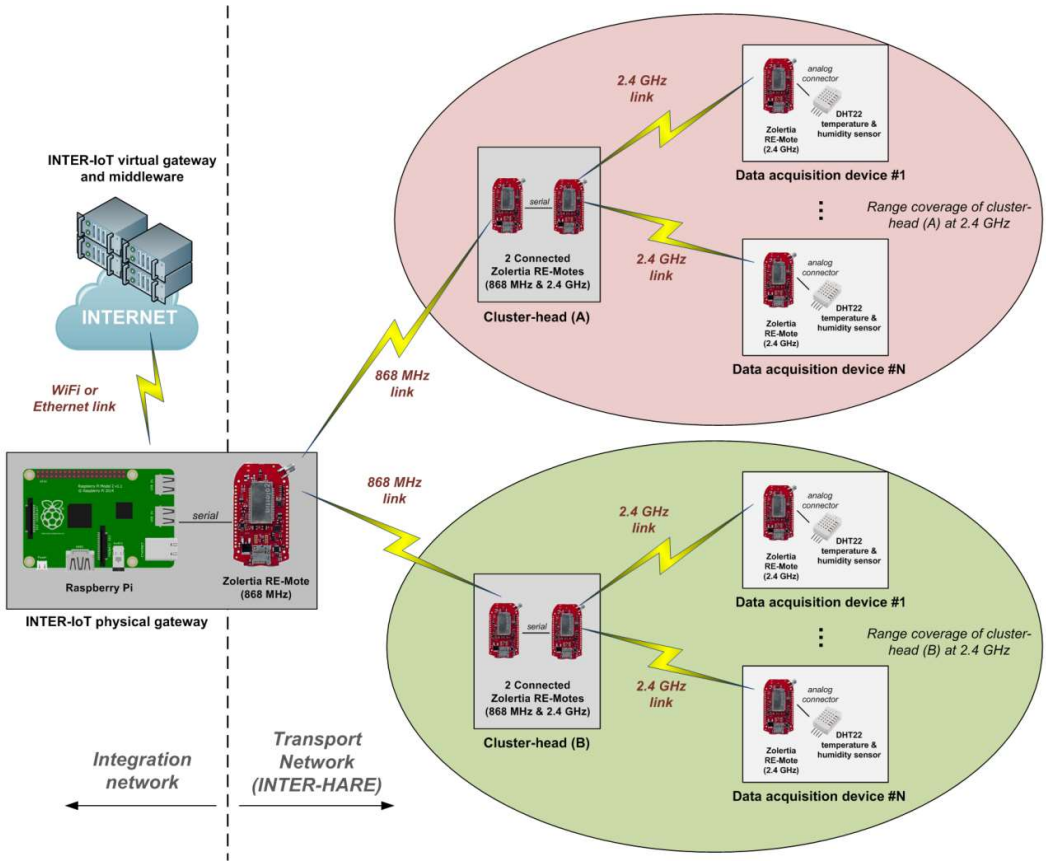


Figure 49: Proposed architecture for the SAT tests

[3] *Children* refers to all CHs of an adjacent higher ring from which a CH receives packets. Similarly, parent refers to that CH from an adjacent lower ring to which a CH transmits its own packets (after aggregating the ones from its children) in its way to the GW.

Transport network

It involves all internal infrastructures responsible for gathering and transporting information from the end-devices to the physical gateway. There are up to 3 different elements in the transport network:

- **Gateway (GW)**
  This device is responsible for controlling the two-tier cluster-tree network, gathering all the collected information, and transmitting this information via serial to the Raspberry Pi 3B+[4], which in turn acts as the main element of *the integration network*. The device selected to perform this function is a Zolertia RE-Mote (revision B) working at 868 MHz frequency band. For more information, we address the reader to check its datasheet (Zolertia, 2016).
- **Cluster-head (CH)**
  Cluster-heads are entitled by the INTER-IoT gateway to manage their corresponding LPLAN in a hierarchic way. They are the only elements with a multiband radio module (working at 868 MHz and 2.4 GHz), so that they gathered the information transmitted by data acquisition devices of their own LPLAN at 2.4 GHz and retransmit it to the INTER-IoT gateway (or alternatively, to the closest relay device) at 868 MHz through the LPWAN. In this case, the device selected to perform the role of cluster-head is obtained from the serial connection of 2 Zolertia RE-Mote, one working at 868 MHz frequency band (acting as master) and the other one at 2.4 GHz frequency band (acting as slave).
- **Data acquisition device (DAD)**
  Data acquisition devices are those elements deployed directly on the area where one or more environmental variables must be monitored. In the current project, the selected data acquisition device is a Zolertia RE-Mote only working in its 2.4 GHz frequency band.

Integration network

It is responsible for ensuring the communication between the physical gateway and the rest of the INTER-IoT system (or more specifically, with the virtual gateway), as shown in Figure 50: Structure of the integration INTER-HARE platform in INTER-IoT project.

Consequently, it is redefined the INTER-IoT gateway, which is considered the brain of the INTER-HARE platform and the single point of contact between the physical network and the rest of the INTER-IoT system. Due to its dual conception, it is easy to split its internal architecture into:

- **Physical gateway**
  A combination of a wireless frontend (responsible for the communication with the rest of the transport network) and a controller (responsible for the communication with the virtual gateway). Both elements are connected through a serial link. While the wireless frontend is a Zolertia RE-Mote, as explained in Section 0 Transport network, the controller is an OSGi bundle executed by the Raspberry Pi 3B+.
- **Virtual gateway**
  A virtual entity which can be executed in a remote location, based in the Docker platform; i.e., a virtual container that provides an additional layer of abstraction and automation of operating-system-level virtualization on Windows and Linux.

[4] Raspberry Pi main website - https://www.raspberrypi.org/

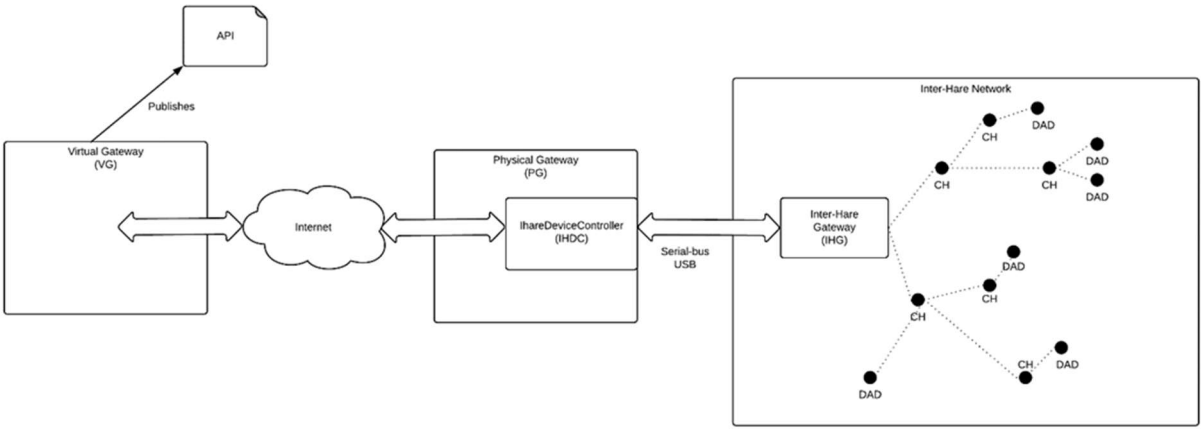


Figure 50: Structure of the integration INTER-HARE platform in INTER-IoT project

3.3.3.1 Integration of IoT framework

The architecture of the integration network is closely related to that of the employed gateway. In the specific case of the INTER-HARE platform, and according to the options presented in deliverable ***D3.1. Methods for Interoperability and Integration***, the gateway element has been split into two parts: *"the physical part for the embedded device and the part that can be executed in a virtual container"*.

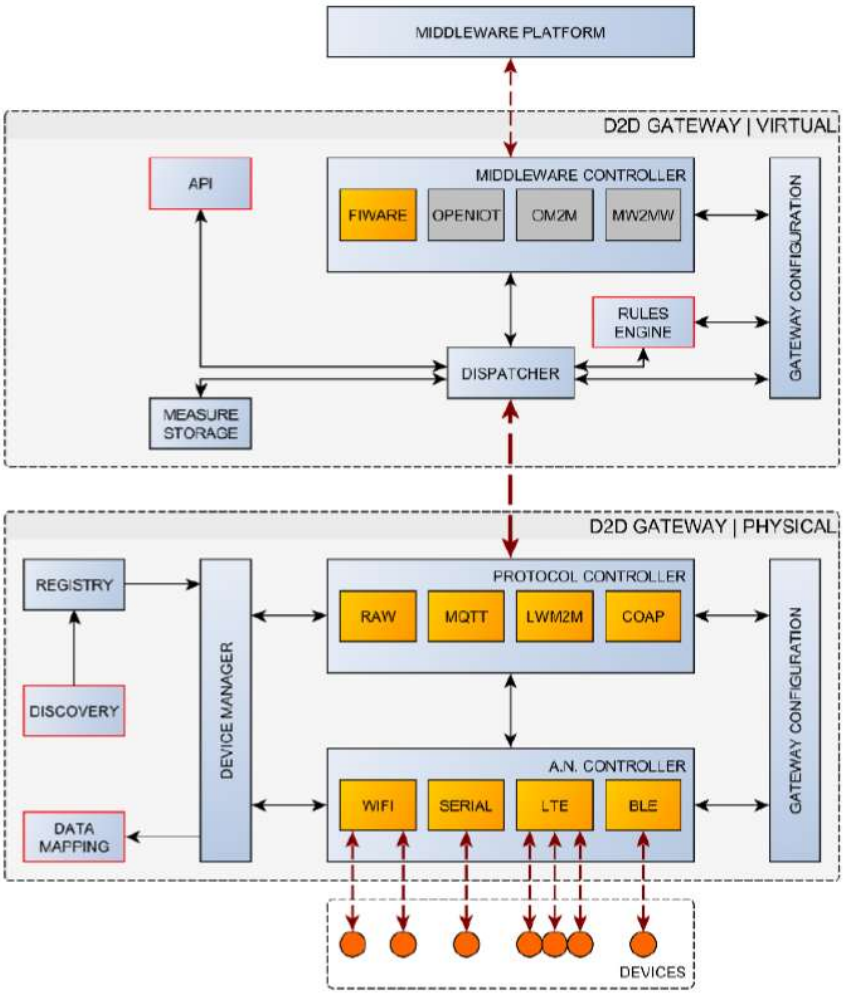


Figure 51: Generic gateway architecture of the INTER-IoT project

A clear definition of this architecture can be observed in Figure 51: Generic gateway architecture of the INTER-IoT project, where the two parts of the gateway (the *physical* and the *virtual*) are clearly defined. In the INTER-HARE platform, the elements performing each role are defined as follows:

Physical gateway: A combination of a wireless frontend (responsible for the communication with the rest of the transport network) and a controller (responsible for the communication with the virtual gateway). Both elements are connected through a serial link.

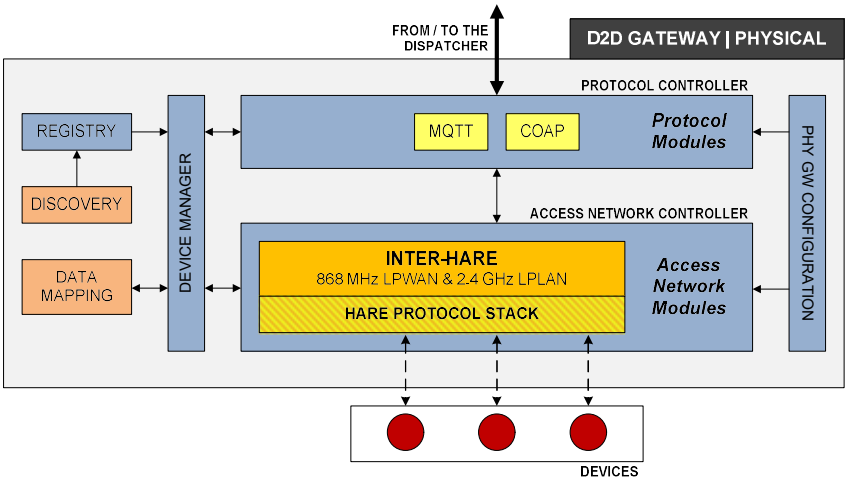


Figure 52: Detail of the elements composing the physical gateway

The physical gateway is made up of the elements shown in Figure 52: Detail of the elements composing the physical gateway**.** As for its implementation in the INTER-HARE platform, a Zolertia RE-Mote will act as a wireless frontend responsible for gathering all the wireless information transmitted to the gateway at 868 MHz. The rest of elements (the access network module, the protocol modules and others) are embedded in a Raspberry Pi 3B+ board[5].

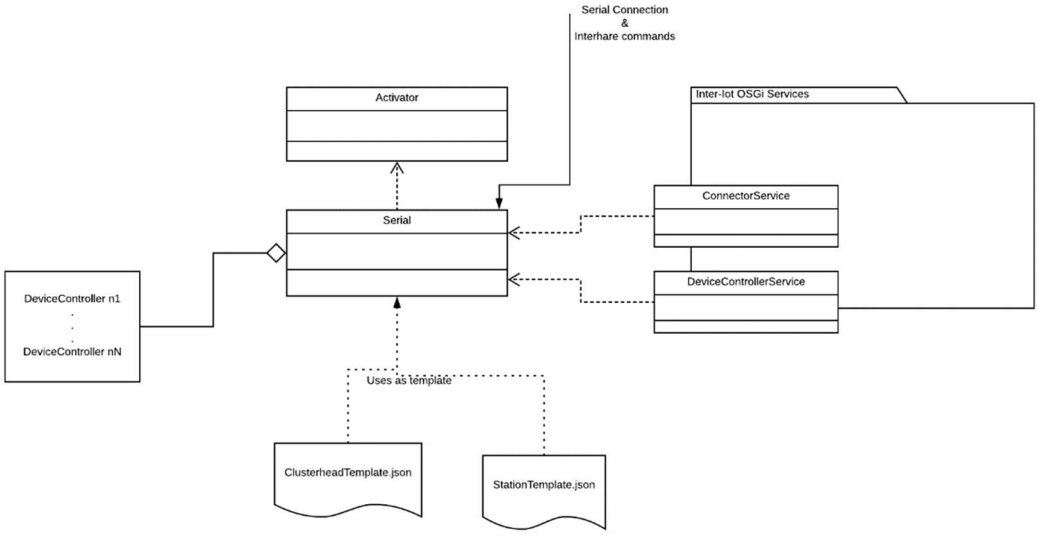


Figure 53: Internal structure of the INTER-HARE device controller

The device controller extension developed by UPF is an OSGi bundle integrated in the physical gateway infrastructure. It first instantiates an Activator, created by the class serial, in charge of receiving and processing the information from the serial-bus connection. This serial

[5] Preliminary development and tests in a computer are considered before using the Raspberry Pi 3B+ board.

class recognizes the orders sent from the INTER-HARE transport network and creates new device controllers depending on the type of device. Once registered, these devices are able to send their gathered information to the virtual gateway.

Virtual gateway: A virtual entity which can be executed in a remote location, based in the Docker[6] platform; i.e., a virtual container that provides an additional layer of abstraction and automation of operating-system-level virtualization on Windows and Linux.

> This element is originally intended to be run in a remote location and controlled by the INTER-IoT consortium together with virtual gateways from other technologies. Because the virtual gateway is implemented as a Docker container, it can be also deployed in a physical location for testing purposes.

According to the classification of the different ways to connect to IoT sensors and actuators included in subsection 4.2.1. from deliverable ***D3.2. Methods for Interoperability and Integration v.2***, the INTER-HARE platform uses the second approach:

"At the middle level the dedicated sensors and actuators can connect, these are the COTS IoT devices. Usually these sensors and actuators have some dedicated communication protocol between the wireless sensor and some piece of electronics with a small processing core. They are capable of handling their own access controller and protocol controller and can be connected through a dedicated extension module implementing a Device Controller".

The physical and virtual gateway implementations share a common base and runtime code. Both are based in an OSGi framework wrapper (the OSGi framework has to be R4 compliant) with a customized bootstrap and initiation routines. This framework first loads the third party libraries, then the core components and afterwards the extension modules. Then a routine to initiation all the modules starts, and the Physical and Virtual Core take the main thread to control the gateway. In Figure 54: Physical gateway components and Figure 55: Virtual gateway components a schema and a summary of the OSGi Framework, wrapper and components is shown.

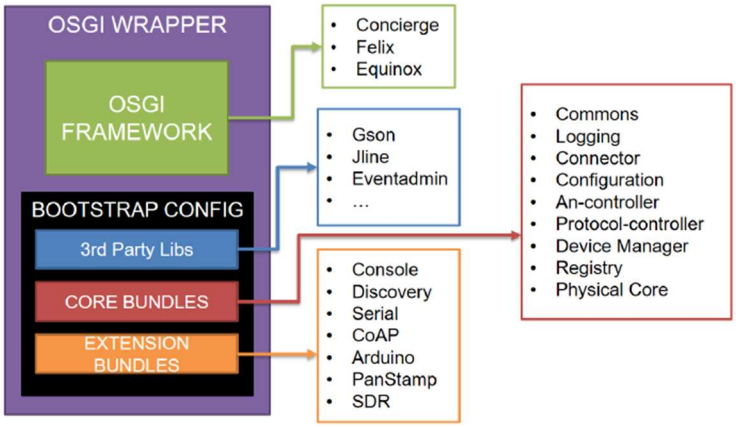


Figure 54: Physical gateway components

[6] Docker main website - https://www.docker.com/

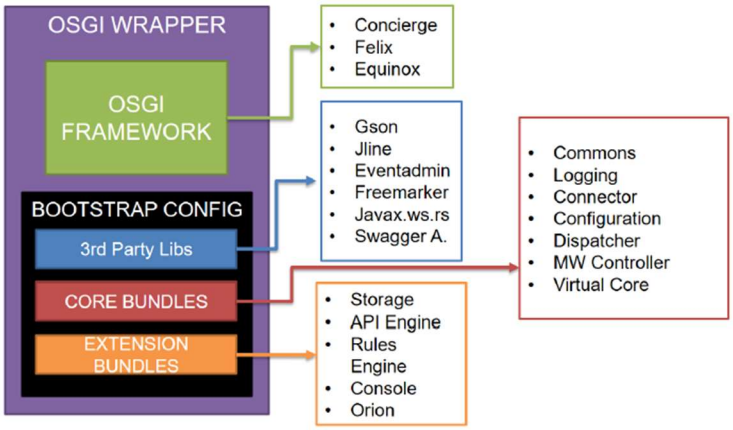


Figure 55: Virtual gateway components

In the INTER-IoT device-to-device interoperability gateway there are four different APIs:

- Gateway CLI: The gateway console extension provides a Command-Line Interface (CLI) to control the physical or virtual gateway instance.
- Gateway REST API module: REST API exposed by the virtual gateway API Engine extension module to interact with the virtual and physical gateway.
- Physical/Virtual Communication API: Messages exchanged between the physical and virtual through the connector module.
- Programmatic API: Libraries and interfaces needed to develop new extension modules for the gateway.

Communication protocols

The communication inside the INTER-HARE integration network is able to properly transmit data from the wireless frontend of the physical gateway to the virtual gateway, and vice versa. According to the aforementioned architecture, communication involves the following elements:

- Physical gateway
  - Wireless frontend
  - Access network module
- Virtual gateway

Consequently, two different communication protocols must be defined; one within the physical gateway (between the wireless frontend and the access network module), and another one between the access network module of the physical gateway and the virtual gateway.

Both communication protocols have been first tested in an on-premises integration network, where all the elements except from the wireless frontend are virtualized and run in a laptop, as shown in Figure 56: Diagram of the on-premises integration network (used for testing purposes). Note the presence of a Java application, which receives data from the wireless frontend and (in combination with a simplified Docker version of the INTER-IoT physical gateway) delivers it to the virtual gateway.

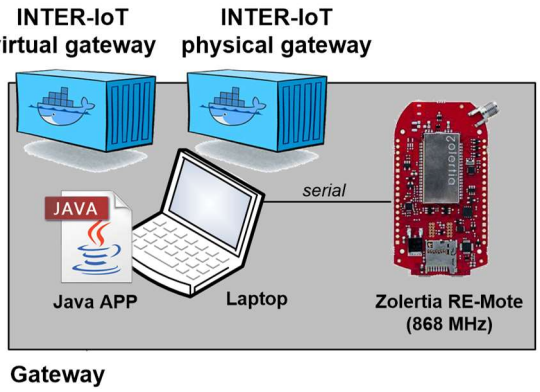


Figure 56: Diagram of the on-premises integration network (used for testing purposes)

Once validated the on-premises approach, the proposed final integration network runs the INTER-IoT virtual gateway in a server of a remote location, connected through Internet to the INTER-HARE physical gateway (see Figure 57: Diagram of the final integration network). In the latter case, while the wireless frontend is still operated by the Zolertia RE-Mote, the laptop is replaced by a Raspberry Pi 3B+ containing all the necessary software to act as a real access network module.

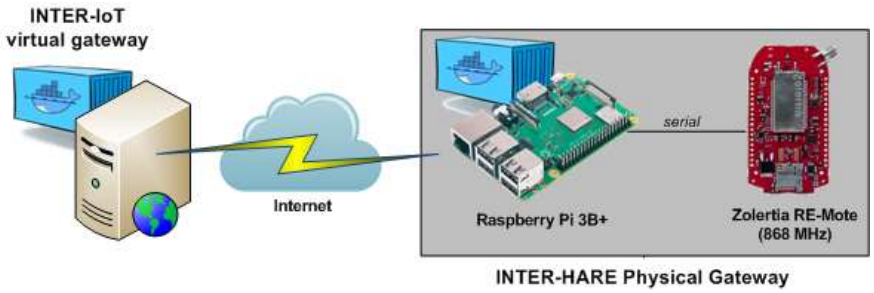


Figure 57: Diagram of the final integration network

Communication inside the physical gateway

Communication between the Raspberry Pi 3B+ (or the laptop in the first stage of the integration) and the Zolertia RE-Mote is based on a serial connection over USB. Data is thus easily transmitted and information flow can be controlled by means of simple commands.

Both devices are always running a routine listening to the USB port, so that whenever a new transmission is detected, the device acts according to the frame codification.

Communication between the physical and the virtual gateway

Between the physical and virtual parts of the gateway there is a communication protocol consisting in a JSON message exchanged through a WebSocket connector, as defined in subsection 4.2.4.3 from deliverable ***D3.2. Methods for Interoperability and Integration v.2***. This system allows to have different implementations of physical gateways (i.e., more flexibility and implementation options) if the communication fits with the reference implementation for a given version.

The specification of the communication protocol between the physical and virtual part of the gateway can be found in the following Wiki page of the INTER-IoT project.[7]

[7] https://git.INTER-IoT.eu/INTER-IoT/gateway-extensions/wiki/Home (authentication required)

Transmission scheduling mode

The two communication protocols defined in subsection 0 follow different transmission scheduling modes. While the communication inside the physical gateway is asynchronous (i.e., information is retransmitted once it has been received), the communication between the physical and the virtual gateway can follow synchronous or asynchronous patterns depending on the information criticality.

Raspberry Pi 3B+ ⟷ Zolertia RE-Mote	Transmission scheduling mode
Any transmission	**Asynchronous**

Table 19: Transmission scheduling mode in the communication in physical gateway

Virtual gateway ⟶ Raspberry Pi 3B+	Transmission scheduling mode
System management	**Asynchronous**
Data requests	
Virtual gateway ⟵ Raspberry Pi 3B+	**Transmission scheduling mode**
Data transmission (continuous, query-driven, and event-driven data delivery models)	**Synchronous**
Statistics transmission (continuous data delivery model)	
System management	**Asynchronous**
System alarms	

Table 20: Transmission scheduling modes between physical and virtual gateway

IoT framework components

The purpose of this subsection is to depict the different IoT framework components (and their interfaces) to be integrated in the pilot.

Network	Component	Interface	Test code
Transport	Gateway (GW)	Wireless frontend (868 MHz)	T4.46.1, T4.60.1, T4.60,7, T4.61.1, T4.62.3
	Cluster-head (CH)	868 MHz frequency band	T4.46.1, T4.60.1, T4.60,7, T4.61.1, T4.62.3
		2.4 GHz frequency band	T4.46.1, T4.60.2, T4.60,7, T4.61.1, T4.62.3
	Data acquisition device (DAD)	2.4 GHz frequency band	T4.46.1, T4.60.2, T4.60,7, T4.61.1, T4.62.3
Integration	Physical gateway	Controller	T4.46.1, T4.60,7, T4.61.1, T4.62.3
	Virtual gateway	-	T4.46.1, T4.60,7, T4.61.1, T4.62.3
	API	-	T4.19.1, T4.61.1

Table 21: IoT components used in the SAT tests

3.3.3.2 Deliverables and version overview

The following table contains a deliverable list which need to be signed of before SAT testing commences.

ID	Description	Check
Documents		
1	Validation and Test reports of the IoT system components	
2	Validation and Test reports of the Pilot system components	
3	SAT document	
Hardware		
4	Zolertia RE-Mote	
5	Raspberry Pi 3B+	
6	DHT22 temperature and humidity sensor	
7	Grove luminance sensor	
Tools		
8	Java viewer tool 1.0	
9	Development and demonstration environments setup (in Microsoft Azure Cloud)	
10	INTER-FW portal	

Table 22: Deliverable checklist

The following table shows the software components and version of which the system release version consists of.

ID	Description	Version	Check
IoT Data Acquisition Device			
1	INTER-HARE System - Data Acquistion Device Firmware	V2.0	
IoT Cluster Head			
2	INTER-HARE System - Cluster Head Firmware	V2.0	
IoT Relay			
3	INTER-HARE System - Relay Firmware	V2.0	
IoT Physical Gateway			
4	AN Controller	V2.0	
5	Protocol Controller	V2.0	
6	INTER-HARE System - Gateway Firmware	V2.0	
IoT Virtual Gateway			
7	Fiware	V4.2.3	
8	Virtual gateway docker	V0.3.0	
UniversAAL container			
7	UniversAAL REST API	V3.2.1	

Table 23: Component version overview

3.3.3.3 Requirements, scenarios and use cases

The following table provides an overview of the relation between the requirements and the test(s) that validate their implementation.

ID	Description	Covered	Test code
Architecture			
2	Scalability. Design	**YES**	T4.60.1, T4.60.2, T4.60.7, T4.62.3
6	Efficiency of the processing of information	**YES**	T4.60.7, T4.62.3, T4.46.1

9	Multi-level data processing support	**YES**	T4.60.7, T4.62.3, T4.46.1
Communications			
7	Support of opportunistic communications to avoid data loss	**YES**	T4.46.1
14	Platform independent	**YES**	T4.60.7
15	Common IoT communication protocols must be supported.	**YES**	T4.62.3
17	Dynamic network support	**YES**	T4.60.7, T4.46.1, T4.61.1
18	Roaming across networks	**YES**	T4.46.1
39	Gateway capabilities	**YES**	T4.60.1
45	Connectivity not based on HW identifiers	**YES**	T4.60.7
80	Support multicast communication among devices	**YES**	T4.62.3, T4.19.1, T4.61.1
153	System cache in gateways and upper levels	**YES**	T4.62.3, T4.46.1
232	Fault tolerance	**YES**	T4.62.3, T4.46.1
233	Flow control and network information tracking	**YES**	T4.62.3, T4.46.1
Functionality			
11	Addressability and reachability	**YES**	T4.60.7, T4.62.3, T4.46.1, T4.19.1, T4.61.1
19	Mobility	**YES**	T4.61.1
20	Real time support	**YES**	T4.62.3, T4.46.1, T4.61.1
21	Real time output	**YES**	T4.62.3, T4.46.1, T4.61.1
22	Unique identifier	**YES**	T4.60.7, T4.46.1, T4.61.1
23	Device semantic definition	**YES**	T4.60.7, T4.61.1
25	Remote programming of devices	**YES**	T4.62.3, T4.46.1, T4.19.1, T4.61.1
26	Remote device control	**YES**	T4.62.3, T4.46.1
43	IoT Services discovery	**YES**	T4.19.1, T4.61.1
89	Priority of routing and processing of critical messages upon low-priority sensor data	**YES**	T4.62.3, T4.46.1
API			
243	Gateway access API	**YES**	T4.19.1, T4.61.1
Interoperability			
13	Extensibility	**YES**	T4.61.1
16	Inter-connection support	**YES**	T4.60.7, T4.61.1
55	Independence of network layer	**YES**	T4.19.1, T4.61.1
56	Secure synchronization	**YES**	T4.62.3, T4.46.1
93	Standard protocol for the device communications	**YES**	T4.62.3, T4.46.1
138	User device capability detection	**YES**	T4.60.7, T4.19.1, T4.61.1
226	API for network services	**YES**	T4.19.1, T4.61.1
Legality			
29	Communication legislation and law	**YES**	T4.60.1, T4.60.2
Operational			
57	Device monitoring and self-awareness of the system	**YES**	T4.62.3, T4.46.1
75	The interaction between IoT endpoints may follow the M2M communication concept	**YES**	T4.62.3, T4.46.1

204	Support smart network resources allocation in heterogeneous wireless sensor networks	**YES**	T4.62.3, T4.46.1
205	Provide services to detect and predict devices' events in heterogeneous wireless networks	**YES**	T4.62.3, T4.46.1
206	Support scalable devices using power saving communication protocols	**YES**	T4.62.3, T4.46.1
207	Shall support scalable network topologies	**YES**	T4.60.7, T4.62.3, T4.46.1
Performance			
72	Communication should be done using protocols that are efficient in terms of amount of exchanged information over message size	**YES**	T4.62.3, T4.46.1, T4.61.1
Security			
27	System security	**YES**	T4.62.3, T4.46.1
28	System privacy	**YES**	T4.62.3, T4.46.1
95	Robustness, resilience and availability	**YES**	T4.62.3, T4.46.1
98	Data provenance	**YES**	T4.61.1
Virtualization			
242	Object/Device virtualization	**YES**	T4.60.7, T4.62.3, T4.46.1, T4.61.1
244	Gateway virtualization	**YES**	T4.19.1, T4.61.1

Table 24: Requirements vs. test mapping

The following table provides an overview of the relation between the scenarios and the test(s) that validate their implementation.

ID	Scenario name	Covered	Test code
*	Warehouse storage monitoring	**YES**	All tests

Table 25: Scenario vs test mapping

3.3.3.4 Test environment

Introduction

To test the functionality of the integrated INTER-Hare in combination with the IoT framework representative test hooks in the system are needed. This chapter will describe this environment and the used hardware, software, tools and platforms.

Test environment

This testbed is considered as the last stage of implementation of the INTER-HARE platform and will be useful to evaluate the system's performance in real conditions as well as to provide developers with enough information to set the best configuration for the given scenario. Therefore, it is possible to list the mail goals of this testbed (or test setup for integration) as:

1. To validate in a real scenario the different hardware elements included in the INTER-HARE platform.
2. To validate the different communication protocols developed for the INTER-HARE platform and analyze their performance.
3. To validate (according to the INTER-IoT consortium) an end-to-end communication scheme with the rest of the INTER-IoT platform, thus ensuring interoperability between both environments.

UPF first proposed to test its INTER-HARE platform in the ***Monitoring reefer container*** use case. The scenario is focused on tracking and monitoring the temperature of the container through different operators along its route, and to obtain faster responses in front of any issue with the temperature of the container.

However, and thanks to the expertise of the INTER-IoT project partner ***Valenciaport Foundation[8] (VPF),*** two main issues were detected prior to the deployment of a scenario based on real containers at Port of Valencia:

1. Metallic structure of port reefer containers may hinder the proper propagation of wireless signal and even block any kind of communication between the devices previously employed by UPF.
2. Port reefer containers stored at Port of Valencia are sealed with a seal number in order to prevent any container tampering until it has reached its final destination. This preventive measure makes difficult the deployment of wireless devices within these containers.

As an alternative solution, ***VPF*** proposed to change the use case to ***Warehouse storage monitoring*** and conduct the performance evaluation of the INTER-Hare in an *ad hoc* testbed located in one or some selected warehouses of ***Friopuerto[9]***, Valencia.

Friopuerto aims to provide port-oriented coldstorage services for importers and exporters internationally. It currently operates coldstore facilities in Mexico, Morocco, Portugal, Spain and Uruguay, all of them inside port zones. Its broad range of customers benefit from a state-of-the-art technology and processes to run frozen and chilled chambers, where they offer all kind of transloading, cross-docking and value-added services.

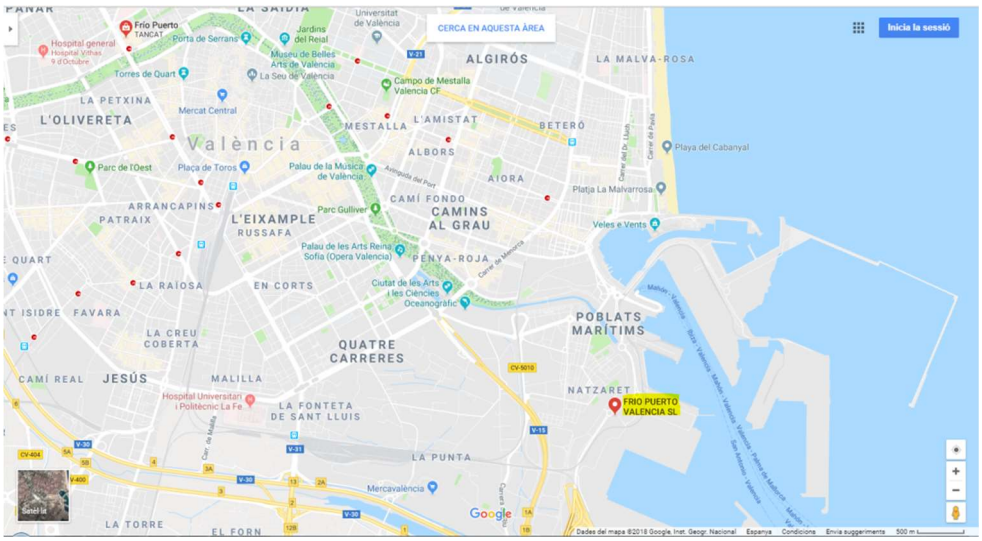


Figure 58: Friopuerto location at Valencia city

Friopuerto's vision is to provide top of the line services for perishable cargo requiring temperature control. They strive to assist any company in:

- **Warehousing**
  Friopuerto coldstores are clearly foreign-trade oriented and thus they are located within ports or very near them. They have been built with flexibility in mind, so they all incorporate different chambers for both chilled and frozen product. Their rack systems can be adapted for different pallet heights depending on customer needs.

[8] Valenciaport Foundation main website - http://www.fundacion.valenciaport.com
[9] Friopuerto main website - http://www.friopuerto.com/

- **Added-value services**
  Friopuerto facilities are designed to provide its customers with added-value services to help them streamline their supply chains. From picking and packing, to labelling and processing, its staff meets the demands of the most requiring clients.
- **Transportation**
  Friopuerto can assist companies with a broad range of transportation services, either directly or through top-class partners. They specialize in inland haulage of reefer containers as well as pick-up and distribution with refrigerated trucks for both LTL (Less than Truck Load) and FTL (Full Truck Load) shipments.

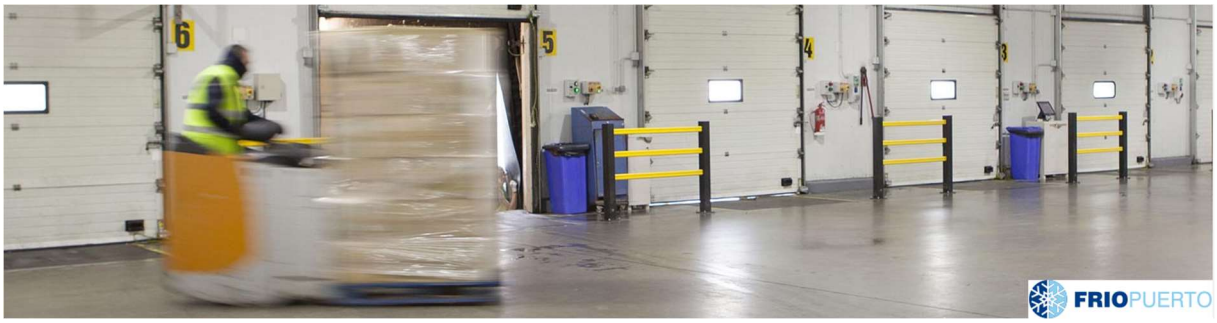


Figure 59: Friopuerto warehousing facilities at Valencia

SAT experiments will be conducted in the *Friopuerto*'s coldstore located at Port of Valencia, a 10.000 m^2 reefer warehouse with a storage capacity of up to 12.000 palet positions. After its 2015 extension, the facility now has 12 loading docks, 4 frozen chambers operating at -21ºC and 4 chambers for chilled products operating at temperatures from +2ºC to +14ºC.

Figure 60: Main features of Friopuerto's coldstore

Friopuerto is the only cold store operating inside the port of Valencia, which allows customers to enjoy cost savings and reducing the time needed for their operations. Being located inside the port, stored products can be stored under bonded and non-bonded status. Valencia is Spain's largest port and the hub for the Madrid area.

As for the specific coldstores planned to be used in the SAT tests of the INTER-HARE platform, their average temperature ranges from +8ºC to +10ºC. As it can be seen in Table 26, these values are included within the limits of the operating ambient temperature range of Zolertia RE-Mote.

Parameter	Min	Max	Unit
Operating ambient temperature range	-40	85	°C
Operating supply voltage	3	16 (PS+EXT) 5.1 (USB)	V

Table 26: Recommended operating conditions of Zolertia RE-Mote (Zolertia, 2016)

The features of the DHT22 temperature and humidity sensor that will be employed in SAT make it able to measure the temperature of *Friopuerto* coldstores, as this sensor has an operational range from -40ºC to +80ºC (see Table 27). Additionally, its error behaviour depending on the measured temperature is offered in Figure 61.

Parameter	Condition	min	typ	max	Unit
Resolution			0.1		℃
			16		bit
Accuracy			±0.5	±1	℃
Range		−40		80	℃
Repeat			±0.2		℃
Exchange		Completely interchangeable			
Response	1/e(63%)		<10		S
Drift			±0.3		℃/yr

Table 27: DHT22 temperature performance

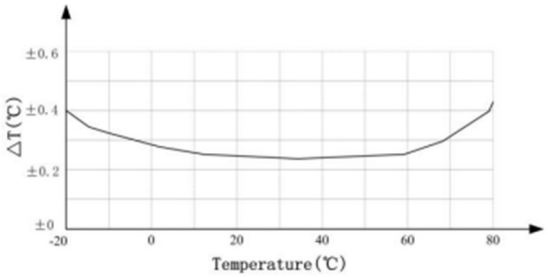


Figure 61: DHT22 Maximum temperature error depending on the measured value

Hardware components

A comprehensive list of all devices (and their consisting hardware components) considered for the INTER-HARE testbed is provided in Table 28.

Device	Subnetwork	Communication protocol	Operating frequency	Language / Software	Hardware
INTER-IoT gateway	Integration network	Connection from/to the INTER-IoT dispatcher	WiFi (2.4 GHz) / Ethernet (-)	*Java over Linux OS*	Raspberry Pi 3B+
	⇕ (1) Transport network	INTER-HARE	868 MHz	C over Contiki OS	Zolertia RE-Mote
Relay device	Transport network	INTER-HARE	868 MHz	C over Contiki OS	Zolertia RE-Mote
Cluster-head	Transport network	INTER-HARE	868 MHz + 2.4 GHz	C over Contiki OS	2 connected Zolertia RE-Motes
Data Acquisition device	Transport network	HARE	2.4 GHz	C over Contiki OS	Zolertia RE-Mote + DHT 22 temperature and humidity sensor

***(1)** Connected through the corresponding access network module*

· Electric line powered device
· Battery powered device

Table 28: List of INTER-HARE testbed components

INTER-IoT gateway

The INTER-IoT gateway (specifically, its *physical* part) is considered the brain of the INTER-HARE platform and the single point of contact between the physical network and the rest of the INTER-IoT system. Due to this dual conception, it is easy to split its internal architecture into the conforming elements responsible for interacting with the network running the INTER-HARE platform (*transport network*) and the ones exchanging information with the rest of the INTER-IoT system (*integration network*).

As for the part of the INTER-IoT gateway addressed to the transport network, it can be perfectly performed with a Zolertia RE-Mote working at 868 MHz.

Relay device

Only if it is necessary to extend the range coverage of the LPWAN network and retransmit the information from both the INTER-IoT gateway and the cluster-heads in the *transport network*, a relay device has been also considered in the INTER-HARE architecture. The Zolertia RE-Mote working at 868 MHz has also been the selected technology to perform this task.

Cluster-head

Cluster-heads are entitled by the INTER-IoT gateway to manage their corresponding LPLAN in a hierarchic way. They are the only elements with a multiband radio module (working at 868 MHz and 2.4 GHz), so that they gathered the information transmitted by data acquisition devices of their own LPLAN at 2.4 GHz and retransmit it to the INTER-IoT gateway (or alternatively, to the closest relay device) at 868 MHz through the LPWAN.

Originally, Zolertia Orion Router[10] was the selected device to perform the role of cluster-head due to its ability to communicate with devices both from 868 MHz and 2.4 GHz band. However, the operating frequency must be selected in the programming stage and cannot be changed during execution time, thus making impossible the normal CH operation.

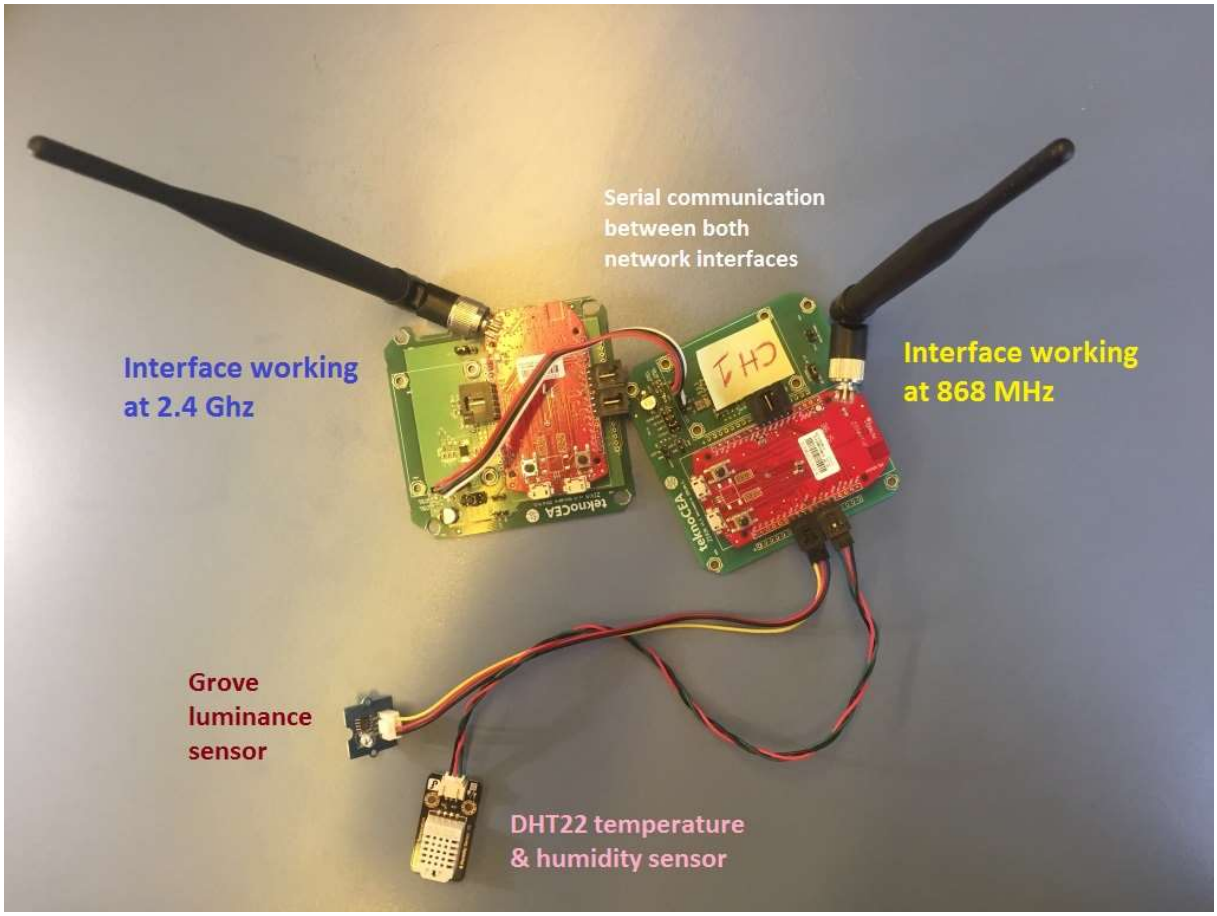


Figure 62: Cluster-head developed for the INTER-HARE platform

The proposed solution for developing a fully functional CH consists of connecting two Zolertia RE-Motes (one working at 868 MHz and the other one at 2.4 GHz) in a master-slave scheme. The one working at 868 MHz acts as *master*, controlling the activation and

[10] Zolertia Orion Router main website - https://github.com/Zolertia/Resources/wiki/Orion

deactivation cycles of the *slave*. In addition, the master device also includes a DHT22 temperature and humidity sensor and a Grove luminance sensor for monitoring purposes (see Figure 62).

Data acquisition device

Data acquisition devices are those elements deployed directly on the area where one or more environmental variables must be monitored. In the current project, the selected data acquisition device is a Zolertia RE-Mote only working in its 2.4 GHz frequency band.

As for the additional sensor embedded in these devices, the selected one has been the DHT22 temperature and humidity sensor. Transmission of environmental data acquired by the DHT22 sensor is transmitted to the Zolertia RE-Mote by means of an analogic connector. Additionally, a Grove luminance sensor can be also attached to the device in order to detect the intensity of the ambient light (in %) on a surface area.

Software components

Contiki 3.0 OS[11] is selected to validate the INTER-HARE platform, mainly due to its ability to easily execute multiple processes concurrently and its powerful COOJA network simulator. The INTER-HARE platform will be fully programmed in novel hardware-independent modules, one for each of the four possible network roles (INTER-IoT gateway, relay device, cluster-head device, and data acquisition device), adding all the required functionalities in order to ensure the proper operation of the *transport network*.

As for the *integration network*, the controller will be programmed in Java, embedded as a OSGi extension system together with all the other required functionalities needed to properly run its functionality; i.e., communication modules and INTER-IoT libraries are provided. This extension is compiled and compressed in a .jar file and stored inside the corresponding folder structure.

Deployment

Performance evaluation will be performed in an *ad hoc* testbed located in one or some selected warehouses of *Friopuerto.* The proposed testbed will consist of the elements depicted in ***Table 29***, which will run their own version of the INTER-HARE platform.

Device	Quantity
INTER-IoT gateways	1
Cluster-heads (868 MHz & 2.4 GHz)	2
Data acquisition devices (2.4 GHz)	4 or more per cluster-head (8 in total)
Relay devices (868 MHz)	Only if necessary

Table 29: Estimated pilot equipment

[11] Contiki OS main website - http://contiki-os.org/

Test tools, hooks and probes

The following paragraphs will define the used setups, tools, hooks and probes which will be referred to by the test descriptions.

Test setups

Test setup	Description
TS_01	Point-to-point topology (868 MHz)
TS_02	Point-to-point topology (2.4 GHz)
TS_08	Full INTER-IoT topology

Table 30: Test setups summary

TS_01 Point-to-point topology (868 MHz)

The point-to-point topology consists of only 1 GW and 1 CH, both working at 868 MHz frequency band. It is mainly intended to determine the channel conditions at this frequency band and the maximum coverage range achievable.

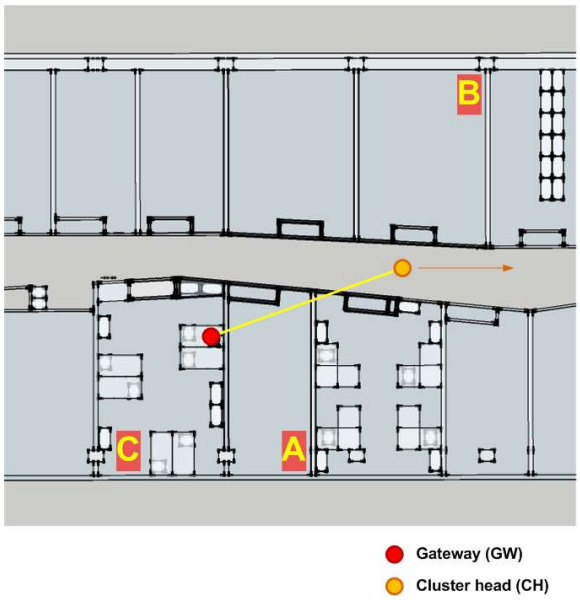


Figure 63: TS_01 network topology

TS_02 Point-to-point topology (2.4 GHz)

Similar to the previous topology, only two devices are used in this setup: 1 CH and 1 DAD. In this case, both devices work at 2.4 GHz frequency band and their main purpose is to determine the channel conditions.

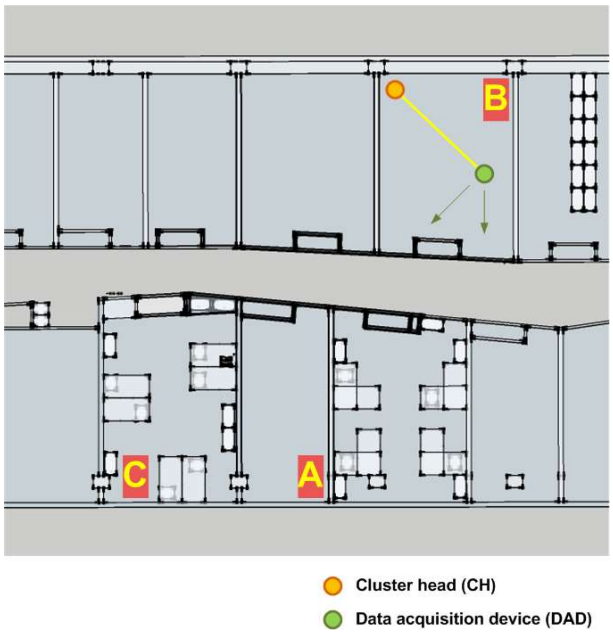


Figure 64: TS_02 network topology

TS_08 Full INTER-IoT topology

Lastly, all the elements of the INTER-HARE platform are interconnected in this test setup, from the data acquisition devices to the virtual gateway. As described in the system's architecture, two different networks collaborate to ensure end-to-end communications:

A. The transport network, consisting of the DADs, the CHs and the GW.
B. The integration network, consisting of the GW (considered in the deployment as the *physical* gateway), and the virtual gateway with its related middleware.

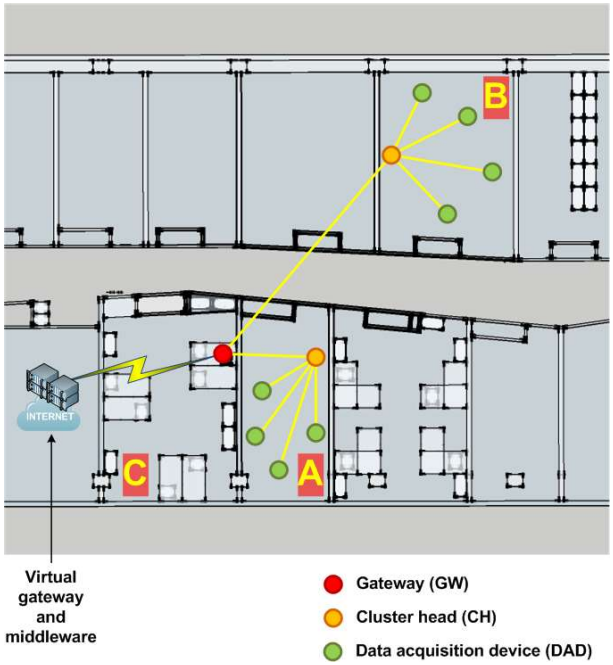


Figure 65: TS_08 network topology

Test tools

Test setup	Description
TT_01	Java viewer tool
TT_02	Zolertia RE-Mote leds
TT_06	Development and demonstration environments setup

Table 31: Test tools summary

TT_01 Java viewer tool

To facilitate the debug of the programmed protocols, code will contain messages and flags that will be transmitted via serial output by each system's device. All Zolertia platforms have a serial-to-USB converter on-board, meaning that devices can be connected to one USB port of a PC without any additional hardware but a cable. Serial output implemented in Contiki OS is supported by the standard C library API for printing.

However, to see the messages generated by devices is necessary to maintain a Linux terminal session active, which in turn makes difficult to store the received information. For this reason, it has been necessary to define the requirements of a new monitoring tool:

- Multi-platform (Windows, Linux, Mac OS)
- No necessity of Contiki OS previously installed
- Connection via USB
- Log recording
- Data filtering
- Debugging

The result is a Java-based monitoring tool able to gather, show and store all the log messages emitted by an STA directly connected to a PC via an USB cable. As can be seen in Figure 66, the tool informs about the COM port in which the STA is connected (for instance, /dev/com5) and show the ID of the STA once it has been properly recognized (in the example, ID is 6).

Different filters based on character strings can be applied on the gathered information in order to show only some relevant data or those debugging flags placed in the code run by STAs. Information can also be exported to a .txt file for post-processing purposes.

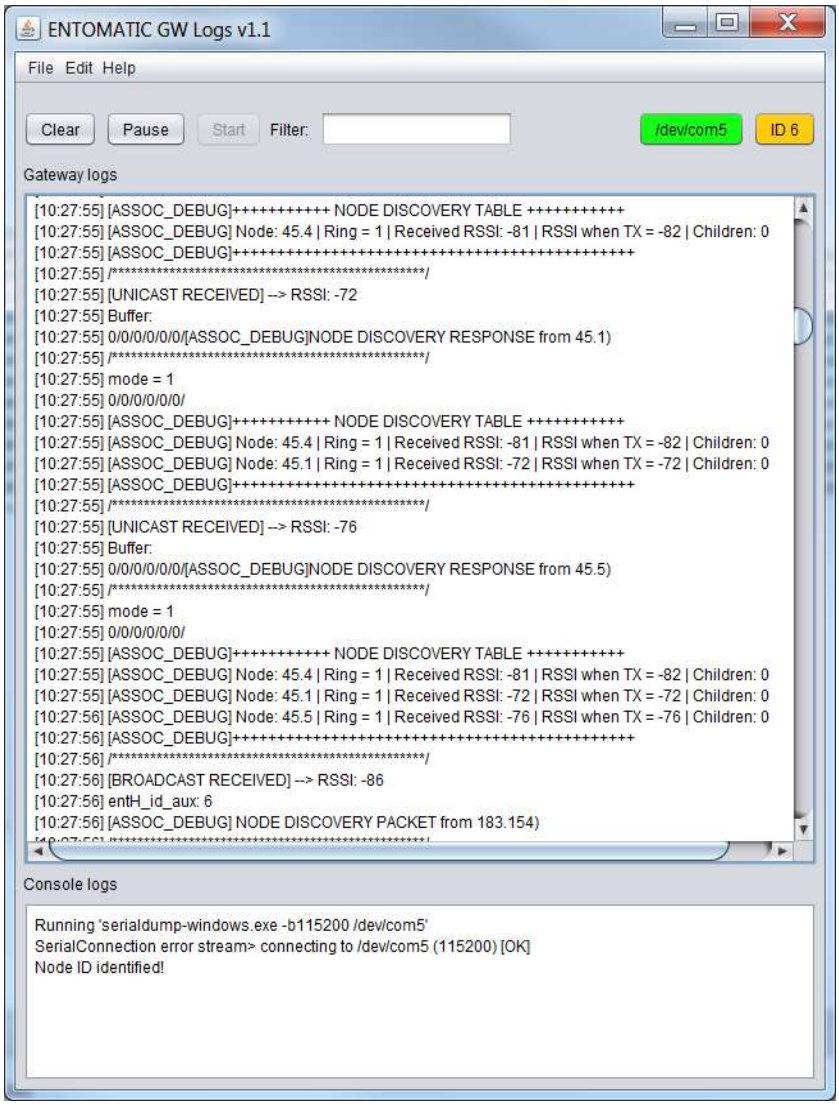


Figure 66: INTER-HARE monitoring tool based on Java

TT_02 Zolertia RE-Mote leds

LEDs are a simple but important tool to communicate with users or to debug programs. Each one of the hardware platforms over which the INTER-HARE platform will contain its own set of led codes.

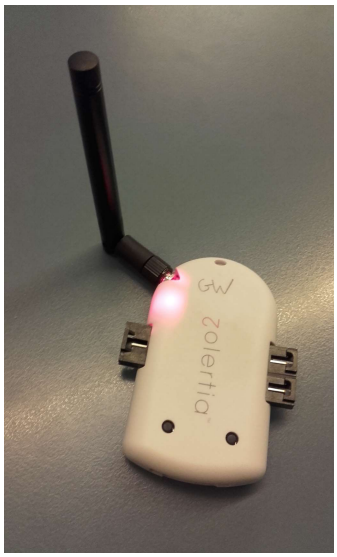


Figure 67: A Zolertia RE-Mote device with its led switched on in red

TT_06 Development and demonstration environments setup

As defined in ***Deliverable 3.2. Methods for Interoperability and Integration***, to enable an homogeneous controlled environment for the development of the different layers, a development cloud environment has been set up.

This environment is based in the Microsoft Azure Cloud and comprises a set of 7 commodity servers to decouple the different software modules development and, at the same time, making the latest features available to be tested. These servers are all accessed through a unique stepping-stone server. The access to this environment is securized through Microsoft Azure standard security mechanisms.

Test hooks

Test setup	Description
TH_01	Iterative switching on
TH_02	Continuous traffic injection
TH_03	Random traffic injection
TH_04	Error addition
TH_05	Occasional switching off
TH_06	User browsing

Table 32: Test hooks summary

TH_01 Iterative switching on

A device is switched on repeatedly, executing its initialization process. Although each considered device (gateway, cluster head, relay and data acquisition device) is programmed with different functions, they all will try firstly to send a discovery message to their immediate parent and start their corresponding mechanism in order to be registered in the network.

In some tests, this iterative switching on will be performed in the CHs or the DADs from increasingly distances to their closer GW or CH, respectively. It will be useful to determine the maximum distance from which that device can establish a connection with its parent.

TH_02 Continuous traffic injection

Continuous traffic injection consists in the periodic data acquisition and/or simulation by DADs and its corresponding transmission to their immediate parent. By following the beacon scheduling of the INTER-HARE platform, DADs will send a data packet every cycle.

TH_03 Random traffic injection

Random traffic injection is conceived as random processes executed in both the GW and the DADs, generating query-driven and event-driven traffic, respectively.

- Query-driven traffic is generated at the GW and consists of a data request which must be transmitted to a specific DAD of the network (for instance, a user asks for the temperature value of a determined station).
- Event-driven traffic is generated at DADs when a predetermined threshold has been surpassed (for instance, the temperature sensor has detected a value over 40 °C).

TH_04 Error addition

The whole system can be altered with the arbitrarily introduction of a certain error probability when sending both *application packets* and their corresponding ACKs (it is worth noting here that neither messages implied in the association process nor *statistics packets* are affected by arbitrary generated errors to not artificially disturb the network setup nor the collection of operation information).

Errors are generated through a uniformly distributed random variable according to mean error values from four different error configurations (see Table 33). Before sending a message, STAs compute this value and discard messages accordingly.

Error configuration	Data Error	ACK Error
$E_{0/0}$	0%	0%
$E_{10/5}$	10%	5%
$E_{20/10}$	20%	10%
$E_{30/15}$	30%	15%

Table 33: Definition of error configurations

TH_05 Occasional switching off

In some tests, an STA will be deliberately switched off in order to analyze the behaviour of the rest of the network to overcome this issue and rearrange the routing paths to the GW.

TH_06 User browsing

The user of the platform freely browses the different network configuration options and receives information regarding the network state and main functionalities.

Test probes

Test setup	Description
TP_01	Gateway log
TP_02	Cluster Head log
TP_03	Relay log
TP_04	Data Acquisition Device log
TP_05	INTER-FW portal

Table 34: Test probes summary

TP_01 Gateway log

The Gateway log includes information regarding both the transport and the integration network. It is stored in separated files according to the file system implemented in the device.

A summarized version of this log, only containing net information and network events could be optionally stored in a microSD connected to the device.

TP_02 Cluster Head log

The Cluster Head log is a file created by this device where all the gathered information as well as all hooks and flags are stored. It also includes information about the DADs directly associated to that device.

TP_03 Relay log

Similarly to other logs, the Relay log is a file which contains all the information received and transmitted by this device.

TP_04 Data Acquisition Device log

The Data Acquisition Device log is created each time this device is started and includes information with regard to the data acquired from sensors and to the connection with the corresponding CH.

TP_05 INTER-FW portal

As described in the example demo of ***Subsection 4.2.6. Demo*** from ***Deliverable 3.2. Methods for Interoperability and Integration,*** the INTER-FW portal is used to demonstrate how the data flows through all the test setup.

3.3.3.5 Test description

Warehouse storage monitoring

The objective of this scenario is to interoperate and use a warehousing company IoT platform that is currently able to monitor cargo from reefer containers during its storage in coldstores. This integration will allow a quick reaction in case of an alarm regarding the functioning of refrigerated goods and it will benefit warehousing companies to avoid the periodic human inspection required for this kind of cargo.

Interoperability in this scenario is required to connect the IoT platforms from warehousing and transport companies as well as final cargo owners.

The resulting service will be obtained by the integration of:

- IoT platform of warehousing company
- IoT platform of transport company
- IoT platform of final cargo owner

The coldstores from the warehousing company ***Friopuerto***, located at Port of Valencia, will be the selected facilities where the different tests from SAT will take place.

A comprehensive list of the considered use cases together with their associated tests is defined in the following lines.

Use case	Associated test
[19] User interacts with sensors or devices	T4.19.1 User interaction
[46] Device failure detection	T4.46.1 Resilience against failures
[60] Device registry	T4.60.1 Range coverage at 868 MHz
	T4.60.2 Range coverage at 2.4 GHz
	T4.60.7 Multiple sensor registration
[61] Platform Configuration on the Gateway	T4.61.1 Platform setup and simulation
[62] Device (sensor) triggers information	T4.62.3 Hybrid data delivery model test

Table 35: Summary of SAT tests and definition

Concept	Test code	Test name	Test setup	Tools	Hooks	Probes	Outcomes
A. Range coverage	T4.60.1	Range coverage at 868 MHz	TS_01	TT_01 TT_02	TH_01	TP_01 TP_02	Max. distance
	T4.60.2	Range coverage at 2.4 GHz	TS_02	TT_01 TT_02	TH_01	TP_02 TP_04	Max. distance
B. Association & Registration	T4.60.7	Multiple sensor registration	TS_08	TT_01 TT_02 TT_06	TH_01	TP_01 TP_05	Pass/Fail, Assoc. delay
C. Data transmission	T4.62.3	Hybrid data delivery model test	TS_08	TT_01 TT_06	TH_02 TH_03 TH_04	TP_01 TP_05	Pass/Fail, PDR (%), Delay, Throughput, PRM, Energy

							consumption
D. Resilience	T4.46.1	Resilience against failures	TS_08	TT_01 TT_06	TH_02 TH_03 TH_05	TP_01 TP_05	Pass/Fail
E. Integration network	T4.19.1	User interaction	TS_08	TT_02 TT_06	TH_02 TH_03 TH_06	TP_05	Pass/Fail
	T4.61.1	Platform setup and simulation	TS_08	TT_02 TT_06	TH_01 TH_02 TH_03 TH_05 TH_06	TP_05	Pass/Fail

Table 36: Diagram compiling the different test setups

	CONCEPT	A. Range Coverage		B. Association & Registration	C. Data transmission	D. Resilience	E. Integration network	
	TEST CODE	T4.60.1	T4.60.2	T4.60.7	T4.62.3	T4.46.1	T4.19.1	T4.61.1
ARCHITECTURE	2	X	X	X	X			
	6			X	X	X		
	9			X	X	X		
COMMUNICATIONS	7					X		
	14			X				
	15				X			
	17			X		X		X
	18							
	39	X						
	45			X				
	80				X		X	X
	153				X	X		
	232				X	X		
	233			X	X	X		
FUNCTIONALITY	11			X	X	X	X	X
	19							X
	20				X	X		X

	21				X	X		X
	22			X		X		X
	23			X				X
	25				X	X	X	X
	26				X	X		
	43						X	X
	89				X	X		
API	243			X	X		X	X
INTEROPERABILITY	13							X
	16			X				X
	55						X	X
	56				X	X		
	93				X	X		
	138			X			X	X
	226						X	X
LEGALITY	29	X	X					
OPERATIONAL	57				X	X		
	75				X	X		
	204				X	X		
	205				X	X		
	206				X	X		
	207			X	X	X		
PERFORMANCE	72				X	X		X
SECURITY	27				X	X		
	28				X	X		
	95				X	X		
	98							X
VIRTUALIZATION	242			X	X	X		X
	244			X		X	X	X

Table 37: List of requirements to be analyzed in each test

U19 – User interacts with sensors or devices

The whole platform is accessed remotely by the user, who can change some configuration settings. The user experience is analyzed.

T4.19.1 User interaction

ID	T4.19.1
Test	User experience analysis when configuring the INTER-HARE platform
Type	E. Integration network
Setup	Need test setup TS_08

Start	All DADs, CHs and the GW are already registered.
Req.	[80], [11], [25], [43], [243], [55], [138], [226], [244]
Input	Test hooks TH_02, TH_03, and TH_06
Output	Check system's ability to configure the parameters selected by the user.
Logs	E. Integration\T4.19.1_ux.txt
Outcome	Pass / Fail

U46 – Device failure detection

The system is able to detect problems in intermediate devices (CHs and DADs) and to act consequently, by reconstructing routing paths and ensuring data transmissions.

T4.46.1 Resilience against failures

ID	T4.46.1
Test	INTER-HARE resilience analysis against failures
Type	D. Resilience
Setup	Need test setup TS_08
Start	All DADs, CHs and the GW are already registered.
Req.	[6], [9], [7], [17], [153], [232], [233], [11], [20], [21], [22], [25], [26], [89], [56], [93], [57], [75], [204], [205], [206], [207], [72], [27], [28], [95], [242]
Input	Test hooks TH_02, TH_03, and TH_05
Output	Check if the system is able to rebuild routing routes after one (or more) DAD (or CH) switches off. Check if the system is able to maintain high reliability levels after one (or more) DAD (or CH) switches off.
Logs	D. Resilience\T4.46.1_resilience.txt
Outcome	Pass / Fail

U60 – Device registry

Devices are able to determine if they are within the range coverage of their immediate parent. If so, they execute the association process to be part of the network by receiving the corresponding network address and listening to the schedule beacons.

T4.60.1 Range coverage at 868 MHz

ID	T4.60.1
Test	Analysis of range coverage at 868 MHz
Type	A. Range coverage
Setup	Need test setup TS_01
Start	GW located in a fixed position. CH initially located close to the GW. CH is moved further from the GW.
Req.	[2], [39], [29]

Input	Test hook TH_01 in different CH positions
Output	CH inside or outside the range coverage of the GW.
Logs	A. Range\T4.60.1_868.txt
Outcome	Max. distance

T4.60.2 Range coverage at 2.4 GHz

ID	T4.60.2
Test	Analysis of range coverage at 2.4 GHz
Type	A. Range coverage
Setup	Need test setup TS_02
Start	CH located in a fixed position. DAD initially located close to the CH. DAD is moved further from the CH.
Req.	[2], [29]
Input	Test hook TH_01 in different DAD positions
Output	DAD inside or outside the range coverage of the CH.
Logs	A. Range\T4.60.2_24.txt
Outcome	Max. distance

T4.60.7 Multiple sensor registration

ID	T4.60.7
Test	Multiple DAD (sensor) registration into the INTER-IoT network
Type	B. Association & Registration
Setup	Need test setup TS_08
Start	1 GW, 2 CHs & multiple DADs located in a fixed position.
Req.	[2], [6], [9], [14], [17], [45], [233], [11], [22], [23], [16], [138], [207], [242]
Input	Test hook TH_01
Output	Check proper CHs & DADs registration into the INTER-IoT network.
Logs	B. Association\T4.60.7_multiple_reg.txt
Outcome	Pass / Fail

U61 – Platform Configuration on the Gateway

The whole platform is accessed remotely by the user. The user configures the system, activates the devices and controls the execution, even applying setup changes and/or sending specific requests to selected DADs.

T4.61.1 Platform setup and simulation

ID	T4.61.1
Test	INTER-HARE setup and full simulation
Type	E. Integration network
Setup	Need test setup TS_08
Start	All devices (1 GW, 2 CHs and multiple DADs) are located in a fixed position but are not associated to the network yet.
Req.	[17], [80], [11], [19], [20], [21], [22], [23], [25], [43], [243], [13], [16], [55], [138], [226], [72], [98], [242], [244]
Input	Test hooks TH_01, TH_02, TH_03, TH_05, and TH_06
Output	Check system's ability to configure the parameters selected by the user. Check correct transmission of packets from/to DADs according to the hybrid data delivery model.
Logs	E. Integration\T4.61.1_platform_conf.txt
Outcome	Pass / Fail

U62 – Device (sensor) triggers information

A device, typically a sensor, triggers an event sending determined information to the gateway in order to be stored in the platform.

T4.62.3 Hybrid data delivery model test

ID	T4.62.3
Test	Performance analysis of hybrid traffic
Type	C. Data transmission
Setup	Need test setup TS_08
Start	All DADs, CHs and the GW are already registered.
Req.	[2], [6], [9], [15], [80], [153], [232], [233], [11], [20], [21], [25], [26], [89], [56], [93], [57], [75], [204], [205], [206], [207], [72], [27], [28], [95], [242]
Input	Test hooks TH_02, TH_03, and TH_04
Output	Check performance of different network metrics.
Logs	C. Transmission\T4.62.3_hybrid.txt
Outcome	Pass / Fail
	PDR (%)
	Delay
	Throughput
	PRM
	Energy consumption

3.3.3.6 Test outcome overview

The following table will provide an overview of the test result of all the performed tests in this SAT.

Concept	Test	Description	Outcome	Value
A. Range coverage	T4.60.1	Range coverage at 868 MHz	Max. distance	
	T4.60.2	Range coverage at 2.4 GHz	Max. distance	
B. Association & Registration	T4.60.7	Multiple sensor registration	Pass / Fail	
			Assoc. delay	
C. Data Transmission	T4.62.3	Hybrid data delivery model test	Pass / Fail	
			PDR (%)	
			Delay	
			Throughput	
			PRM	
			Energy consumption	
D. Resilience	T4.46.1	Resilience against failures	Pass / Fail	
E. Integration network	T4.19.1	User interaction	Pass / Fail	
	T4.61.1	Platform setup and simulation	Pass / Fail	
	SAT Outcome		Pass / Fail	

Table 38: Test outcome overview

3.3.3.7 Integration ethics and security

Our research group, as a member of the ***Universitat Pompeu Fabra***, has established guidelines regarding ethics in all its projects. The university created the Internal Committee of Projects (CIREP-UPF) in December 2014. The committee wants to improve the evaluation of the ethical standards and personal data protection in research activities and academics practices.

Every research project performed by our group is previously analyzed by the members of our group following the guidelines of CIREP. The procedure is the following:

1. A self-assessment is performed by the principal investigators by filling an Ethics Checklist Form. If a YES answer is checked, the IP must send the form and proceed to step 2.
2. The IP has to prepare the following documents for the ethics committee: a summary information form that includes the title of the project and a summary of the research activity (including all the activities that involve any personal data or humans participating in them); a procedure form, that includes a detailed description of the methodology that the project will use in order to be evaluated by the CIREP; and, finally, an informed consent form that will be used during the research activities.
3. An external evaluation by a reviewer in the specific research field is performed. A peer review will be submitted by him/her to CIREP members.
4. Finally, the CIREP experts meet and discuss the researcher's documents and the peer review, to issue an Ethics Review Report to the researcher.

Our research project has been analysed following the Ethics checklist form, and, as it neither contemplates the participation of humans nor uses any human data, the ethics of our project has been evaluated and the basics requirements are fulfilled.

On the other hand, the use of confidential data along the pilot is not either contemplated. Hence, the security is not compromised. The user IDs in our network are codified, so that any external intruder with non-authorized access to the sensor network will not have any knowledge on the data due to the codification of the elements belonging to the network.

Management of Research Data

INTER-HARE makes its research data Findable, Accessible, Interoperable and Reusable (FAIR), following the EC data principles:

- Data protection legislation will be upheld at a national level as well as at a European level. Project partners will have access to anonymised datasets for the purposes of analyses based on well-protected identifier codes to denote identity.
- Data will not be shared, published or distributed without being pseudonymised first, and access to the raw data will not be provided without justification and then in such a way as to prevent any potential harm to participant. Data cannot be linked to any individual (numeric codes are used for this data; no one is identifiable by the data they provide).
- Primary data will be kept for the lifetime of the project (plus any statutory period), and then will be physically and securely destroyed.

INTER-HARE aims to ease the accessibility to our generated data and as such the technology providers of the project will also make available software libraries for reading the provided data. The selected datasets will be stored in a server of one of the project partners (UPV). Datasets will be accessible from the project website.

3.3.4 Third Party: Mission Critical operations based on IoT analytics

The “Mission Critical operations based on IoT analytics” (MiCrOBIoTa) project, aims at exploiting the INTER-IoT platform as a means to gather information from heterogeneous sensors in a converged way. As a result, Nemergent will integrate a new “IoT monitoring and analytics” component in its mission critical product portfolio, and especially into the Nemergent Control Room application. Figure 68 illustrates the overall Nemergent mission critical applications framework and the specific scope of the work proposed in MiCrOBIoTa.

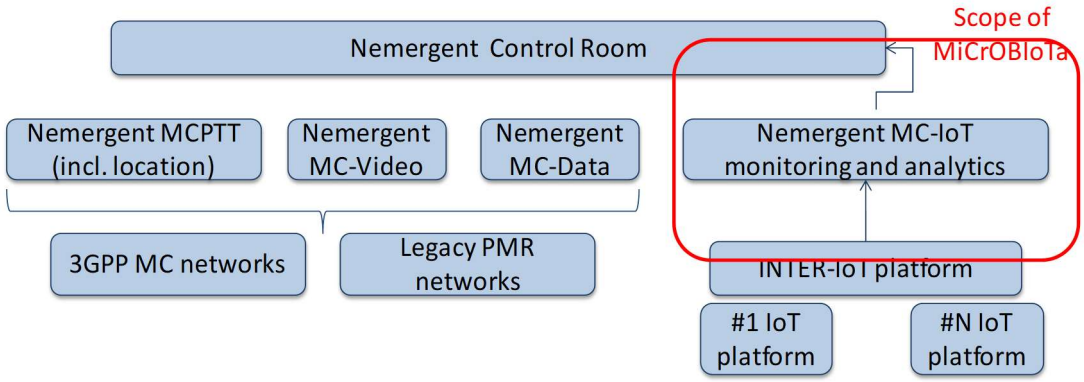


Figure 68: Scope of MiCrOBIoTa activities.

Figure 69 depicts an overall description of the main system to be used, tested and integrated in the scope of the INTER-IoT project.

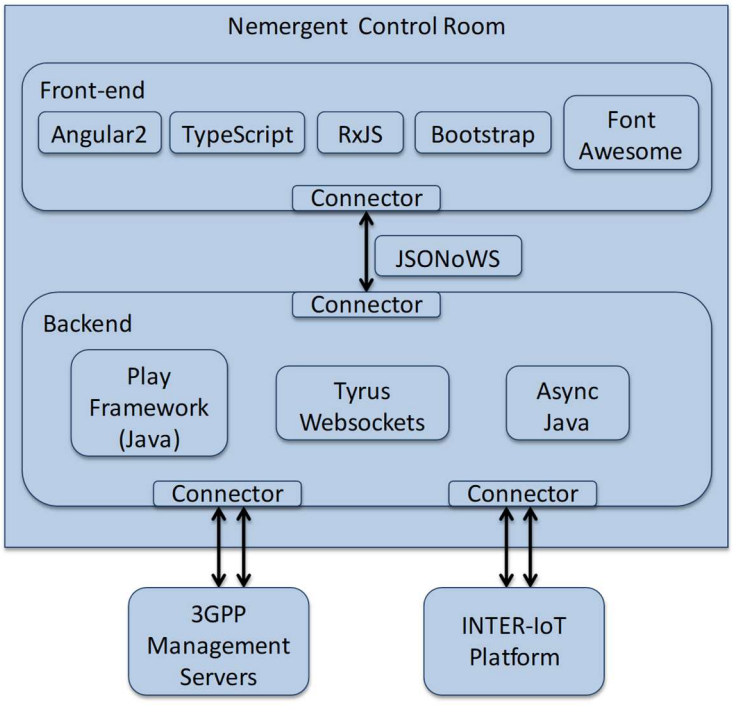


Figure 69: Overall system description.

The Nemergent Control Room system is made up of two main components.

The Nemergent Frontend component is an extensible HTML5-capable control interface that communicates through a Java backend with a plethora of different services and data providers. This component is built upon a series of state-of-the-art web technologies. The protocol used to sync real-time data between the Java backend and the HTML frontend is

defined as a series of JSON messages encapsulated through a WebSocket tunnel, in order to provide bidirectional real-time flows between the final user and the rest of the system.

A new provider has been added to the Nemergent Control Room, which handles all messages related to this project. This provider generates events that other frontend components can receive after subscription. For example, events can be created when new devices are received, which can be used for updating the map to include these devices.

The Nemergent Back-end is a Java-based component that implements most of the business logic related to data gathering and the specific interfaces to the different underlying systems. Similarly to the front-end, the back-end component uses several modern technologies and implements specific connectors to each underlying communications system.

In the scope of INTER-IoT, a new connector has been developed to cover the communication with the INTER-IoT platform. This connector implements all the necessary calls to the INTER-IoT API in order to gather the selected information with the corresponding message formats and data types.

Also, a new service has been created, which will decide what to do in response to each of the messages. Measurement data processing is done by a new module (analytics engine). For example, temperature variation can be monitored, raising an alarm when it exceeds a predetermined threshold.

The INTER-IoT – backend and backend – frontend interfaces are detailed in the appendix.

The anticipated benefits of interoperable “Mission Critical operations based on IoT analytics” are unquestionable. A typical situation in mission critical operations support systems is to include information coming from specifically deployed devices to gather environmental measurements. Examples of these devices are temperature sensors, meteorological and hydrological probes, traffic monitoring cameras, etc. We propose to add the Mission Critical IoT (MC-IoT) system, which includes a new monitoring and analytics component and an evolved Control Room interface tailored to the specific needs of the use case. In the case of a simulated crisis, significant information from on-body health-related sensors and port logistics devices will provide life-saving information to the mission critical operations support system. Besides, the available mission critical communications components can be used to demonstrate the crisis handling use case.

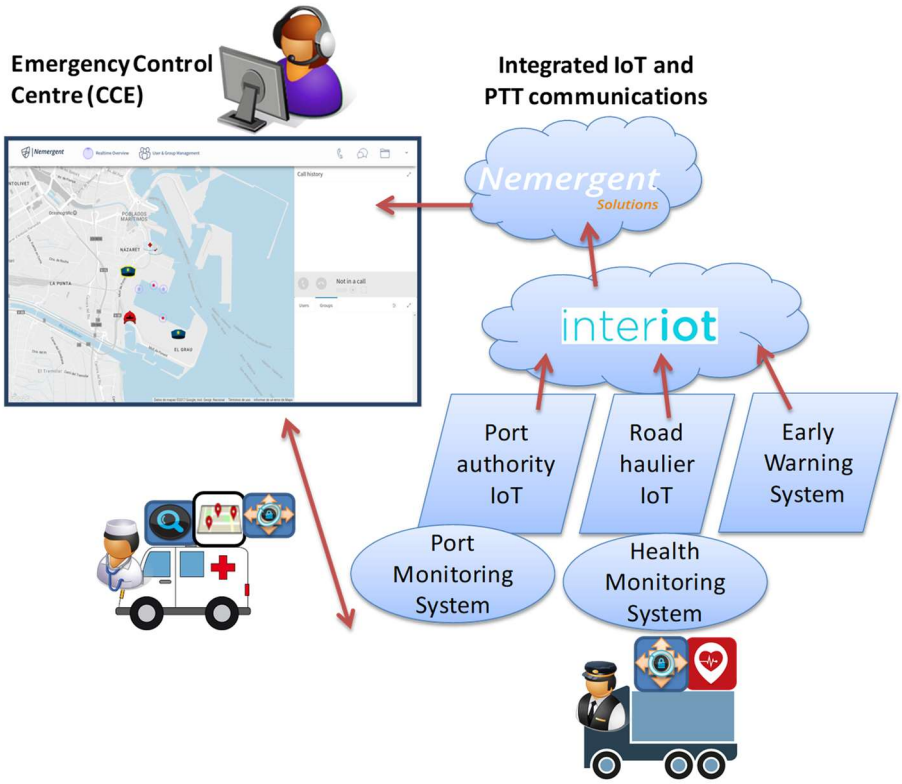


Figure 70: IoT-aided Mission Critical operations scenario.

Taking into account the overall picture and the availability of different IoT platforms, a complex use case could be created for an emergency simulation exercise. This scenario would include a typical emergency intervention, enhancing the operations support through the use of new communication technologies over commercial networks.

An example operational procedure is provided hereafter:

1. A road haulier comes into the port area. Upon an incident / health issue, the on-board health monitoring sensor reports the anomalous data to the INTER-IoT system through the road haulier company IoT platform.
2. The relevant data arrives at the Port Authority emergency control centre (CCE), which manages incidents taking place within the port and coordinates with other first responders (police, firefighters, ambulances, etc.).
3. The CCE operator accesses the MC-IoT system through the web-based Graphical User Interface (GUI), which will provide different types of icons for the different sources of information, and different views targeted at different emergency response units.
4. The CCE operator can use this platform to communicate with field response units (e.g., ambulance driver), providing them not only with location and navigation support but also with specific context information useful for the intervention.
   Besides the IoT-related data processing, the extended use case will make use of the Nemergent Mission-Critical Push-To-Talk (MCPTT) communication systems in order to resemble real-time communication between the different entities involved.

3.3.4.1 Integration of IoT framework

From a high level architectural standpoint, the integration of the MC-IoT external application is depicted hereafter.

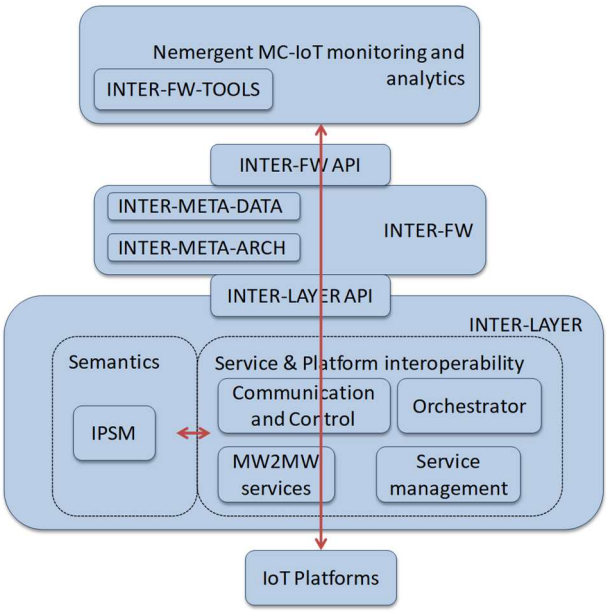


Figure 71: High-level perspective of the integration.

The MC-IoT system will run as an external application to the INTER-IoT platform. In order to access the heterogeneous data from different IoT platforms, the MC-IoT application needs to interact with the INTER-LAYER "Platform interoperability" functional component, which involves the "Communication and Control" and "MW2MW services" INTER-LAYER components. This MC-IoT component will also need to interaction with the "Semantics" functional component, which is implemented through the INTER-LAYER IPSM module. Additionally, the MC-IoT external application may need the use of composite IoT-related services. Thus, the invoking of the "Service interoperability" functional component may be needed. This would require the involvement of the INTER-LAYER "Orchestrator" and "Service management" components.

In order to interoperate with the INTER-IoT platform (e.g., registration, authorisation, etc.) and being able to invoke the API functions, the MC-IoT external application will make use of the INTER-FW tools and API. To some extent, the INTER-FW API acts as a wrapper of the INTER-LAYER API, exposing only those methods available to the external applications.

From a use case perspective, the external application interfaces with the INTER-IoT platforms in order to gain access to two / potentially three types of information:

1. Access to the port monitoring information (WSO2 platform).
2. Access to the health monitoring information installed on a haulier company (data coming from MyDriving and UniversAAL systems and integrated through Microsoft Azure platform).
3. Access to well-formed incident / emergency information, as potentially detected by a third party Early Warning System and submitted to the INTER-IoT platform.

The process to obtain data from INTER-IoT platform is as follows:

1. Authentication: A token must be obtained to make the requests in the following sections. A post request to the path /token with content "grant_type=password&username=**Username**&password=**Password"** \ and

Authorization header “Basic **Base64(consumer-key:consumer-secret)**” is done in order to obtain the token. Variables are marked in bold. These variables (Username, Password, consumer-key, consumer-secret) are provided beforehand. The authorization value is the Base64 encoding of the string consisting of consumer-key and consumer-secret separated by ‘:’. A client has also been provided, with a client id that is needed for the requests in the next steps. The calls to create this client or update it, if needed, are detailed in the appendix.

2. Get devices: With a get request to the path /mw2mw/devices all devices are obtained, with their id, name, location, etc. A platform can be optionally specified as a parameter to obtain devices from a specific platform. In this pilot, with this request the Valencia port INTER-IoT devices information will be obtained.
3. Subscribe to devices: Subscription to some of the devices obtained in the previous step must be done in order to receive events about the devices. This is done using a post request to the path /mw2mw/subscriptions, specifying in the content the deviceIds of the devices to subscribe to. A conversation id is returned to identify that subscription.
4. Get events: The method for obtaining the event data is specified when creating the client, and it can also be updated (with PUT /mw2mw/clients/{managedClientId})
    1. Pull: Messages will be obtained by making periodical POST requests to /mw2mw/responses.
    2. Push: A callbackUrl is specified for the client (can also be updated, like push or pull policy). The server will send the messages to that url, using a POST request.

   For example, a subscription to a weather measurement device in the Valencia port can be performed. After that, messages will be received, with the measurement data, such as average temperature, wind speed, etc.

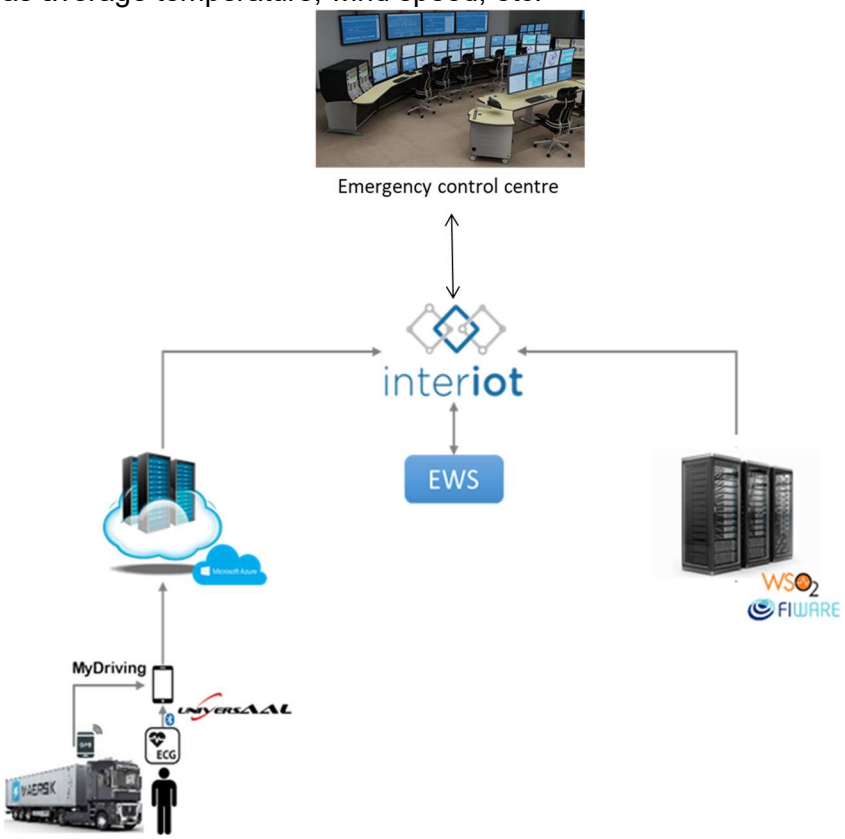


Figure 72: Use case perspective of the integration.

Following the proposed pilot, the trucks will be monitored once they are in the port facilities. In case an accident or a medical problem is detected, the Early Warning system will publish

a notification through INTER-IoT in a standard format (EDXL). Once the emergency control centre receives the notification, it can start communication with the driver with a push to talk protocol in the driver's mobile.

The main benefits we can get from this scenario are: apply in the port communications a standard format in accident reporting like EDXL, real time identification of the location of the accident, direct communication with the closest control centre when an accident occurs and monitoring driver's health if it is necessary.

3.3.4.2 Deliverables and version overview

The following table contains a deliverable list which need to be signed of before SAT testing commences.

ID	Description	Check
Documents		
1	Validation and Test reports of the IoT system components	
2	Validation and Test reports of the Pilot system components	
Hardware		
3	Gigabyte Barebone on which the Nemergent System is installed and configured	
4	Additional Laptop with basic Internet Browsing Capabilities	
5	Connection cables (Ethernet connection between both systems and Internet)	
Tools		
6	Preferably a recent Linux OS	
7	TestNG Java	
8	Jenkins 2	

Table 1: Deliverable checklist

The following table shows the software components and version of which the system release version 1.0 consists of.

ID	Description	Version	Check
Nemergent CtrlRoom Frontend			
1	JSONoWS API	V1.3.0	
2	WebRTC API	V1.0.0	
3	GUI	V1.0.0	
Nemergent CtrlRoom Backend			
4	JSONoWS API	V1.3.0	
5	MCPTT MS API	V1.0.1	
6	INTER-IoT API	V1.0.0	

Table 39: Component version overview

3.3.4.3 Requirements, scenarios and use cases

The following table provides an overview of the relation between the requirements and the test(s) that validate their implementation.

ID	Description	Covered by
47	API for third-party developers	T1.1.1, T1.1.2, T1.2.1, T1.2.2, T1.2.3, T1.2.4, T1.2.5, T1.3.1
51	API for data publication	T1.2.3, T1.2.4, T1.2.5, T1.3.1
53	Location of sensor and measurement is included in semantic models	T1.2.1, T1.2.2
69	Confidentiality	T1.2.5
123	Use of standards	T1.1.1, T1.1.2, T1.2.3, T1.2.4, T1.2.5, T1.3.1

Table 2: Requirements vs test mapping

The following table provides an overview of the relation between the scenarios and the test(s) that validate their implementation.

ID	Scenario name	Covered by
9	Accident at the port area	T1.2.1, T1.2.3, T1.3.1
How to access the health information of the truck driver is still to be defined. This scenario cannot be developed without this information.		
32	Third party developer using INTER-FW to access data from two different platforms	T1.1.1, T1.1.2, T1.2.1, T1.2.3, T1.2.4

Table 3: Scenario vs test mapping

3.3.4.4 Test environment

Introduction

To test the functionality of the integrated Accident at the port area in combination with the IoT framework representative test hooks in the system are needed. This chapter will describe this environment and the used hardware, software, tools and platforms.

Test environment

This paragraph describes the test setups, hooks and probes used in the system used during this SAT.

Test tools, hooks and probes

This paragraph describes the test setups, tools, hooks and probes used in the by this document defined tests.

A test setup defines a setup used for a test, and describe the used system parts, interface etc.

Tools define the used COTS tooling that will be used during testing e.g. Logic analyzers, packet sniffer like e.g. Wireshark, etc.

Hooks are used to inject data or control parts of the system used to get the system to handle a needed scenario. This can be hidden, debug or service interfaces which contain the needed functionality.

Probes are used to gather logging from parts of the system under test to provide system feedback and to prove that the system is handling the scenario as it should.

The following paragraphs will define the used setups, tools, hooks and probes which will be referred to by the test descriptions.

TS_01 – Dedicated Docker Image

All the Nemergent testing activity will be performed inside an ad-hoc developed Docker image, containing all the necessary software tools to perform this testing phase.

This image will contain the Jenkins instance, and will have a copy of the software repositories freshly obtained in the moment of the Docker image building. This way, Nemergent can assure maximal compatibility between hardware vendors, software distributions and network environments. This image itself is prone to be tested inside a continuous integration scheme.

With a command launched in the host Linux system, the integrated Jenkins tool will execute all the tests sequentially, making detailed reports of the results available through its web interface, along with performance measurements and precise timings. This Jenkins builds will archive as artefacts all the network recordings, profiling files and log files produced during the test executions.

The Nemergent CtrlRoom software will communicate with the INTER-IoT components through the virtual network interface offered by Docker to the container, helping with the network debugging issue.

TS_02 – Installed system and person monitoring physical environment

When testing final phase tests, where a person is required to operate the system and another one has to be situated on field monitoring the real-life results of the whole project, a real-time distant communication technology is to be used. If the project is deployed along Nemergent's cutting-edge MCPTT ecosystem, it could be used for reliable voice PTT communications between the field monitoring person and the staff operating the ControlRoom. Other options available as legacy PTT radio, regular phone call or VoIP systems could be also used.

The staff operating the control room shall indicate the test number and/or test description to the field monitoring person, and then wait for this other person to report field environment changes back to the control room staff, in order to assess the test successful execution.

TT_01 - TestNG Java

As a testing environment we will use different suites of TestNG Java testing framework, which will implement the different business logic associated with the working model. For example, a different test suite can be generated for each different EDXL document type, thus assuring maximal code coverage for each module and case.

TT_02 - Jenkins 2

The TestNG Java tests will be orchestrated though a Jenkins 2 server, using the newly released Pipelines functionality, providing expressive description of the test case scenarios.

TH_01 - Jenkins Pipelines

Jenkins Pipelines definitions support parameterized runs, which can be used to inject relevant EDXL messages right into the message broker, depending on target of the tests.

TP_01 – Protractor

Graphical testing can be performed through the implementation of Protractor tests. These tests can assure that frontend graphics being rendered on a virtual screen corresponds to what the requirements requires. This would provide similar confidence as a real human visual testing, for example that a specific EDXL document triggers the area and information painting on the Port Control Centre.

3.3.4.5 Test description

Startup tests

T1.1.1 Boot up and first contact

ID	T1.1.1
Test	Nemergent MC-IoT Module boots up and contacts INTER-IOT Platform
Type	System Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is not launched
Req.	[47], [123]
Input	Launch the Jenkins job labelled as “**T1.1.1**”
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

T1.1.2 INTER-FW Authentication

ID	T1.1.2
Test	Nemergent MC-IoT Module authenticates correctly against INTER-FW gateway.
Type	System Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is already launched, but module has only made first contact.
Req.	[47], [123]
Input	Launch the Jenkins job labelled as “**T1.1.2**”
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section:

	• backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

Basic functionality tests

T1.2.1 Obtain a list of trackable entities

ID	**T1.2.1**
Test	Nemergent MC-IoT Module queries INTER-FW for a list of authorised trackable entities.
Type	Service Discovery
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [53]
Input	Launch the Jenkins job labelled as "**T1.2.1"**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

T1.2.2 Paint all the trackable entities obtained

ID	**T1.2.2**
Test	Nemergent MC-IoT Module passes the processed data to the Nemergent CtrlRoom main module and paints the listed entities on a map.
Type	System Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [53]
Input	Launch the Jenkins job labelled as "**T1.2.2"**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt

	• test_report.html
Outcome	Pass / Fail

T1.2.3 Subscribe to the event streams

ID	T1.2.3
Test	Nemergent MC-IoT Module performs subscription for the appropiate entities in order to maintain updated information about them, but without over saturating network and system resources.
Type	System Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51], [123]
Input	Launch the Jenkins job labelled as "**T1.2.3"**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

T1.2.4 Receive updates from INTER-FW

ID	T1.2.4
Test	Nemergent MC-IoT Module passes the processed data to the Nemergent CtrlRoom main module and repaints the listed entities on a map.
Type	System Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51], [123]
Input	Launch the Jenkins job labelled as "**T1.2.4"**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

T1.2.5 Access control to resources

ID	T1.2.4
Test	CtrlRoom backend controls what resources can be accessed by which users. Using the frontend client, this test will try to access resources, such as list of devices or their information, as an unauthorized user.
Type	System Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51], [69], [123]
Input	Launch the Jenkins job labelled as "**T1.2.5"**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html

End to end tests

T1.3.1 Accident at the port area

ID	T1.3.1
Test	Accident or medical emergency is simulated, via tampering with the related sensors
Type	End to end Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51], [123]
Input	The field monitoring person simulates the medical emergency via tampering with the sensors so they receive a signal that would indicate a medical emergency and notifies the control room staff of when the tampering has been performed to ensure that the alert occurs within a reasonable time frame. Additionally launch the Jenkins job labelled as "T_others"
Output	The alert is received in the control room, with the proper information about the truck and container status. Jenkins job output which consists of core components logs and network captures.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

CtrlRoom failure testing

T1.4.1 Connection drops between frontend and backend

ID	T1.4.1
Test	Connection drops can occur occasionally between frontend and backend, which can cause problems if not handled well. The system should buffer the messages pending that couldn´t be transmitted when the socket was closed to send when the reconnection happens. This test will purposefully close for a time and then reopen the socket, while doing different requests from frontend to backend, and check that the system recovers correctly (messages are sent, although late, and all exchanges complete without errors).
Type	Failure Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51]
Input	Launch the Jenkins job labelled as “**T1.4.1”**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

T1.4.2 Authentication error

ID	T1.4.2
Test	Incorrect credentials will be provided in the authentication. Three scenarios will be tested (only username incorrect, only password incorrect or both password and username incorrect). Additionally, fringe cases would be tested, such as what happens when username and/or password are left blank. The expected result is a message indicating an authentication failure being shown to the user and no further action taken.
Type	Failure Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51]
Input	Launch the Jenkins job labelled as “**T1.4.2”**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt

	• test_report.html
Outcome	Pass / Fail

T1.4.3 Containment in login page while unauthenticated

ID	T1.4.3
Test	While the user is not authenticated, pages outside the login page should be unreachable. This tests will make petitions to each of the pages except the login page while being in the unauthenticated status to ensure that all do nothing but redirect to the login page.
Type	Failure Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51]
Input	Launch the Jenkins job labelled as "**T1.4.3"**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

T1.4.4 Subscription to device error

ID	T1.4.4
Test	This test will try to request subscription to a device that doesn´t exist. A message should be showed to the user making the subscription indicating that error.
Type	Failure Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51]
Input	Launch the Jenkins job labelled as "**T1.4.4"**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

Interface failure testing

T1.5.1 Recover from invalid token state

ID	T1.5.1
Test	For requests that use token for authorization, upon receiving a response with status code unauthorized the token is refreshed automatically and the request is repeated with a valid token. This test checks that making each of these requests using an invalid token ultimately ends with the proper response, after obtaining a new token
Type	Failure Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51]
Input	Launch the Jenkins job labelled as "**T1.5.1**"
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

T1.5.2 INTER-FW Connectivity loss

ID	T1.5.2
Test	Check that, when repeated requests fail to contact the INTER-FW server, INTER-IoT functionality gets temporarily halted, and frontend receives notification about it. After connectivity is recovered, frontend should receive a notification about the end of the downtime, and all INTER-IoT functionality should resume properly
Type	Failure Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51]
Input	Launch the Jenkins job labelled as "**T1.5.2**"
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

T1.5.3 Malformed subscription event responses

ID	T1.5.3
Test	A malformed event response (received because of having subscribed to devices. E.g weather measurements) should be ignored, and further processing should not be performed. In case of pulling multiple messages at the same time, a malformed message should not affect the processing of the rest of the messages
Type	Failure Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51]
Input	Launch the Jenkins job labelled as "**T1.5.3"**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

T1.5.4 Health monitoring information unavailable

ID	T1.5.4
Test	When the system providing health monitoring information is unavailable, or not working correctly, and thus not sending the needed information, functionality should stop working gracefully, and this incident should be logged. Additionally, frontend clients should receive an alarm indicating this problem, with the maximum severity
Type	Failure Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51]
Input	Launch the Jenkins job labelled as "**T1.5.4"**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

T1.5.5 Malformed devices

ID	T1.5.5
Test	Get devices response should contain all required fields, with proper values (Location must specify valid latitude and longitude, id, name and hostedBy must be strings, etc.. This test will check that, for each type of malformation, the backend behaves correctly, which is defined as logging the incident and not doing any further processing. A malformed device information should not be sent to the frontend.
Type	Failure Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51]
Input	Launch the Jenkins job labelled as "**T1.5.5"**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

T1.5.6 Values out of valid ranges

ID	T1.5.6
Test	For event data fields, values will be checked for validation. These are some of the validation checks that can be performed: • Emissions cannot be negative • Wind speed cannot be negative • Dates must follow the proper date format. When receiving event data with invalid values, the message will not be processed further, and an alert will be sent to the frontend, indicating the reason for the message being invalid alongside the device id related to the event data that caused the error.
Type	Failure Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51]
Input	Launch the Jenkins job labelled as "**T1.5.6"**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt

	• test_report.html
Outcome	Pass / Fail

T1.5.7 Trucks exited / entry correspondence

ID	T1.5.7
Test	Exited vehicles must be listed on a later date as entry vehicles. A vehicle cannot be listed as exit vehicle or entry vehicle twice without its corresponding entry/exit. This test will check that, when this occurs, a message is generated indicated the anomalous situation to the frontend, and that this incident is logged.
Type	Failure Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51]
Input	Launch the Jenkins job labelled as "**T1.5.7"**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

T1.5.8 Event must have a device

ID	T1.5.8
Test	Events must have a device associated. Events with a device id not corresponding to an existing device must be discarded, without further processing.
Type	Failure Testing
Setup	Need test setup TS_01
Start	Nemergent CtrlRoom is properly launched and authenticated
Req.	[47], [51]
Input	Launch the Jenkins job labelled as "**T1.5.8"**
Output	Check Jenkins job output, whether the job has succeeded or it has failed. Check outputted artifacts for debugging purposes.
Logs	Jenkins Job artifacts section: • backend_log.txt • frontend_log.txt • network_eth0.pcap • network_capture_logs.txt • test_report.html
Outcome	Pass / Fail

3.3.4.6 Test outcome overview

The following table will provide an overview of the test result of all the performed tests in this SAT.

Test	Description	Outcome
T1.1.1	Boot up and first contact	Pass / Fail
T1.1.2	INTER-FW Authentication	Pass / Fail
T1.2.1	Obtain a list of trackable entities	Pass / Fail
T1.2.2	Paint all of the trackable entities obtained	Pass / Fail
T1.2.3	Subscribe to the event streams	Pass / Fail
T1.2.4	Receive updates from INTER-FW	Pass / Fail
T1.2.5	Access control to resources	Pass / Fail
T1.3.1	Accident at the port area	Pass / Fail
T1.4.1	Connection drops between frontend and backend	Pass / Fail
T1.4.2	Authentication error	Pass / Fail
T1.4.3	Containment in login page while unauthenticated	Pass / Fail
T1.4.4	Subscription to device error	Pass / Fail
T1.5.1	Recover from invalid token state	Pass / Fail
T1.5.2	INTER-FW Connectivity loss	Pass / Fail
T1.5.3	Malformed subscription event responses	Pass / Fail
T1.5.4	Health monitoring information unavailable	Pass / Fail
T1.5.5	Malformed devices	Pass / Fail
T1.5.6	Values out of valid ranges	Pass / Fail
T1.5.7	Trucks exited / entry correspondence	Pass / Fail
T1.5.8	Event must have a device	Pass / Fail
SAT Outcome		**Pass / Fail**

Table 4: Test outcome overview

3.3.4.7 Integration ethics and security

Introduction

This section addresses recommendations and proposed solutions to the ethical and security points of the integration.

Mission Critical operations based on IoT analytics

During the pilot execution, all the sensitive information (e.g., making reference to the INTER-IOT system or to personal data of people taking part of the pilot) should be hidden from non-authorized eyes at the moment of displaying the information at the Nemergent CtrRoom. The Nemergent CtrlRoom, as the user-endpoint of the system, is the last step in a chain of information processing, where all the information should have been secured and automated by design. Thus, special effort will be paid to ensuring that user privacy and system security is enforced at the Nemergent CtrlRoom GUI.

3.3.5 Third Party: Early Warning System (EWS)

The goal of the H2020 INTER-IoT project is to support interoperability between heterogeneous IoT platforms across the logistics and e-health domains [1]. For demonstration and validation purposes, the project described scenarios to decrease the risk of fatal accidents at the port of Valencia, improving health prevention and enabling quick reaction by reducing time response [2]. The goal of this scenario is to exploit how e-Health and e-Care can use IoT platforms dedicated to logistics to prevent the occurrence of accidents and to support evacuation or attention in case of emergency situations.

An early warning system (EWS) is a distributed system that monitors the physical world and issues warnings if it detects abnormal situations. The Internet-of-Things (IoT) offers opportunities to improve monitoring capabilities of EWS and to realize (near) real-time warning and response. The INTER-IoT-EWS goal is to detect accident risks with trucks that deliver goods at the Valencia port area, interoperating different IoT platforms. The solution addresses the semantic integration of a variety of data sources with processing in safety-critical applications for effective emergency response. The solution considers existing domain-specific ontologies and standards, along with their serialization formats. Accident risks are assessed by monitoring the drivers' vital signs with ECG medical wearables, and the trucks' position with speed and accelerometer data. Use cases include the detection of health issues and vehicle collision with dangerous goods. This EWS is developed with the SEMIoTICS framework, which is composed of a model-driven architecture that guides the application of data representations, transformations and distributed software components. This framework enables EWSs to be a semantic broker for situation-aware decision support.

Early Warning System (EWS)

An EWS is a system for "the provision of timely and effective information, through identified institutions, that allows individuals exposed to a hazard to take action to avoid or reduce their risk and prepare for effective response" [3]. An effective EWS must be people-centered and integrate knowledge about the risks, risks' monitoring and warning, dissemination of meaningful warnings and public awareness [4]. Modern EWSs comprise software and hardware for data acquisition, situation awareness, decision-making, and information dissemination. Current experimental prototypes incorporate IoT technology to improve their functionality [5]. The conceptual architecture of EWS typically consists of three parts [3, 5, 6] (Figure 1, top):

(i) Upstream data acquisition: Distributed sensor systems transform observations into digital signals, pre-process the associated data values to ensure that they represent relevant information for decision making and transmit these data values to a message- and/or event-oriented middleware (broker).

(ii) Decision support: The data is stored in a data storage and is subjected to rules to detect situations of interest. The rules are represented as models, which can be deterministic (rule-based) and/or non-deterministic (machine learning) approaches. Once a situation is detected, the EWS considers the requirements of the alert targets to assess the risk and determine the emergency response.

(iii) Downstream information dissemination: Different target groups, comprising humans (e.g. the public) and machines (e.g. sirens), receive adequate messages.

Interoperability is an important feature of effective EWSs for the integration of internal components and interworking of different EWSs. The level of interoperability depends on the standardization of interfaces, data exchange formats and protocols [6]. The design problem

to be addressed here is the improvement of IoT EWSs' interoperability with data sources, alert targets, and other EWS to detect emergency risks.

INTER-IoT-EWS

Requirements and use cases

The requirements given in the scenario are:

(IIOT-FR1) IoT platforms should be able to coordinate with emergency systems by detecting accidents and accident risks with trucks within the port area. The EWS should be able to identify vehicle collisions and severe changes of the driver's cardiac behavior, alerting their urgency and severity to multiple targets. The acceptance criteria is to check if the EWS, built on top of IoT platform(s), is able to coordinate with emergency systems through emergency interoperability standard(s).

(IIOT-FR2) The haulier IoT platform and the port IoT platform should be able to share health information about the driver, monitored in real-time through an electrocardiography (ECG) device. The solution should be able to provide both raw and calculated data, e.g. ECG sequence (time series) and heart rate (HR). These data need to be integrated in a way that the port emergency control system can consume them. The acceptance criteria is to check if the EWS, at the application level, is able to process the health data to identify cardiac issues and be able to send data to emergency systems based on emergency interoperability standard(s).

(IIOT-NFR1) IoT platforms should be semantically and syntactically interoperable. The solution should be able to integrate the involved IoT platform(s) in a way that their data syntax and semantics are understandable, i.e. can achieve common understanding among the participating parts. The acceptance criteria is to check the use of a mechanism to translate data syntax (lexicon) and semantics of messages exchanged.

(IIOT-NFR2) E-Health and logistics should be integrated at the INTER-IoT application and semantics (A&S) level, including primitives for data interpretation of medical and logistics data. The acceptance criteria is to check the use of semantic models to represent e-health and logistics data within the IoT use cases (listed below).

(IIOT-NFR3) The energy consumption (battery level) of the devices being used for the situation identification mechanism should be monitored. The acceptance criteria is to check if the solution is able to provide energy consumption data for the application level, consumed by the EWS.

Five use cases were conceived to test these requirements:

(UC01) Vehicle collision detection: use of accelerometer data of the truck from mobile phone and health device;

(UC02) Hazardous health changes: detect occurrences of stress and arrhythmia (e.g. bradycardia and tachycardia);

(UC03) Temporal relations between UC01 and UC02: detect if a health issue occurred before, during or after a vehicle collision;

(UC04) Accidents with dangerous goods: monitor dangerous goods being transported (according to UN list of dangerous goods) in all use cases (1-3), adding the adequate information regarding emergency procedures for effective response.

Note that UC03 has situations that require the integration of data from both domains (health and logistics) and can represent complex behaviours. For example, there is a possibility that bradycardia is detected followed by continuous decrease of the heart rate after a vehicle collision is detected. This situation reflects a car collision where the driver got injured and is classified as extreme severe with immediate urgency. In this situation the vehicle collision will be identified with both accelerometers from the ECG device and from the smartphone, considering device features, as accuracy and energy consumption.

Semantic IoT EWS framework

The "SEmantic Model-driven development for IoT Interoperability of emergenCy serviceS" (SEMIoTICS) is a framework to improve the semantic interoperability within and among EWSs [7, 8]. It consists of an architecture (Figure 1, bottom), technologies and guidelines based on model-driven engineering (MDE) inspired in [9]. SEMIoTICS uses the Endsley's situation awareness theory [10], which is harmonized with the Unified Foundational Ontology (UFO) [7] and aligned to the semantic healthcare system architecture [11].

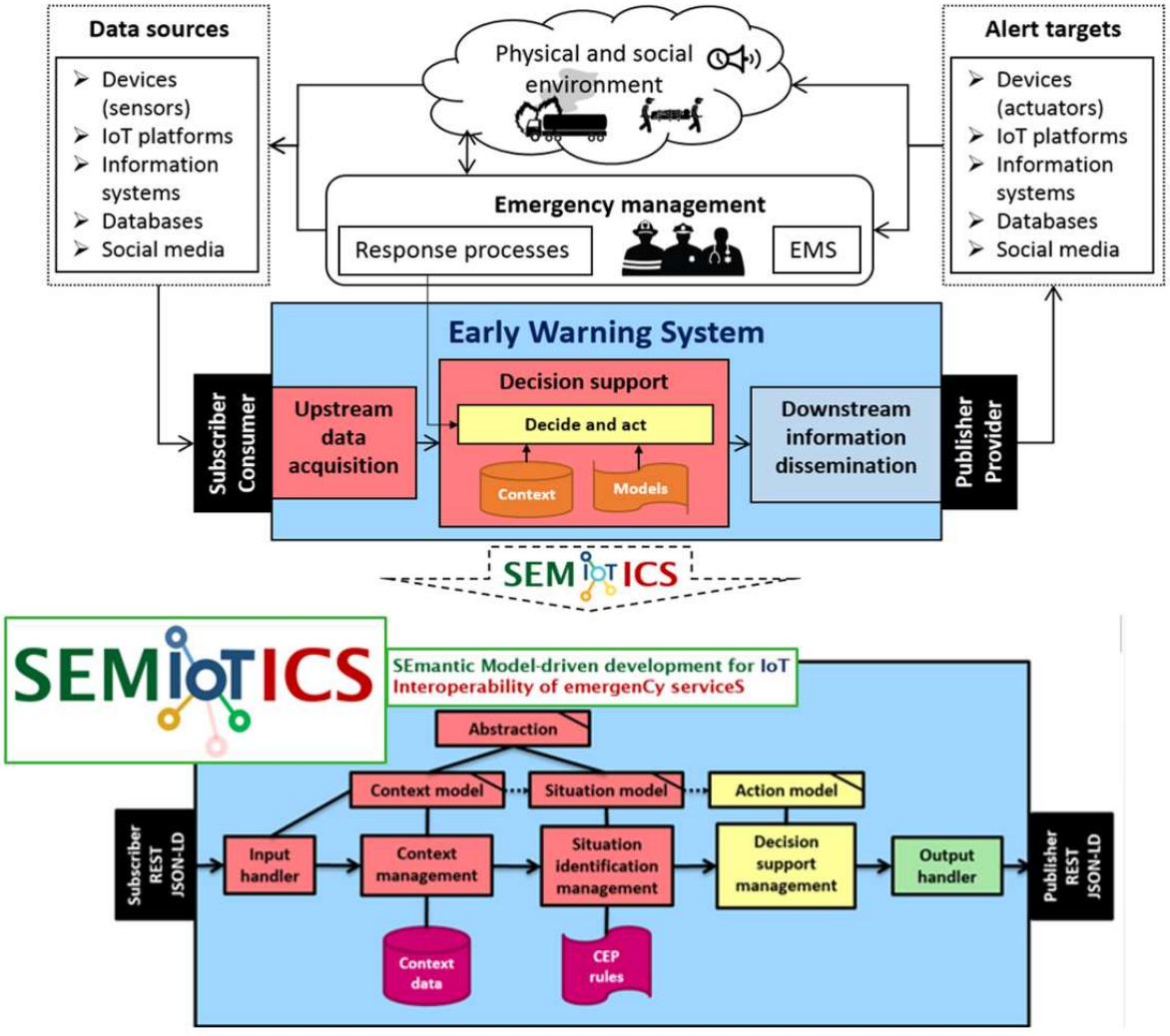


Figure 73: Typical EWS architecture (top) and the SEMIoTICS architecture (bottom).

The architecture has six elements addressing the 3 main functions of an EWS:

(1) Input handler: upstream data acquisition through adaptors;

(2) Abstraction: foundational ontology;

(3) Context model: domain ontology;

(4) Situation model: complex event processing;

(5) Situation awareness: data flows,

(6) Output handler: downstream emergency notification.

It follows the publisher/subscriber pattern and RESTful services with JSON-LD.

Adaptors are implemented as syntactic and semantic translations [12]. The input handler is responsible for message translation, which relies on the syntax of each ontology being used and, therefore, will also require semantic as well as syntactic translations, e.g. from RDF/XML to JSON-LD and from HL7 to EDXL. Messages are translated from the original ontologies to our context model (core ontology), which is aligned to W3C SSN and incorporates terms from EDXL and HL7. This approach aims on optimizing the data and semantics maintenance when integrating distinct domains. The abstraction component refers to foundational ontologies, which are designed to maximize the support for interoperability of high level categories, e.g. event, process, physical object and system. The core ontology and SSN are grounded on the UFO (through OntoUML) and DOLCE Ultralite (DUL), respectively [7]. UFO and DOLCE share the same definitions for some conceptualizations, facilitating the alignment between the ontologies extended with them.

The situation model is responsible for the situation identification mechanism, i.e. the formalization of the emergency risk detection [13]. We adopted a rule-based approach, allowing the specification and implementation of complex event processing (CEP). CEP is a common component of IoT platforms to correlate data using temporal predicates (events' relations), as Cepheus, the CEP engine of FIWARE IoT platform. Guidelines describe how CEP technologies can implement the situation models, e.g. in Java ESPER[12] and Drools Fusion technologies [14]. Decision support is enabled by the adoption of a workflow management system that enables the end user to design business processes, e.g. emergency plans, as data flows. Big data integration tools for workflow development automatically generates code and is able to deploy data flows at runtime, e.g. Node-Red[13]. This component also covers the deployment and execution of the data flows for decision making. The output handler is responsible for brokering the emergency risk notifications to the correct targets, according to the emergency procedures defined on the decision support component. For each predetermined risk, targets are enumerated with their information requirements. The data format of the notifications follows EDXL standards serialized as JSON-LD. The risk notification services are exposed as data publishers.

Solution

The solution architecture (Figure 74) includes the Shimmer ECG 3 device[14] to collect ECG data from drivers. This device has high accuracy and usability, and provides a mobile application (Shimmer Capture Xamarin), which uses the Shimmer API to transmit data from the ECG device to a smartphone, enabling the mobile device to play the role of a "field

[12] http://www.espertech.com/

[13] https://nodered.org/

[14] http://www.shimmersensing.com/products/ecg-development-kit

gateway". This mobile app was improved with (a) semantic technologies, being able to receive the data from the ECG device and enrich the data semantics with an extension of the ETSI SAREF[15] ontology (SAREF4health), and with (b) device-to-cloud connection, being able to send the semantically enriched data as JSON-Ld messages to a cloud gateway ("context broker"), i.e. the Microsoft Azure IoT Hub.

Similarly, the MyDriving mobile application for logistics (open use case of Azure IoT platform[16]) transmits the data about the truck position, speed, acceleration and goods information to the cloud infrastructure. These logistics data are serialized as JSON-LD messages, following the structure of SAREF ontology aligned to LigiCO ontology[17]. SAREF was chosen because of its capabilities for tracking devices' energy consumption. IPSM module is responsible for syntactically and semantically translate these data: from JSON-LD to the INTER-IoT JSON-LD syntax, which is structured JSON-LD (two @graph) with middleware information; and from SAREF to the INTER-IoT core ontology semantics, which is aligned to SSN/SOSA. These translations are configured priory in IPSM by the application developer through a REST service.

The data represented as INTER-IoT JSON-LD syntax and INTER-IoT core ontology semantics are published in the broker in a topic, which the EWS subscribes to receive real-time data. Then, the EWS input handler certifies whether new translations to harmonize the data in the SEMIoTICS core ontology are necessary and, if so, the input handler requests the translations to IPSM.

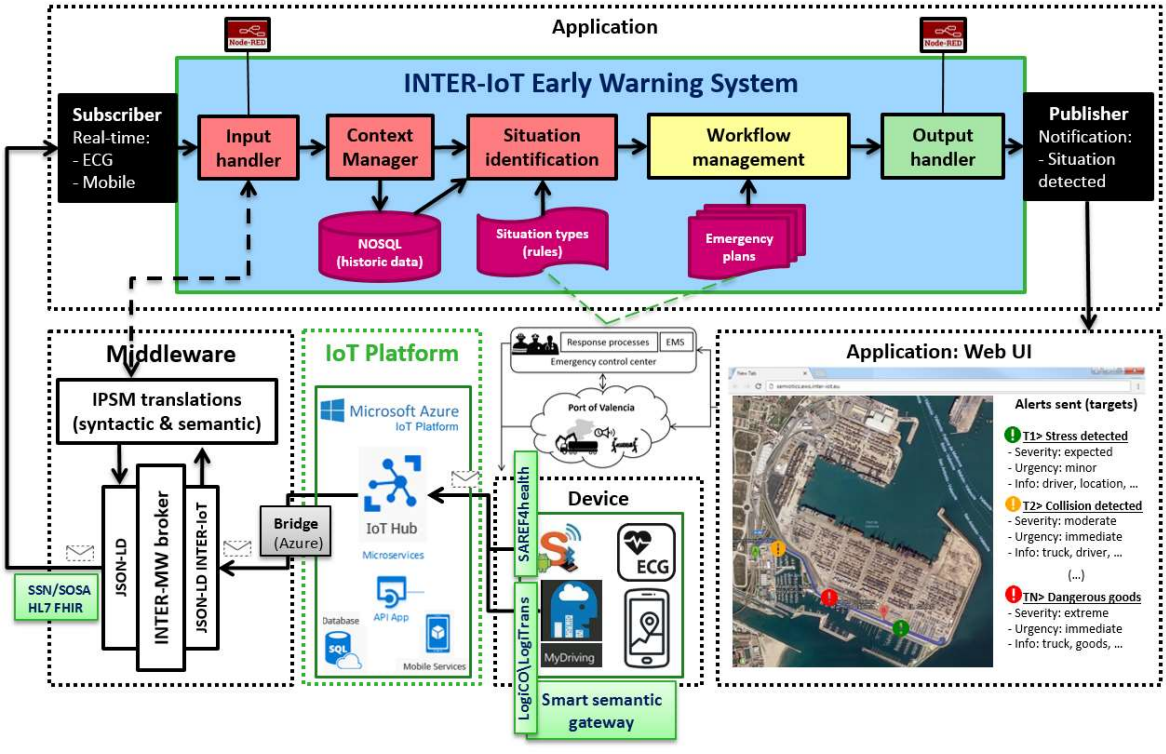


Figure 74: INTER-IoT-EWS to detect accident risks and accidents at the port of Valencia.

[15] http://ontology.tno.nl/saref/

[16] https://azure.microsoft.com/en-us/campaigns/mydriving/

[17] http://ontology.tno.nl/logico/

The data is annotated with the core ontology and stored in a NOSQL database (MongoDB) for historic data storage. Both real-time data and historic data are used by the risk (situation) identification management component, i.e. the NESPER CEP engine [15]. Situation types are defined a priori, as rule sets, describing the risky situations of interest based on emergency plans. Each situation type is linked to a response process, i.e. the specific workflow to be executed once a situation is identified. Therefore, the risk identification component triggers the workflow management, which executes the linked processes. The workflow component is responsible for checking the information requirements of each alert target, passing this information to the output handler, which is responsible to transform the data to EDXL compliant messages semantically enriched. Therefore, the output handler enables the brokering of notifications of situations detected, following the JSON-LD syntax and the EDXL structure, which is able to link to the semantics used. A web UI application shows each alert sent by the EWS with its severity and urgency, and other information, including the targets that received the notification and the message sent to each target.

The EWS is developed with Node-Red (input and output handlers) and a .NET/C# REST application that executes (asynchronously) the context data management, the situation identification manager and the decision support management components (to respond to situations identified, formatting the messages according to information requirements of the targets). Table 1 summarizes the involved data.

External	Health	Logistics
Data	Driver's ECG, accelerometer	Position, speed, accelerometer, goods
Device	Shimmer (SPINE), Mobile	Mobile (MyDriving Android or iOS)
IoT platform	MS Azure IoT	MS Azure IoT
Ontologies	ETSI SAREF4health, HL7 FHIR	ETSI SAREF, LogiCO, LogiTrans

Table 40. Data sources.

Validation plan

Validating the achievement of providing effective situation awareness and emergency response requires a comparison whether the response processes triggered through the workflow management is adherent to emergency procedures, reflecting pragmatic interoperability between the EWS and an emergency manager. This will measure whether the system works for the intended risks' detection and warning. So, it includes simulation of the use cases (test cases) with multiple target groups with different information requirements.

The validation plan is organized as

(a) Factory acceptance tests: in a lab environment the EWS will be deployed in a cloud environment and the components integration will be tested through mock objects; and

(b) Site acceptance tests: a pilot in the port, where accidents will be simulated in accordance with the port emergency exercises, e.g. vehicle collision through hard breaks, bradycardia/tachycardia by decreasing the thresholds and adequate response procedures for accidents with dangerous goods.

The validation plan includes the performance evaluation of data transfer, processing and storing JSON-LD as payload. Total time to be observed:

(i) for data acquisition;

(ii) to semantically translate a message;

(iii) to syntactically translate a message for the POCO used by the CEP engine;

(iii) to process data (from CEP working memory) and process (serialize and deserialize) for risk identification; and

(iv) to create the alert messages, i.e. serialize the output data as EDXL semantic model.

(v) The brokering performance will be validated in terms of scalability for single cluster and multi-broker, having the semantic IoT EWS approach [5] as a baseline.

Validating the achievement of semantic interoperability depends on whether the components of the solution have the same "understanding" of the data. Since the approach is based on multiple semantic translations, the translation accuracy will be measured to calculate the semantic interoperability indicator. The accuracy reflects the semantic loss when translating a message from A to B, which is measured by executing the transformations in sequence from ontology A (e.g. SAREF) to ontology B (e.g. SSN) and from B to A, i.e. check how x is different to $T(T(x)_{A>B})_{B>A}$, where $T(x)_{A>B}$ represents the semantic translation function from A to B [12].

This plan includes data management with the FAIR data principles.

3.3.5.1 Integration of INTER-IoT components

The layered approach of INTER-IoT is reflected in Figure 74 (device, network, middleware and application layers). The scope of our participation in the project, agreed in the beginning of the collaboration, is twofold: (1) the implementation of an EWS at the INTER-IoT application level, consuming the data provided by the IoT platforms integrated with INTER-IoT, and (2) the support on the definition of the semantic alignment (translations) between W3C SSN/SOSA and ETSI SAREF, the two main standardized IoT ontologies. However, since the IoT platforms used in the Valencia port case would not be able to provide cardiac data, required by IIOT-FR2, we gave support in making these data available through our own e-Health IoT platform, which is built on top of MS Azure IoT and Shimmer3 ECG device.

At the device level, our solution uses (per truck-driver) one Android smartphone and one Shimmer3 ECG unit. The smartphone device plays the role of a semantic field gateway, having two apps merged (Shimmer3 Capture Xamarin and MyDriving) and improved with semantic technologies, as described in section 3.

In the network layer, the ECG device sends data to the smartphone through Bluetooth, using the Shimmer API. From the smartphone to the cloud, either Wi-Fi or 4G is used for connectivity.

In the middleware layer, the main component is the MS Azure IoT Hub, a pub/sub middleware (context broker), which receives data from the smartphone. One IPSM alignment was built to support the translations between SAREF and SSN/SOSA. Two approaches were planned:

(1) The EWS consuming data directly from the Azure IoT Hub. In this case the EWS Input Handler is able to execute the semantic translations through two translation mechanisms, configured when deploying the EWS: (a) Through "raw" SPARQL; and (b) using the IPSM REST service. The rationale for these choice is to, at first, guarantee the execution of the EWS functional tests through (a), which does not depend on any external component. By implementing (b) the solution is integrated with INTER-IoT framework and the results can be compared to the results from (a), demonstrating the pros/cons of each solution.
(2) A bridge MS Azure → INTER-MW, under development by the INTER-IoT consortium, is responsible to forward the data to INTER-MW and enable the automatic semantic

translation of the messages by using the IPSM. The EWS only subscribes to INTER-MW, asking to receive the data from the IoT Hub.

OBSERVATION: The second approach (2) is not ready (until the writing of this document). Developing, creating and maintaining an environment of INTER-MW with Azure IoT bridge are under INTER-IoT consortium responsibility. Therefore, the tests listed here, using option 2, will only be executed once the environment is ready.

At the application level, the input handler of the EWS, implemented as a flow in Node-Red server (INTER-IoT), is responsible to subscribe to the topics of INTER-MW that are related to the data submitted by our IoT solution, receiving these data translated from the middleware layer (see above). Then, this flow forwards the data to the EWS core processor, which implements the Context, Situation and Workflow managers by receiving the data through a REST API deployed in a separate server (either local or in the cloud). This REST API is a black box for INTER-IoT framework. When a situation is identified, the EWS core processor publishes the notifications, as EDXL messages, in a service bus. The output handler, implemented as a flow in INTER-IoT Node-RED, receives these notification messages, filter them and send to each specific target. The targets and their information requirements, i.e. the “parts” of the data (classes, object and data properties of the ontologies) that each target requires for each situation type (risk), are pre-defined. A simple web UI prototype demonstrates how to list and plot the notifications in a map. This UI is an example showing how the emergency centre can develop a web app that understands the EDXL messages and visualize the data.

3.3.5.2 Deliverables and version overview

The following table contains a deliverable list which need to be signed of before SAT testing commences.

ID	Description	Check
Documents		
1	Validation and Test report of the EWS	
Applications		
2	Shimmer3 API (TinyOS, ECG device)	
3	MyDriving-LD: porting of Shimmer Capture (Android app) to MyDriving, improved with semantic technologies.	
4	MyDriving App service (C# REST, app server)	
5	INTER-IoT-EWS input/output handlers (C# REST, app server)	
6	INTER-IoT-EWS core (C# REST, NESPER, app server(s))	
7	INTER-IoT-EWS UI Prototype for Tests Execution	
Hardware		
8	Shimmer3 ECG unit – should be returned to UT after the execution	
9	Android smartphone (Bluetooth) – should be returned to UT after the execution	
Subscriptions – all subscriptions will be cancelled after the tests execution		
10	Internet 3G/4G	
11	MS Azure: IoT Hub, app servers, DB servers	
12	AWS EC2: app servers, DB servers (optional)	

Table 41: Deliverable checklist

The following table shows the software components and version of which the system release version 1.0 consists of.

ID	Description	Version	Check
IoT Field Gateway (smartphone)			
1	Smart semantic gateway 1: Shimmer3 Capture Xamarin	V1.0.0	
2	Smart semantic gateway 2: MyDriving	V1.0.0	
IoT Cloud Gateway (context/service) broker			
3	MS Azure IoT Hub	Subsc.B1	
INTER-IoT framework			
4	Bridge MS Azure	TBD	
5	INTER-MW	TBD	
6	IPSM	V0.8.5	
IoT EWS (app layer)			
7	Input/output handlers: C# REST	V1.0.0	
8	EWS core processor: C# REST	V1.0.0	
9	INTER-IoT-EWS UI Prototype for Tests Execution	V1.0.0	

Table 42: Component version overview

3.3.5.3 Test environment

Introduction

To test the functionality of the integrated <File-Properties-T_PilotName> in combination with the IoT framework representative test hooks in the system are needed. This chapter will describe this environment and the used hardware, software, tools and platforms.

Test environment

This paragraph describes the test setups, hooks and probes used in the system used during this SAT.

In summary, the test environment reflects the deployment components illustrated in Figure 2, classified by the INTER-IoT framework layers:

- **Device:** at least 2 devices for each driver: a smartphone, acting as a field gateway (local broker), and an ECG device, the Shimmer3 ECG unit. The smartphone must be configured with 2 applications developed by us: Shimmer-LD (health data) and MyDriving-LD (logistics data).
- **IoT platform:** Azure IoT platform is used in two ways: (1) for health data, an IoT hub instance with at least the Azure subscription Basics 1 (B1); (2) for logistics data, an IoT hub instance – can be the same of #1, depending on the size and number of messages per day – and the basic services required to run MyDriving app, i.e. a mobile app service (app server) and a SQL Server database (DB server).
- **Middleware:** the INTER-MW component is required to subscribe to the IoT hub instance(s) to receive the messages through a bridge and to forward them to IPSM component. IPSM is responsible to execute the pre-defined translations (from SAREF to SSN/SOSA) and provides the data to the application layer.
- **Application:** the INTER-IoT-EWS system is composed by some subsystems:
    - **Input handler flow:** a Node-RED flow that subscribes to the INTER-IoT middleware mechanism (INTER-MW/IPSM), performing some validation on the input data and forwarding it to two REST APIs in parallel: Context Manager and Situation Identification Manager.

- **Context Manager:** a c#/.NET 4.5.2 core web API (app server) and a NOSQL database MongoDB (DB server). These services shouldn't be deployed in the same server.
- **Situation Identification Manager:** a c#/.NET 4.5.2 core web API (app server) running a CEP server (NESPER): http://www.espertech.com/esper/esper-faq/
- **Situation Reaction Manager:** this application is embedded in the Situation Identification Manager REST API, thus, deployed along with it. This component sends the emergency notification messages to a broker in the cloud. Ideally, it should publish in a way that INTER-MW could access. For testing purposes these messages will be published in an Azure IoT Hub instance that can be the same used in the IoT platform level, depending on the number/size of messages and throughput.
- **Output handler flows:** Node-RED flow(s) responsible to deliver a message to its target, e.g. by e-mail or SMS or a cloud broker. The number of flows depend on the configuration of each test case, in terms of targets and their information requirements.
- **Web UI prototype:** a very simple web UI (app server) that plays a role of a target, showing the emergency notifications in a map.

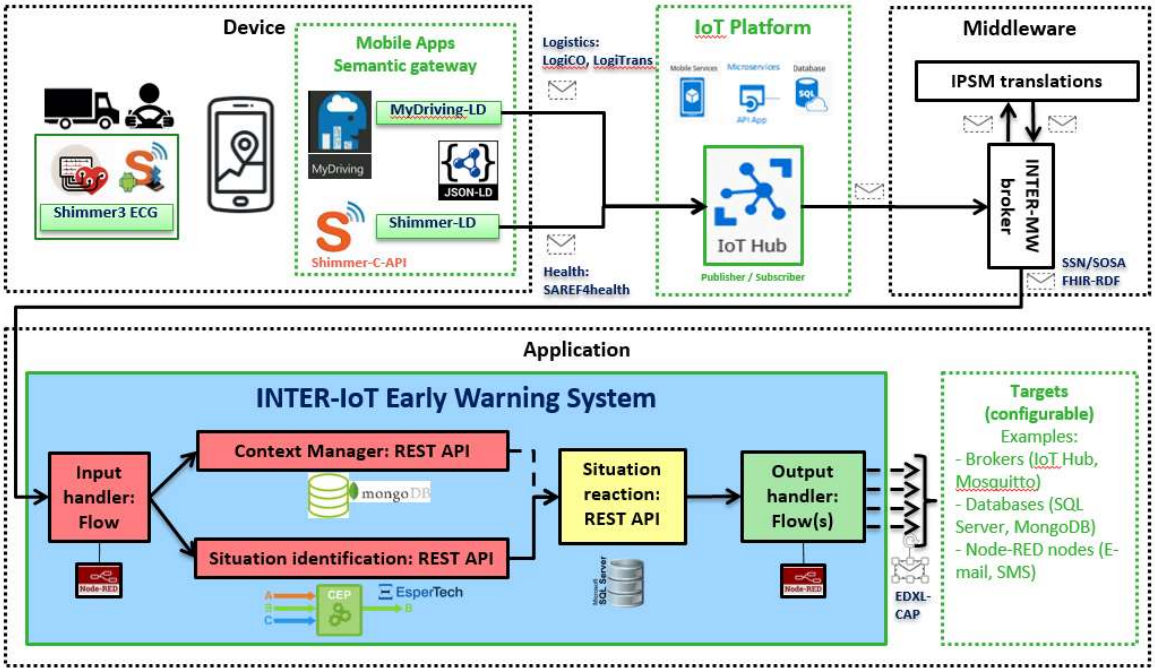


Figure 75: Deployment components or INTER-IoT-EWS data flow

Test tools, hooks and probes

This paragraph describes the test setups, tools, hooks and probes used in the by this document defined tests.

A test setup defines a setup used for a test, and describe the used system parts, interface etc.

Tools define the used COTS tooling that will be used during testing e.g. Logic analyzers, packet sniffer like e.g. Whireshark, developed scripts, etc.

Hooks are used to inject data or control parts of the system used to get the system to handle a needed scenario. This can be hidden, debug or service interfaces which contain the needed functionality.

Probes are used to gather logging from parts of the system under test to provide system feedback and to prove that the system is handling the scenario as it should.

The following paragraphs will define the used setups, tools, hooks and probes which will be referred to by the test descriptions.

The Figure 2 illustrates the data flow through the main components of INTER-IoT-EWS, summarizing the main technologies involved.

The Shimmer3 ECG unit is attached to the driver's chest (5 electrodes) and sends frequency-based data (usually 256 or 512Hz), using Shimmer3 API (TinyOS, default installed), to the smartphone, which requires the ShimmerLD application to receive and retransmit data. ShimmerLD is able to accumulate the data as time series, format it according to SAREF4health ontology and send messages in another frequency (e.g. each 5s) to the Azure IoT Hub, using either Internet 3/4G or Wi-Fi.

MyDrivingLD mobile app should also be installed in the smartphone. It is able to transform the trip (travel) information (e.g. position, acceleration) to Azure IoT Hub, in a frequency/position manner: e.g. at least at 3 seconds or 5m position shift. Data is formatted according to SAREF and logistics ontologies (LogiCO, LogiServ, LogiTrans). As default, MyDriving also communicates with an application service which stores trip data after the trip is ended (relational data). Optionally, the driver can use an OBD-2 sensor, which MyDriving app is able to capture more accurate data.

INTER-IoT-EWS receives the data from the Azure IoT Hub by subscribing to the INTER-IoT middleware, which is responsible to connect to IoT Hub through a bridge and translate the messages. Two translations (channels) are performed: from SAREF to SSN/SOSA and from SAREF4health to FHIR RDF. A node in INTER-IoT App level (Node-Red) will enable INTER-IoT-EWS to receive the translated messages through a flow. This flow forwards the data asynchronously (in parallel) to the Context Manager and to the Situation Identification Manager, both REST APIs able to receive IoT messages according to SSN/SOSA, health data as FHIR RDF and logistics data as the logistics ontologies.

The context manager stores the data as JSON document in a NOSQL database (MongoDB). These data is further collected by the Situation Reaction component, according to the information requirements of the targets.

The situation identification manager implements a Complex Event Processing (CEP) server, implemented with NEsper, which enables processing large volumes of incoming messages or events, filtering and analyzing events according to the Event Processing Language (EPL) statements, responding to conditions of interest (situation types) with minimal latency.

When a situation type is identified, e.g. vehicle collision or bradycardia, the situation identification manager calls the situation reaction component. For each situation type pre-defined, according to the 5 use cases, is mapped to a set of targets are notified. A default target is configured, to receive all the related (original) data of the trip through a broker. The situation reaction component is responsible to manage the brokering plan (situation types x reaction processes), transforming the output data in EDXL-CAP messages (JSON-LD, possible linked to the original data). These messages, including the instructions for brokering, are sent to a flow (Node-Red, INTER-IoT APP), which receives the data as a

REST API and performs the brokering actions, e.g. forward the data to a broker or send e-mails, SMS, etc.

TS_01 EWS basic deployment

- ECG device: 1 Shimmer3 ECG unit
- Smartphone: 1 Android with Bluetooth (Motorola)
    - Installed MyDrivingLD
- Cloud gateway (broker): 1 Azure IoT Hub (B1)
- MyDriving other basic services:
    - 1 mobile app server
    - 1 DB server (SQL Server), 1 DB instance
- Input/output handlers: REST API (C#/.NET): 1 app server
- Context manager:
    - REST API (C#/.NET): 1 app server
    - 1 DB server (MongoDB), 1 DB instance
- Situation identification and reaction:
    - REST API (C#/.NET) with NESPER (CEP): 1 app server

TS_02 EWS deployment with IPSM

Same of TS_01, but instead of executing the semantic translations inside the Input Handler component, it makes continuous calls to IPSM to translate the data stream published in the IoT Hub.

- INTER-IoT middleware: IPSM

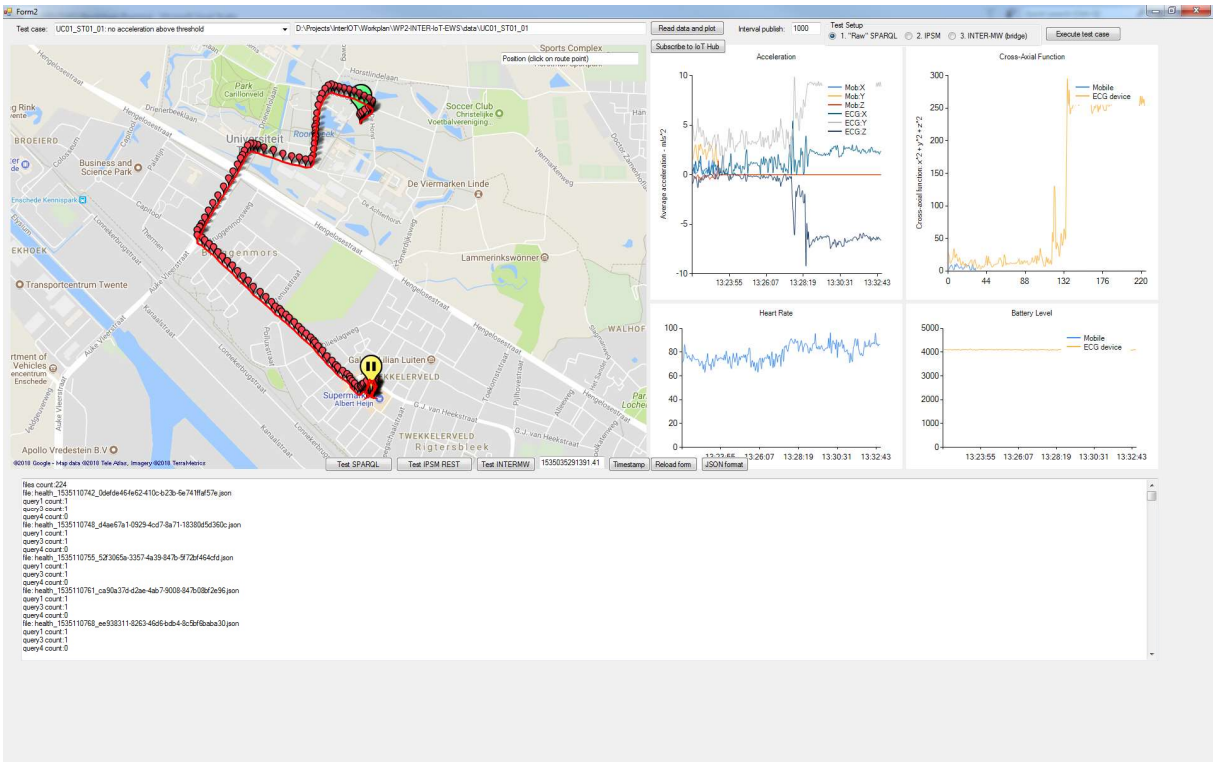
TS_03 EWS deployment with INTER-MW and IPSM

This setup relies on the MS Azure bridge, which is under construction by the INTER-IoT consortium. The EWS subscribes to INTER-MW instead of subscribing to IoT Hub and executing the semantic translations. The idea is that the EWS is decoupled from the specific IoT platform (Azure IoT). This setup, including the bridge, will be provided and managed by the INTER-IoT consortium.

- INTER-IoT middleware: bridge Azure, INTER-MW, IPSM

TT_01 INTER-IoT-EWS UI Prototype

This prototype was developed to support the execution of the tests. It can be either used for input data injection (see TH_01), as well as to subscribe to IoT Hub to simulate how the EWS consumes the data. In addition, this tool also receives the emergency notification messages that the EWS produces. Therefore, this tool is used for both FAT and SAT. This application should be running in debug mode (not deployed) with Visual Studio 2015, taking advantage of the testing tools provided by the IDE, such as the diagnostics tools for monitoring memory and CPU usage. Figure below illustrates the prototype:



TH_01 Input Data Injection

This hook is used to inject input data for all test cases for the laboratory tests. It simulates the smartphone publishing data each 3s to the IoT Hub, implemented in TT_01.

TH_02 Raw Data Visualizer

This hook is used to plot the “raw” data published in IoT Hub, useful to check the real-time data published. It is implemented in TT_01.

TH_03 Thresholds Modifier

This hook is used to change the thresholds to simulate the test cases that cover the use cases and is implemented through a REST service method. For UC01, threshold of collision detection can be modified to enable the simulation of vehicle collisions with hard breaks. For UC02, thresholds regarding bradycardia and tachycardia can be modified to enable the simulation of heart beat decrease and increase.

TH_04 Informing Dangerous Goods

This hook is used to inform the dangerous goods being transported by the truck, which is injected within the deployed MyDriving-LD mobile app.

TH_05 Situation Patterns

For more complex situation patterns, such as in UC02 to detect consecutive arrhythmia and in UC03 to detect arrhythmia after collision, patterns that are considered instances of a STs will be provided.

TH_06 Translation mechanism option

The difference between TS_01 and TS_02 is the way that the translations between SAREF and SSN/SOSA are executed, either through “raw” SPARQL or through IPSM. The TT_01 enables this option configuration at runtime.

TP_01 Log Semantic Translation

This probe is made of log commands in the solution to compute the processing time of the semantic translations. In TS_01 and TS_02, the log commands are added in the Input Handler: before and after each translation. In TS_03, the begin log command is added in a prototype subscribed to IoT Hub and the end log command is added in the Input Handler, measuring the total time processing of the bridge, INTER-MW and IPSM together. This probe supports the measuring of the “burden” that the semantic translation mechanism imposes.

TP_02 Log Syntactic Translation to CEP

This probe is made of log commands in the Situation Identification component (CEP engine server). The begin log command is added right after the semantic translation and the end log command is added right after the observations are sent to the CEP working memory.

TP_03 Log Situation Identification

This probe is made of log commands in the Situation Identification and Reaction components. The begin log command is added right after sending the observations to the CEP working memory. The end log command is added when the situation is identified, i.e. the beginning of the situation response.

TP_04 Log Situation Reaction (Decision Support)

This probe is made of log commands in the Situation Reaction component. The begin log command is added just after the situation is identified and the end log command is added after the EDXL message is generated.

3.3.5.4 Requirements, scenarios and use cases

The following table provides an overview of the relation between the requirements and the test(s) that validate their implementation. In bold the main requirements addressed with our solution.

ID	Description	Covered by
6	Efficiency of the information processing	
9	Multi-level data processing support	
13	Extensibility	
15	Support of common IoT communication protocols	All tests
20	Real time support	All tests
21	Real time output	
23	Device semantic definition	All tests
39	Heterogeneous gateway	
41	Definable and monitored requirements	
42	Heterogeneous information representation	T3.*
47	API for third-party developers	
51	API for data publication	
52	API REST	All tests
53	Location of sensor and measurement is included in semantic	All tests

	models	
62	Constraints on processing of personal and health data	T2.*, T3.*, T4.*
71	Application response time	
74	Ontology support	All tests
79	Service to manage energy consumption of devices	
86	API for proprietary systems interoperate with other systems	All tests
96	Enable (automated or semi-automated) linking of relevant data models	T3.* and T4.*
102	Exchanging complex medical measures across platforms	T2.*, T3.*, T4.*
104	Personal data and user profile management	
106	Definition of reference meaning for health information	
145	Informed consent. Processing of personal data	All tests
164	Medical Device informatics	T2.*, T3.*, T4.*
178	Inter Platform Semantic Mediator provides data and semantic interoperability functionality	All tests
179	Inter Platform Semantic Mediator supports platform communication	All tests
180	Syntactic and semantics interoperability - Data format and semantics translation	All tests
186	Design of required ontologies	All tests
220	Ontology mapping among most prominent standards	All tests
224	Location semantic support for mobile smart objects	All tests
251	Ability of IoT platforms to coordinate with emergency systems	All tests
281	Publish data stream into a platform	All tests

Table 43: Requirements vs. test mapping

The following table provides an overview of the relation between the scenarios and the test(s) that validate their implementation.

ID	Scenario name	Covered by
9	Accident at the port area	All tests

Table 44: Scenario vs test mapping

3.3.5.5 Test description

Scenario: accidents at the port area [id.9]

The functional goal of this scenario is to decrease the risk of fatal accidents at the port of Valencia, improving health prevention and enabling quick reaction by reducing time response.

The non-functional goal of this scenario is to exploit how e-Health and e-Care can use IoT platforms dedicated to logistics to prevent the occurrence of accidents and to support evacuation or attention in case of emergency situations [2]: “interoperate the wearable medical devices with IoT platforms to react quickly, thus reducing time responses during accidents and health prevention” [INTER-IoT deliverable 2.4].

Interoperability in this scenario is required to connect the port authority (including emergency systems) and the road hauliers IoT platforms.

Each use case will be tested using a vehicle from the port, which test cases will be simulated.

UC01: Vehicle collision detection

Monitor the truck's location and detect possible collisions (impacts). In general, the approaches use an accelerometer within the vehicle to collect time series data about its location, i.e. the device's acceleration about the corresponding axes (X, Y, Z), allowing the calculation of the G-Force felt in each instant. Then, for each instant, the detection mechanism compares if the G-Force is above a certain threshold, which is usually 3G for devices deployed in the vehicle chassis [16-19]. According to the patent for "vehicle security with accident notification and embedded driver analytics" (US 9491420 B2) [20], "instances of high acceleration/deceleration are due to a large change in velocity over a very short period of time. These speeds are hard to attain if a vehicle is not controlled by a human driver, which simplifies accident detection since we can assume any instance of high acceleration constitutes a collision involving human drivers". An approach using a smartphone sharing accelerometer data is described in [21]. The Shimmer ECG 3 also provides an accelerometer sensor, thus, it can also provide acceleration data, which gives an opportunity to integrate the health and logistics solutions.

For each instant, the sensor sends the data to the mobile application (Shimmer-LD and MyDriving-LD), which can compute the cross-axial energy function ($Etot = x^2 + y^2 + z^2$) and compare to a threshold.

Cross-axial energy function:

$Etot = x^2 + y^2 + z'^2$, where x,y,z are axial accelerations measured in m/s^2.

z' = *z* - 1G.

Threshold:

Common = 4G (4 x gravity), where 1G = 9.806 m/s^2.

Rule = If (SquaredRoot(Etot) > Threshold)

If it is above the threshold, then a potential fall is detected.

Classification of severity and urgency according to accelerometer data (A) and threshold (B) is described in the table below. In summary, if the cross-axial energy computed is greater than the threshold and less than 20% above the threshold, then it might be a light collision (minor severity). If it is in-between 20% and 40%, then the collision is greater (moderate severity), if it is in-between 40% and 60%, then the collision is severe. Above 60% represents a strong impact, thus, an extreme severity, which probably needs immediate urgency for emergency response.

Range	Severity	Urgency
B < A <= B * 1.2	Minor	Expected
B * 1.2 < A <= B * 1.4	Moderate	Immediate
B * 1.4 < A <= B * 1.6	Severe	Immediate
B * 1.6 < A	Extreme	Immediate

Each test case has an equivalent input and output data file, named TX.Y.json (input and output folders). The type of all test cases here are system testing using scripted data. Each test case is executed 5 times to reflect a normal operation situation and each range of severity/urgency (table above).

Each execution will be performed in TS_01, TS_02 and TS_03 (if the INTER-MW environment is available).

This use case involves these requirements: [23], [72], [180], [249], [251].

For vehicle collision the thresholds will be modified with TH_03 in a way to detect hard breaks of the vehicle, e.g. when driving inside the port at 50km/h the driver breaks to achieve 20km/h in 5 seconds.

Detected with ECG device accelerometer, computed by smartphone

ID	T1.1
Test	Vehicle collision with one accelerometer (from Shimmer3 ECG unit) computed by the mobile application (MyDriving-LD): test ST_UC01_01.
Type	System testing
Setup	TS_01, TS_02, TS_03*, TT_01, TH_03, TP_01-04
Start	Vehicle is stopped in one of the port gates (entering the port area). MyDriving-LD app is running and receiving streaming data from ECG device. Click on button “start trip” and wait the info menu appear on the top.
Req.	[180], [251]
Input	NA.
Output	T1.1_output/Execution_[TS_XX_YYMMDDhhmm]/IoTHubData/*.json: health and logistics messages T1.1_output/Execution_[TS_XX_YYMMDDhhmm]/Emergency/*.json: emergency messages
Logs	T1.1_output/Execution_[TS_XX_YYMMDDhhmm]
Outcome	Pass / Fail

Detected with ECG device accelerometer, computed by EWS (cloud)

ID	T1.2
Test	Vehicle collision with one accelerometer (from Shimmer3 ECG unit) computed by the EWS in the cloud (Situation Identification Manager): test ST_UC01_02.
Type	System testing
Setup	TS_01, TS_02, TS_03*, TT_01, TH_03, TP_01-04
Start	Vehicle is stopped in one of the port gates (entering the port area). MyDriving-LD app is running and receiving streaming data from ECG device. Click on button “start trip” and wait the info menu appear on the top.
Req.	[180], [251]
Input	NA.
Output	T1.2_output/Execution_[TS_XX_YYMMDDhhmm]/IoTHubData/*.json: health and logistics messages T1.2_output/Execution_[TS_XX_YYMMDDhhmm]/Emergency/*.json: emergency messages
Logs	T1.2_output/Execution_[TS_XX_YYMMDDhhmm]
Outcome	Pass / Fail

Detected with smartphone accelerometer, computed by smartphone

ID	T1.3
Test	Vehicle collision with one accelerometer (from smartphone) computed by the mobile application (MyDriving-LD): test ST_UC01_03.
Type	System testing
Setup	TS_01, TS_02, TS_03*, TT_01, TH_03, TP_01-04
Start	Vehicle is stopped in one of the port gates (entering the port area). MyDriving-LD app is running and receiving streaming data from ECG device. Click on button “start trip” and wait the info menu appear on the top.
Req.	[180], [249], [251]
Input	NA.
Output	T1.3_output/Execution_[TS_XX_YYMMDDhhmm]/IoTHubData/*.json: health and logistics messages T1.3_output/Execution_[TS_XX_YYMMDDhhmm]/Emergency/*.json: emergency messages
Logs	T1.3_output/Execution_[TS_XX_YYMMDDhhmm]
Outcome	Pass / Fail

Detected with smartphone accelerometer, computed by EWS (cloud)

ID	T1.4
Test	Vehicle collision with one accelerometer (from smartphone) computed by the EWS in the cloud (Situation Identification Manager): test ST_UC01_04.
Type	System testing
Setup	TS_01, TS_02, TS_03*, TT_01, TH_03, TP_01-04
Start	Vehicle is stopped in one of the port gates (entering the port area). MyDriving-LD app is running and receiving streaming data from ECG device. Click on button “start trip” and wait the info menu appear on the top.
Req.	[180], [249], [251]
Input	NA.
Output	T1.4_output/Execution_[TS_XX_YYMMDDhhmm]/IoTHubData/*.json: health and logistics messages T1.4_output/Execution_[TS_XX_YYMMDDhhmm]/Emergency/*.json: emergency messages
Logs	T1.4_output/Execution_[TS_XX_YYMMDDhhmm]
Outcome	Pass / Fail

Detected with ECG device and smartphone accelerometers, computed by EWS (cloud)

ID	T1.5
Test	Vehicle collision with two accelerometers (from Shimmer3 ECG unit and smartphone) computed by the mobile application (MyDriving-LD): test ST_UC01_05.
Type	System testing
Setup	TS_01, TS_02, TS_03*, TT_01, TH_03, TP_01-04
Start	Vehicle is stopped in one of the port gates (entering the port area). MyDriving-LD app is running and receiving streaming data from ECG device. Click on button “start trip” and wait the info menu appear on the top.
Req.	[180], [251]
Input	NA.
Output	T1.5_output/Execution_[TS_XX_YYMMDDhhmm]/IoTHubData/*.json: health and logistics messages T1.5_output/Execution_[TS_XX_YYMMDDhhmm]/Emergency/*.json: emergency messages
Logs	T1.5_output/Execution_[TS_XX_YYMMDDhhmm]
Outcome	Pass / Fail

UC02: Hazardous health changes

Detect medical issues with the driver by monitoring his/her ECG and derived heart rate, checking possible cardiovascular emergencies. Cardiovascular emergencies are life-threatening disorders that must be recognized as soon as possible to minimize morbidity and mortality. By allowing the EWS to detect cardiovascular emergencies with trucks’ drivers, it is possible to reduce the risk of an accident at the port area. The EWS provides messages that include the information of the cardiovascular emergency situation.

This can be achieved, basically, by using the the INTER-Health IoT solution with Shimmer ECG device attached to the driver’s chest, wired to electrodes, and an Android-based mobile phone, both part of the Body module of the BodyCloud approach implemented with the SPINE framework. Thresholds used by the detection mechanism should be based on existing classifications to detect health risks. For example, target heart rates used for fitness is a classification of indicators that can be used as a baseline for thresholds. Figure XX illustrates such indicators (red, green, yellow, blue) depending on the person’s age. Besides these thresholds, this use case also considers the situations which the driver presents bradycardia and tachycardia, which can be detected with the ECG device (event monitor) [18].

Thresholds used by the detection mechanism should be based on existing classifications to detect health risks, both regarding bradycardia and tachycardia[19]. In particular, the approach based on Early Warning Scores [Ref.] should be considered. These scores are illustrated in table XX, which represents the rules underlying each vital sign scoring function. For example,

[18] http://www.mayoclinic.org/diseases-conditions/bradycardia/diagnosis-treatment/diagnosis/dxc-20321665
http://www.mayoclinic.org/diseases-conditions/tachycardia/diagnosis-treatment/diagnosis/dxc-20253919
[19]http://www.mayoclinic.org/diseases-conditions/bradycardia/diagnosis-treatment/diagnosis/dxc-20321665
http://www.mayoclinic.org/diseases-conditions/tachycardia/diagnosis-treatment/diagnosis/dxc-20253919

if the person presents a heart rate of 115bpm, the score for this vital sign is 2. The sum of the scores represent the person's health situation. For example, the total score of more than 5 is statistically linked to the person requiring an intensive care or near-death situation.

Score	3	2	1	0	1	2	3
Respiratory rate (breaths/min)	>35	31-35	21-30	9-20			<7
SpO2 (%)	<85	85-89	90-92	>92			
Temperature (C)		>38.9	38-38.9	36-37.9	35-35.9	34-34.9	<34
Systolic BP (mmHg)		>199		100-199	80-99	70-79	<70
Heart rate (bpm)	>129	110-129	100-109	50-99	40-49	30-39	<30
AVPU				Alert	Verbal	Pain	Unresponsive

Classification of severity and urgency according to ComputeBPM output (A) and the threshold (B) is described in the table below. In summary, if the BPM calculated is greater than the threshold and less than threshold more 10%, then it might be a light tachycardia (minor severity). If it is in-between 10% and 20%, then the tachycardia is greater (moderate severity), if it is in-between 20% and 30%, then the tachycardia is severe. Greater than 30% represents a strong tachycardia, thus, an extreme severity, which probably needs immediate urgency for emergency response.

Range Tachycardia	Range Bradycardia	Severity	Urgency
B < A <= B * 1.1	B * 0.9 < A <= B	Minor	Expected
B * 1.1 < A <= B * 1.2	B * 0.8 < A <= B * 0.9	Moderate	Immediate
B * 1.2 < A <= B * 1.3	B * 0.7 < A <= B * 0.8	Severe	Immediate
B * 1.3 < A	A <= B * 0.7	Extreme	Immediate

Each test case has an equivalent input and output data file, named TX.Y.json (input and output folders). The type of all test cases here are system testing using scripted data. Each test case is executed 5 times to reflect a normal operation situation and each range of severity/urgency (table above).

Each execution will be performed in TS_01, TS_02 and TS_03 (if the INTER-MW environment is available).

This use case involves these requirements: [23], [72], [180], [249], [251].

For detecting heart arrhythmias and heart attacks (stop beating), simulated data will be inserted in the application to be used at specific pre-defined timestamps while driving inside the port.

Bradycardia detected with fixed threshold

ID	T2.1
Test	From ECG data, the heart rate is calculated and compared to a threshold.
Type	System testing
Setup	TS_01, TS_02, TS_03*, TT_01, TH_03, TP_01-04
Start	Vehicle is stopped in one of the port gates (entering the port area). MyDriving-LD app is running and receiving streaming data from ECG device. Click on button "start trip" and wait the info menu appear on the top.
Req.	[180], [249], [251]
Input	NA.

Output	T2.1_output/Execution_[TS_XX_YYMMDDhhmm]/IoTHubData/*.json: health and logistics messages T2.1_output/Execution_[TS_XX_YYMMDDhhmm]/Emergency/*.json: emergency messages
Logs	T2.1_output/Execution_[TS_XX_YYMMDDhhmm]
Outcome	Pass / Fail

Tachycardia detected with fixed threshold

ID	T2.2
Test	From ECG data, the heart rate is calculated and compared to a threshold.
Type	System testing
Setup	TS_01, TS_02, TS_03*, TT_01, TH_03, TP_01-04
Start	Vehicle is stopped in one of the port gates (entering the port area). MyDriving-LD app is running and receiving streaming data from ECG device. Click on button “start trip” and wait the info menu appear on the top.
Req.	[180], [251]
Input	NA.
Output	T2.2_output/Execution_[TS_XX_YYMMDDhhmm]/IoTHubData/*.json: health and logistics messages T2.2_output/Execution_[TS_XX_YYMMDDhhmm]/Emergency/*.json: emergency messages
Logs	T2.2_output/Execution_[TS_XX_YYMMDDhhmm]
Outcome	Pass / Fail

Multiple occurrences of bradycardia detected with fixed threshold

ID	T2.3
Test	From ECG data, the heart rate is calculated and compared to a threshold.
Type	System testing using scripted data
Setup	TS_01, TS_02, TS_03*, TT_01, TH_03, TP_01-04
Start	Vehicle is stopped in one of the port gates (entering the port area). MyDriving-LD app is running and receiving streaming data from ECG device. Click on button “start trip” and wait the info menu appear on the top.
Req.	[180], [249], [251]
Input	NA.
Output	T2.3_output/Execution_[TS_XX_YYMMDDhhmm]/IoTHubData/*.json: health and logistics messages T2.3_output/Execution_[TS_XX_YYMMDDhhmm]/Emergency/*.json: emergency messages
Logs	T2.3_output/Execution_[TS_XX_YYMMDDhhmm]
Outcome	Pass / Fail

Multiple occurrences of tachycardia detected with fixed threshold

ID	T2.4
Test	From ECG data, the heart rate is calculated and compared to a threshold.
Type	System testing
Setup	TS_01, TS_02, TS_03*, TT_01, TH_03, TP_01-04
Start	Vehicle is stopped in one of the port gates (entering the port area). MyDriving-LD app is running and receiving streaming data from ECG device. Click on button “start trip” and wait the info menu appear on the top.
Req.	[180], [249], [251]
Input	NA.
Output	T2.4_output/Execution_[TS_XX_YYMMDDhhmm]/IoTHubData/*.json: health and logistics messages T2.4_output/Execution_[TS_XX_YYMMDDhhmm]/Emergency/*.json: emergency messages
Logs	T2.4_output/Execution_[TS_XX_YYMMDDhhmm]
Outcome	Pass / Fail

UC03: Temporal relations (UC01 ~ UC02)

This use case exploits the possible temporal relations between UC01 and UC02 for detection of an accidents in the port area. For example, if a truck collision is detected from the accelerometers of the medical and mobile devices (T1.3) and right after (e.g. within 1-2 minutes) detecting large variation of heart rate (T2.9) then there is a high probability that a severe accident occurred, the driver is injured and he/she requires urgent medical help. Notice that the temporal relationship (“right after”) is crucial to integrate these use cases.

For the temporal relations, a mix of the both above is planned, for example, simulating that right after (within 30 seconds of) a hard break, simulated data representing a bradycardia is used by the mobile app.

Vehicle collision followed by bradycardia

ID	T3.1
Test	Slow heart rate right after (within 2 minutes) a collision is detected can represent that an accident just occurred and the driver is probably injured.
Type	System testing
Setup	TS_01, TS_02, TS_03*, TT_01, TH_03, TP_01-04
Start	Vehicle is stopped in one of the port gates (entering the port area). MyDriving-LD app is running and receiving streaming data from ECG device. Click on button “start trip” and wait the info menu appear on the top.
Req.	[180], [249], [251]
Input	NA.
Output	T3.1_output/Execution_[TS_XX_YYMMDDhhmm]/IoTHubData/*.json: health and logistics messages T3.1_output/Execution_[TS_XX_YYMMDDhhmm]/Emergency/*.json: emergency messages

Logs	T3.1_output/Execution_[TS_XX_YYMMDDhhmm]
Outcome	Pass / Fail

Bradycardia followed by vehicle collision

ID	T3.2
Test	Slow heart rate right before (within 2 minutes) a collision is detected can represent that an accident occurred because the driver had a cardiac issue.
Type	System testing
Setup	TS_01, TS_02, TS_03*, TT_01, TH_03, TP_01-04
Start	Vehicle is stopped in one of the port gates (entering the port area). MyDriving-LD app is running and receiving streaming data from ECG device. Click on button “start trip” and wait the info menu appear on the top.
Req.	[180], [249], [251]
Input	NA.
Output	T3.2_output/Execution_[TS_XX_YYMMDDhhmm]/IoTHubData/*.json: health and logistics messages T3.2_output/Execution_[TS_XX_YYMMDDhhmm]/Emergency/*.json: emergency messages
Logs	T3.2_output/Execution_[TS_XX_YYMMDDhhmm]
Outcome	Pass / Fail

UC04: Accident involving dangerous goods

This use case will extend the use cases UC01-03 by checking whether dangerous goods are being transported, which will increase the situation urgency and severity and include the dangerous goods classification according to UNECE[20]. Data test will include simulation of trips including the transportation of class 1 (explosives), 3 (flammable liquids), 4 (flamed solids), 6 (toxic and infectious) and 7 (radioactive).

Hard coding the goods being transported with TH_04, re-executing each previews simulation (UC01-UC03).

UC01 with dangerous goods

ID	T4.1
Test	Tests of UC01 incremented with a check whether dangerous goods are being transported.
Type	System testing
Setup	TS_01, TS_02, TS_03*, TT_01, TH_03, TP_01-04
Start	Vehicle is stopped in one of the port gates (entering the port area). MyDriving-LD app is running and receiving streaming data from ECG device. Click on button “start trip” and wait the info menu appear on the top.
Req.	[180], [249], [251]
Input	NA.

[20] https://www.unece.org/fileadmin/DAM/trans/danger/publi/unrec/rev17/English/Rev17_Volume1.pdf

Output	T4.1_output/Execution_[TS_XX_YYMMDDhhmm]/IoTHubData/*.json: health and logistics messages T4.1_output/Execution_[TS_XX_YYMMDDhhmm]/Emergency/*.json: emergency messages
Logs	T4.1_output/Execution_[TS_XX_YYMMDDhhmm]
Outcome	Pass / Fail

UC02 with dangerous goods

ID	**T4.2**
Test	Tests of UC02 incremented with a check whether dangerous goods are being transported.
Type	System testing
Setup	TS_01, TS_02, TS_03*, TT_01, TH_03, TP_01-04
Start	Vehicle is stopped in one of the port gates (entering the port area). MyDriving-LD app is running and receiving streaming data from ECG device. Click on button “start trip” and wait the info menu appear on the top.
Req.	[180], [249], [251]
Input	NA.
Output	T4.2_output/Execution_[TS_XX_YYMMDDhhmm]/IoTHubData/*.json: health and logistics messages T4.2_output/Execution_[TS_XX_YYMMDDhhmm]/Emergency/*.json: emergency messages
Logs	T4.2_output/Execution_[TS_XX_YYMMDDhhmm]
Outcome	Pass / Fail

UC03 with dangerous goods

ID	**T4.3**
Test	Tests of UC03 incremented with a check whether dangerous goods are being transported.
Type	System testing
Setup	TS_01, TS_02, TS_03*, TT_01, TH_03, TP_01-04
Start	Vehicle is stopped in one of the port gates (entering the port area). MyDriving-LD app is running and receiving streaming data from ECG device. Click on button “start trip” and wait the info menu appear on the top.
Req.	[180], [249], [251]
Input	NA.
Output	T4.3_output/Execution_[TS_XX_YYMMDDhhmm]/IoTHubData/*.json: health and logistics messages T4.3_output/Execution_[TS_XX_YYMMDDhhmm]/Emergency/*.json: emergency messages
Logs	T4.3_output/Execution_[TS_XX_YYMMDDhhmm]
Outcome	Pass / Fail

3.3.5.6 Test outcome

The following table will provide an overview of the test result of all the performed tests in this SAT.

Test	Description	Outcome
T1.1	Vehicle collision with one accelerometer (from Shimmer3 ECG unit) computed by the mobile application (MyDriving-LD): test ST_UC01_01	Pass / Fail
T1.2	Vehicle collision with one accelerometer (from Shimmer3 ECG unit) computed by the EWS in the cloud (Situation Identification Manager): test ST_UC01_02	Pass / Fail
T1.3	Vehicle collision with one accelerometer (from smartphone) computed by the mobile application (MyDriving-LD): test ST_UC01_03	Pass / Fail
T1.4	Vehicle collision with one accelerometer (from smartphone) computed by the EWS in the cloud (Situation Identification Manager): test ST_UC01_04	Pass / Fail
T1.5	Vehicle collision with two accelerometers (from Shimmer3 ECG unit and smartphone) computed by the mobile application (MyDriving-LD): test ST_UC01_05	Pass / Fail
T2.1	From ECG data, the heart rate is calculated and compared to a threshold	Pass / Fail
T2.2	From ECG data, the heart rate is calculated and compared to a threshold	Pass / Fail
T2.3	From ECG data, the heart rate is calculated and compared to a threshold	Pass / Fail
T2.4	From ECG data, the heart rate is calculated and compared to a threshold	Pass / Fail
T3.1	Slow heart rate right after (within 2 minutes) a collision is detected can represent that an accident just occurred and the driver is probably injured	Pass / Fail
T3.2	Slow heart rate right before (within 2 minutes) a collision is detected can represent that an accident occurred because the driver had a cardiac issue	Pass / Fail
T4.1	Tests of UC01 incremented with a check whether dangerous goods are being transported	Pass / Fail
T4.2	Tests of UC02 incremented with a check whether dangerous goods are being transported	Pass / Fail
T4.3	Tests of UC03 incremented with a check whether dangerous goods are being transported	Pass / Fail
SAT Outcome		**Pass / Fail**

Table 45: Test outcome overview

3.3.5.7 Integration ethics and security

Introduction

This section addresses recommendations and proposed solutions to the ethical and security points of the integration.

Early Warning System

Once our collaboration offers the monitoring of ECG data, there is an extreme concern regarding the management of sensitive/private end user data and ethical issues. Our platform follows the guidelines of “Microsoft Azure and data compliance for designing secure health solutions”. Azure services are compliant to several industry applications, including HIPPA compliance for healthcare solutions, the EU Data Protection Directive 95/46/EC (the privacy standard for processing personal data from EU member states) as well as the recent 2016/679 directive.

Key principles are implemented in our solution, such as identity and authentication, role-based access control, certificate acquisition and management and user data encryption when required. Complete control of the data is achieved with the support of an information security management system, configuring a secure IoT Hub and selecting the region that best meets compliance needs, as the Germany or the Netherlands or North European sites. These principles are aligned to the H2020 ethical self-assessment following the specific rules about cardiac data.

As a foundation for the use of our IoT platform, our research provides support to apply the FAIR data principles for data management of the project development and the data stewardship of the solution. We emphasize the “A” aspect of FAIR to deal with the sensitive data management issue, applying authentication and authorization procedures from the device to the network, middleware and service levels. Our solution uses data and metadata retrievable by their identifier, which are based on open standardized communication protocol. Finally, we have experience with FAIR-based data management plans, as the H2020 template, and we plan to store the project data in the INTER-IoT data repository, which is aligned to H2020 and, therefore, with FAIR. In particular, we will follow the data management plan of INTER-IoT (http://www.INTER-IoT-project.eu/wp-content/uploads/2016/02/D8.4_INTER-IoT_Data_Management_Plan.pdf), giving emphasis to the classification of sensitive data according to the table of protection levels (section 2.3. Other data), classifying ECG data as “Protection level 2”.

In preparing, collecting and analyzing the sensitive data, we will comply with ethical principles and international, European and national regulations including the Code of Ethics for Research that was developed by the Association of Universities in the Netherlands (VSNU). The code prescribes that the data will only be used for research purposes and individual persons may never be traceable in publications reporting on the research. Drivers participating in the pilot will be provided with an Informed Consent (by signing a Personal Consent Form) before involving them. In preparing our data collection using individual persons, we will always ask for consent of participation with the guarantee that participants are always free to leave the investigation without consequences or need to give any explanation. Considering that the research will involve people from Spain, the Personal Consent Form will be written in English and Spanish, provided by the port Authority. The researchers have already considered issues related to informed consent including: ethics sensitivity, issues of insurance and incidental findings.

The basic content of the Personal Consent Form will consist in:

• describing the aims, methods and implications of the research, the nature of the participation and any benefits, risks or discomfort that might ensue

• informing the participants about their rights

• state how data will be collected, protected during the project and either destroyed or reused subsequently

• state what procedures will be implemented in the event of unexpected or incidental findings (in particular, whether the participants have the right to know, or not to know, about any such findings).

3.3.6 Third Party: Senshook

The port of Valencia is one of the most important hubs in the world and thus a critical point of entry of invasive species that must be monitored, according to the European Centre of Disease Control.

The pilot will consist on deploying an observation static IoT node in a critical point of the port of Valencia.

The node is composed of a Smart Mosquito Trap capable of mimicking the human body (scent and respiration) and of automatically counting captured mosquitoes, identify the gender and the species. The information collected by each node is then sent to a server.

The pilot will start with preliminary tests in the zoo of Barcelona in May-June 2018 and will last until October 2018. This corresponds exactly with the period of the year when disease-vector mosquitoes are active and must be monitored.

Following is a diagram that gives a high-level overview of the system. When a mosquito enters the trap it gets detected by the sensor.

The sensor is connected to a Senscape board which sends the information to the server running SensHook and the virtual gateway.

The middleware platform can then retrieve the gathered information.

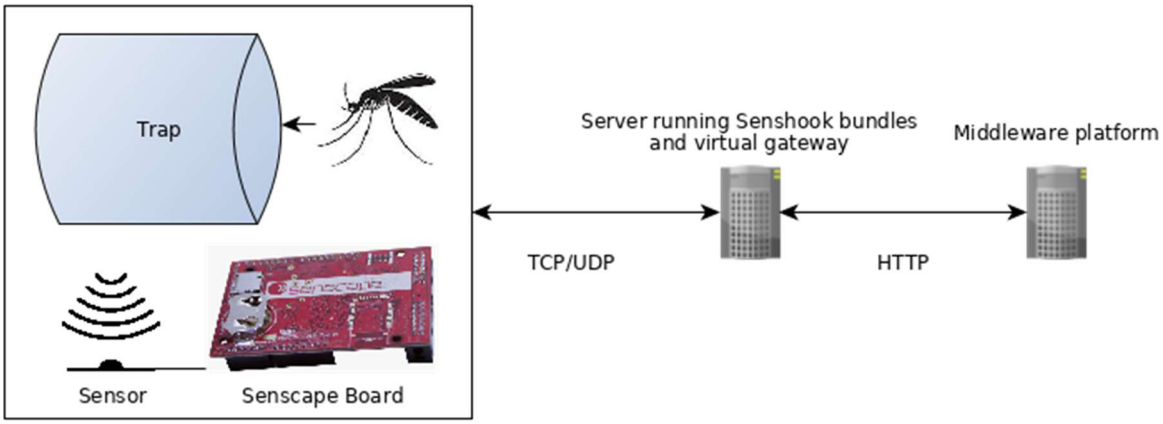


Figure 76: Network Architecture

Objectives of the project

The specific objectives of the project are to:

- Perform a technical feasibility assessment of the SensHook solutions as part of the INTER-IoT project
- Implement SensHook according to INTER-IoT requirements
- Carry out a series of tests/pilots to evaluate the performance and benefits of the tool.

Collaboration approach

Irideon will contribute to the INTER-IoT project by providing a new open tool for the INTER-LAYER building block, which will allow the evolution of products based on INTER-IoT, but at the same time will allow us to evolve our products in order to add new interoperability features.

By contributing to the development of INTER-IoT, Irideon will be able to address new IoT scenarios in which different IoT platforms, apart from those based on Senscape, are involved, and also in those in which more than one application domain is addressed.

SensHook Arquitecture Overview

The system consists of the SensHook software that integrates with the INTER-IoT virtual gateway which provides connection to the middleware platform.

Below you can see an overview of the architecture:

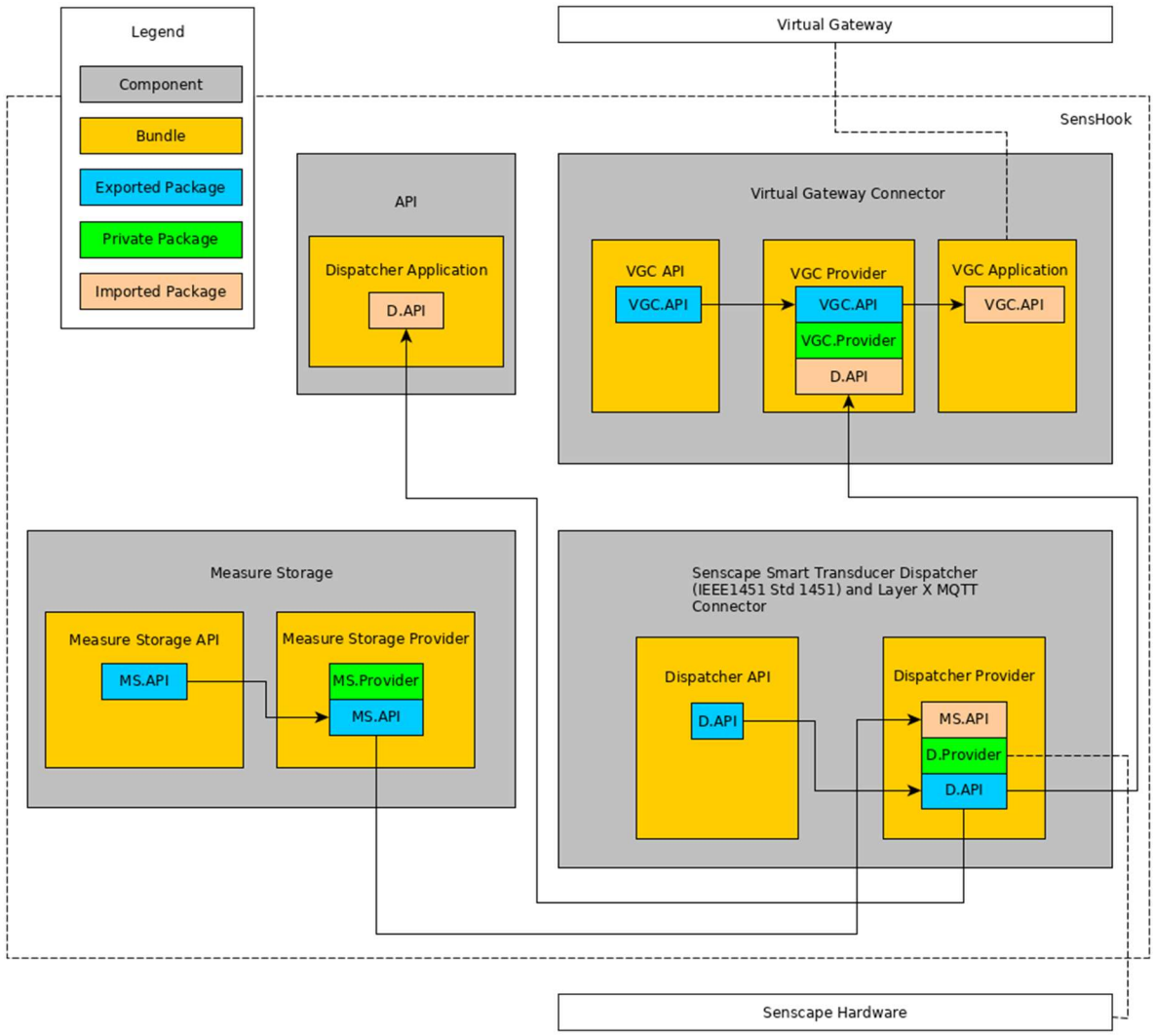


Figure 77: System architecture

Following is a description of the different components of the gateway.

Dispatcher

One of the central parts of SensHook is the dispatcher, it consists of a communication layer which connects to the hardware via TCP/UDP and a service layer which implements the IEEE 1451 standard.

IEEE 1451 is a set of smart transducer interface standards developed by the IIEEE.

Irideon has developed a specific lightweight implementation of this standard for the Senscape hardware devices.

This implementation includes communication protocols, TEDS and common functions.

The dispatcher is implemented in two OSGi bundles:

- Dispatcher API bundle: This bundle defines and exports the interface.
- Dispatcher provider bundle: This bundle implements the service layer, which includes the IEEE 1451 standard and the communication layer with the MQTT connector. Furthermore it registers a service with the dispatcher API.

Virtual Gateway Connector

This is the other central part of SensHook. It processes the petitions of the virtual gateway coming from the middleware platform. The virtual gateway connector consists of the three bundles:

- Virtual gateway connector API bundle: This bundle defines the OSGi interface for the bundle and exports it as a package.
- Virtual gateway connector provider bundle: This bundle implements the virtual gateway connector and provides it as an OSGi service with the virtual gateway connector API.
- Virtual gateway connector application bundle: This bundle consumes the service published by the provider and implements an application that exposes an API. This API is used to interface with the virtual gateway.

Measure Storage

The measure storage has two bundles:

- Measure storage API Bundle: This bundle defines the interface for the bundle and exports it as a package.
- Measure storage provider bundle: This bundle provides the connection to a local database to store and retrieve the values sent and requested by the dispatcher. This bundle implements the interface defined by the API bundle and registers it as a service.

API

This component provides an API to the dispatcher which serves as an interface to interact with the Senscape hardware.

It consists of one bundle which is an OSGi application that consumes the service offered by the dispatcher.

System interfaces

SensHook offers interfaces to two of its components, the virtual gateway connector and the dispatcher.

Dispatcher

The dispatcher comes with two interfaces:

- Terminal interface: This interface features an Apache Felix Gogo Shell. It is thought for debugging.
- OSGi interface: The component registers an OSGi service API which can be used to incorporate the dispatcher in a modular OSGi application.

Virtual Gateway Connector

The virtual gateway connector comes with three interfaces:

- Terminal interface: This interface features a gogo shell. It is thought for debugging.
- OSGi interface: The component registers an OSGi service API which can be used to incorporate the virtual gateway connector in a modular OSGi application.
- API: The virtual gateway connector also offers an API through an OSGi application following the OSGi enRoute model, so it is possible to communicate with the component and having a runtime environment.

3.3.6.1 Integration of IoT framework

SensHook integrates with the IoT framework through the INTER-IoT virtual gateway. SensHook comes with a component called virtual gateway connector that offers an API for the incoming petitions from the middleware platform through the virtual gateway. On the other hand the virtual gateway connector uses methods of the virtual gateway to send information to the middleware platform.

API of the virtual gateway connector

This API is used by the INTER-IoT virtual gateway to send petitions to SensHook.

Method: sendDeviceRequestMessage

This method is used to send request information about devices from SensHook. It receives a string parameter which defines the type of request. These types and their functions are detailed in the following table:

Function	Parameters	Description
List devices	String “list-devices”	This request returns a list of devices registered in SensHook.
Get information about a device.	String “get-device”, Int deviceId	This request returns information about the device (trap or sensor) with ID deviceID.
Read data from device	String “read-device”, Int deviceId	This request reads data from the device with ID deviceID.

Methods of the virtual gateway

In the following table, we describe the methods of virtual gateway used by the virtual gateway connector.

Method	Parameters	Description	Example
new Device()	-	Constructor for a new device object. In this case a device is a mosquito trap.	Device trap_01 = new Device();
Device.setUuid()	String containing UUID of the device	This sets the UUID of the device.	trap_01.setUuid("01");
Device.setName	String	This sets the name	Trap_01.setName("SmartTrap_

()	containing name of the device	of the device.	02");
new DeviceIO()	device type, device name, device data type	Constructor for a new IO device. This can be a sensor or an actuator.	DeviceIO tempSensor = new DeviceIO(DeviceIO.Type.*SENSOR*, "temperature", Attribute.Type.*FLOAT*);
Device.addDeviceIO()	An instance of DeviceIO	This method adds an IO device to a device. In this case a sensor to a mosquito trap.	trap_01.addDeviceIO(tempSensor)
New Measurement()	-	Constructor for a new measurement object. This will be used to send data from sensors.	Measurement measurement = new Measurement();
Measurement.setValue();	Name of the sensor, data type, data value	Sets the value of a mesurement.	measurement.setValue(new Attribute("temperature", Attribute.Type.*FLOAT*), 14.5);

Testing

As described in chapter 7 and 8, we use Postman and some HTTP requests to the middleware platform to do some integration testing.

In the following screenshot you can see an example of the request “Get all entities” to the middleware platform.

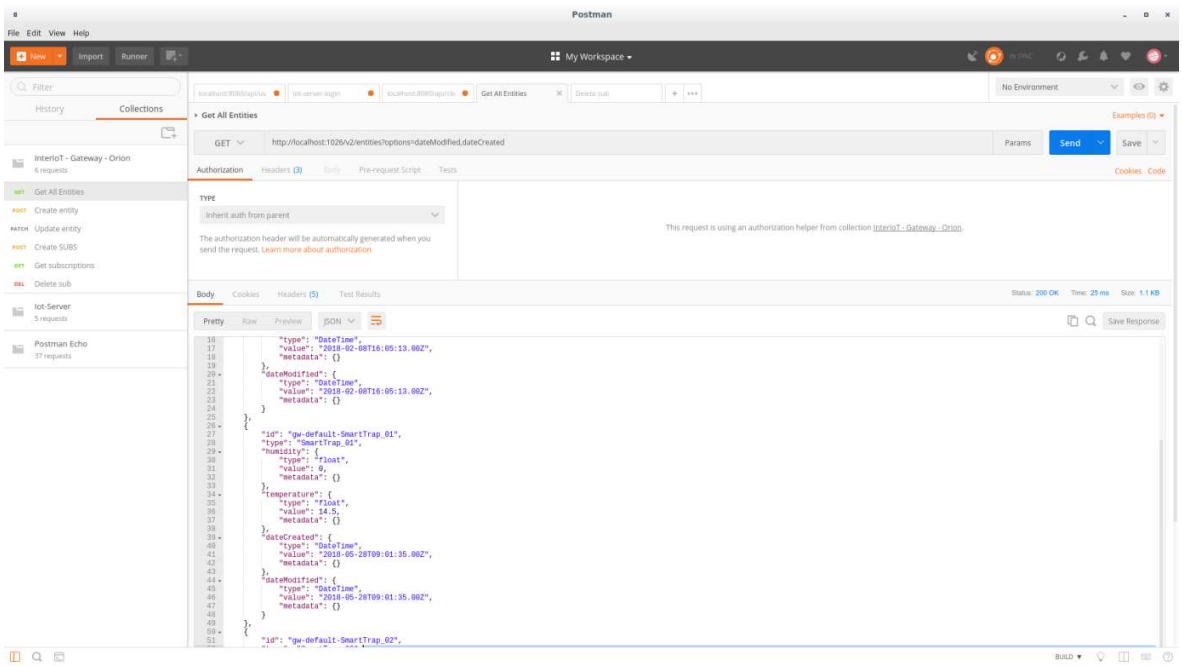


Figure 78: Integration test

3.3.6.2 Deliverables and version overview

The following table contains a deliverable list which need to be signed of before SAT testing commences.

ID	Description	Check
Documents		
1	SAT Document	
Hardware		
2	Senscape Mosquito Trap	
3	Computer running the Senshook bundles, INTER-IoT VGW, Docker with middleware, InfluxDB	
Tools		
4	Eclipse	
5	JUnit	
6	Postman	
7	InfluxDB	

Table 46: Deliverable checklist

The following table shows the software components and version of which the system release version 1.0 consists of.

ID	Description	Version	Check
IoT Virtual Gateway			
1	Dispatcher API Bundle	V1.0.0	
2	Dispatcher Provider Bundle	V1.0.0	
3	Dispatcher Application Bundle	V1.0.0	
4	Measure Storage API Bundle	V1.0.0	
5	Measure Storage Provider Bundle	V1.0.0	
6	Virtual Gateway Connector API Bundle	V1.0.0	
7	Virtual Gateway Connector Provider Bundle	V1.0.0	
8	Virtual Gateway Connector Application Bundle	V1.0.0	

Table 47: Component version overview

3.3.6.3 Requirements, scenarios and use cases

The following table provides an overview of the relation between the requirements and the test(s) that validate their implementation.

ID	Description	Covered by
11	Addressability and reachability	T15.1.1, T15.1.4
15	Common IoT communication protocols must be supported.	T15.1.5, T15.2.9
47	API for third-party developers	15.2.7., 15.2.8
57	Device monitoring and self-awareness of the system	T15.1.6, T15.2.1, T15.2.2, T15.2.3, T15.2.4, T15.2.5, T15.2.6, T15.2.7, T15.2.8,

		T15.2.10
284	Standard protocol for the device communications	T15.1.1
110	Usability	T15.1.1, T15.1.2, T15.1.3, T15.1.4, T15.1.5, T15.2.1, T15.2.2, T15.2.3, T15.2.4, T15.2.5, T15.2.6, T15.2.7, T15.2.8, T15.2.9
123	Use of standards	T15.1.1
154	Time stamped event recording	T15.1.6, T15.2.10
265	API allows device declaration and configuration	15.1.2, 15.1.3, 15.1.4
266	API allows resources/capabilities discovery	T15.2.1, T15.2.2, T15.2.3, T15.2.4, T15.2.5, T15.2.6, T15.2.7, T15.2.8
273	Stores system status for recovery	T15.2.1, T15.2.2, T15.2.3, T15.2.4, T15.1.6, T15.2.5, T15.2.6, T15.2.7, T15.2.8, T15.2.10
283	Manage a sensor or actuator	T15.1.1, T15.1.2, T15.1.3, T15.1.4, T15.2.1, T15.2.2, T15.2.3, T15.2.4, T15.2.5, T15.2.6, T15.2.7, T15.2.8

Table 48: Requirements vs test mapping

The following table provides an overview of the relation between the scenarios and the test(s) that validate their implementation.

ID	Scenario name	Covered by
15	Surveillance systems for prevention programs	T15.2.7, 15.2.8, 15.2.9

Table 49: Scenario vs test mapping

3.3.6.4 Test environment

Introduction

To test the functionality of the integrated SensHook in combination with the IoT framework representative test hooks in the system are needed. This chapter will describe this environment and the used hardware, software, tools and platforms.

Test environment

This paragraph describes the test environment and the complete system setup used during this SAT.

Test setups, tools, hooks and probes

This paragraph describes the test setups, tools, hooks and probes used in the by this document defined tests.

A test setup defines a setup used for a test, and describe the used system parts, interface etc.

Tools define the used COTS tooling that will be used during testing e.g. Logic analyzers, packet sniffer like e.g. Whireshark, developed scripts, etc.

Hooks are used to inject data or control parts of the system used to get the system to handle a needed scenario. This can be hidden, debug or service interfaces which contain the needed functionality.

Probes are used to gather logging from parts of the system under test to provide system feedback and to prove that the system is handling the scenario as it should.

The following paragraphs will define the used setups, tools, hooks and probes which will be referred to by the test descriptions.

TS_01 Unit Test Setup

The unit testing used for testing the Java/OSGI API is done with the IDE *Eclipse* and the framework *JUnit*.

TS_02 InfluxDB Test setup

For the InfluxDB testing we open a terminal and start the *influx* application. This gives us a prompt from which we can issue commands to the databases.

TS_03 Postman test setup

To test the integration with the middleware platform and virtual gateway we use *Postman* and a call to the middleware platform to retrieve status updates from it. Below you can see a diagram of the test setup:

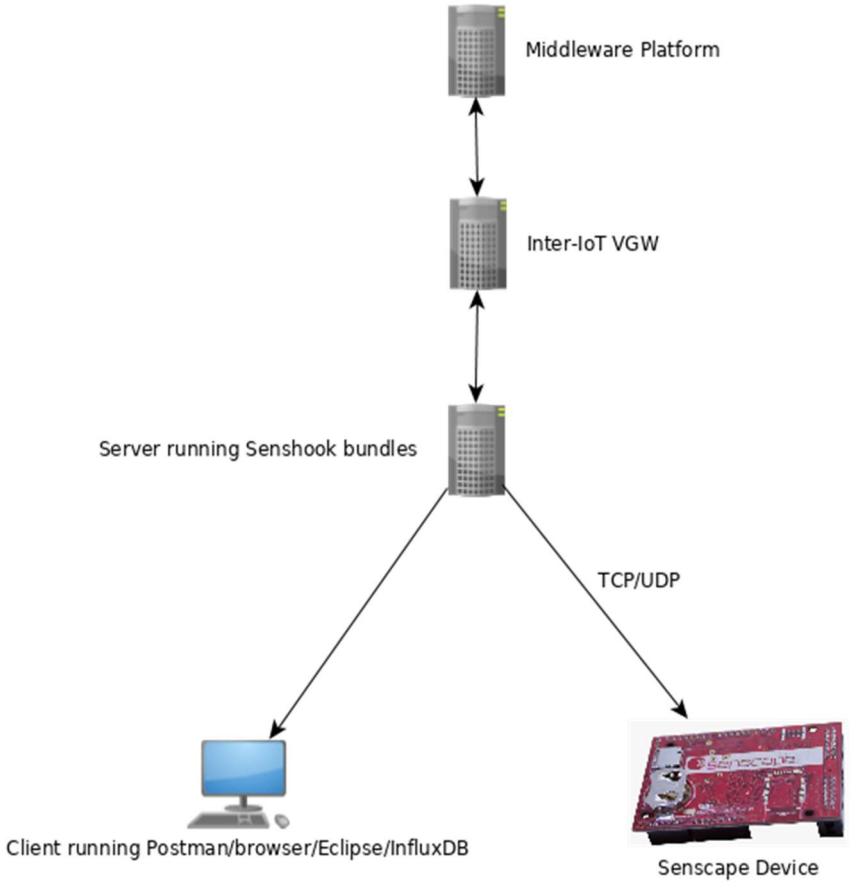


Figure 79: Test setup

TT_01 JUnit

JUnit is a framework for unit testing in Java, it will be used for the testing of the Java/OSGI API tests.

TT_02 Eclipse

Eclipse is an IDE widely used for developing in Java.

TT_03 Postman

Postman is a free API development environment. It offers the option to write complete tests for a REST API, so we use it to do the integration testing with the middleware platform as they communicate via HTTP.

TT_04 InfluxDB

We use the console of InfluxDB to perform some tests. The *Influx* database is designed for times series which makes it well suited for monitoring and analytics.

TH_01 JUnit test script

For the unit tests the *JUnit* test script is responsible for setting up the context of the different tests.

TH_02 Postman test method

For the virtual gateway and middleware platform integration tests the *Postman* test method is responsible for setting up the context.

TP_01 JUnit test script

For the unit tests the *JUnit* test script is responsible for giving the feedback of the test results.

TP_02 Postman test method

For the virtual gateway and middleware platform integration tests the Postman test method is responsible giving the feedback of the test results.

3.3.6.5 Test description

Scenario 15 Surveillance systems for prevention programs

Non-native species cost the EU €12 billion per year in damage and control costs. In the last decades several species of disease carrying mosquitoes have invaded Europe through the transport of goods, increasing international travel and climate change.

SensHook will reduce inspection costs and improves surveillance programs. With our new electronic trap, we will be the first in the world to combine human mimicking with automatic pest information in their value proposition. This allows a whole new population of consumers to establish surveillance programs that were only accessible to those with significant resources.

The demonstration of the Senshook architecture will be based on JAVA bundles that makes up the software. SensHook is designed to enable bi-directional communication between the middleware platform and itself and between the middleware and traps. SensHook is developed for receiving data from the traps in the field, and the database architecture is

defined for the storage of data gathered by the traps. This software module will be able to read data received from the field and store it in the system database.

Connecting and detecting devices

T15.1.1 Device identification at dispatcher component junit test

ID	T15.1.1
Test	Device identification at dispatcher component junit test
Type	System Testing
Setup	TT_01, TT_02, TS_01, TH_01, TP_01
Start	InfluxDB, VGW, middleware platform running
Req.	[11], [284], [110], [123], [283]
Input	Execute Junit test 'testGetTrap'
Outcome	Pass / Fail

T15.1.2 Trap generation at storage component junit test

ID	T15.1.2
Test	Trap generation at storage component junit test
Type	System Testing
Setup	TT_01, TT_02, TS_01, TH_01, TP_01
Start	InfluxDB, VGW, middleware platform running
Req.	[110], [265], [283]
Input	Execute Junit test 'testMakeGetTrap'
Outcome	Pass / Fail

T15.1.3 Sensor generation at storage component junit test

ID	T15.1.3
Test	Sensor generation at storage component junit test
Type	System Testing
Setup	TT_01, TT_02, TS_01, TH_01, TP_01
Start	InfluxDB, VGW, middleware platform running
Req.	[110], [265], [283]
Input	Execute Junit test 'testGetMakeSensor'
Outcome	Pass / Fail

T15.1.4 Trap registration at storage component junit test

ID	T15.1.4
Test	Trap registration at storage component junit test
Type	System Testing

Setup	TT_02, TT_03, TS_01, TH_01, TP_01
Start	InfluxDB, VGW, middleware platform running
Req.	[11], [110], [265], [283]
Input	Execute Junit test ‘testGetMakeSensor’
Outcome	Pass / Fail

T15.1.5 Trap registration at middleware platform

ID	T15.1.5
Test	Trap registration at middleware platform
Type	System Testing
Setup	TT_02, TT_03, TS_03, TH_02, TP_02
Start	InfluxDB, VGW, middleware platform, Postman running
Req.	[15], [110]
Input	Register a new trap Execute Postman call ‘Get All Entities’
Output	Registered devices of the middleware platform
Outcome	Pass / Fail

T15.1.6 Trap registration at influxDB

ID	T15.1.6
Test	Trap registration at influxDB
Type	System Testing
Setup	TT_02, TT_04, TS_02
Start	InfluxDB, VGW, middleware platform running
Req.	[57], [154], [273]
Input	Register a new trap Open influx application Execute query “SELECT * FROM trap”
Output	Registered traps
Outcome	Pass / Fail

Obtain information about connected sensors and traps

Information about the connected sensors to the different Senscape devices is retrieved. Senscape implements the IEEE 1451.4 Transducer Electronic Data Sheets (TEDS) standard. Besides the ability to retrieve detailed information about the connected sensors it also provides plug and play functionality for sensors.

15.2.1 Trap information retrieval at dispatcher component junit test

ID	T15.2.1
Test	Trap information retrieval at dispatcher component junit test
Type	System Testing
Setup	TT_01, TT_02, TS_01, TH_01, TP_01
Start	InfluxDB, VGW, middleware platform running
Req.	[57], [110], [266], [273], [283]
Input	Execute Junit test ‘testGetTrap’
Outcome	Pass / Fail

15.2.2 Trap list retrieval at dispatcher component junit test

ID	T15.2.2
Test	Trap list retrieval at dispatcher component junit test
Type	System Testing
Setup	TT_01, TT_02, TS_01, TH_01, TP_01
Start	InfluxDB, VGW, middleware platform running
Req.	[57], [110], [266], [273], [283]
Input	Execute Junit test ‘testTrapList’
Outcome	Pass / Fail

15.2.3 Sensor list retrieval at dispatcher component junit test

ID	T15.2.3
Test	Sensor list retrieval at dispatcher component junit test
Type	System Testing
Setup	TT_01, TT_02, TS_01, TH_01, TP_01
Start	InfluxDB, VGW, middleware platform running
Req.	[57], [110], [266], [273], [283]
Input	Execute Junit test ‘testSensorList’
Outcome	Pass / Fail

15.2.4 Trap information retrieval at storage component junit test

ID	T15.2.4
Test	Trap information retrieval at storage component junit test
Type	System Testing
Setup	TT_01, TT_02, TS_01, TH_01, TP_01
Start	InfluxDB, VGW, middleware platform running
Req.	[57], [110], [266], [273], [283]

Input	Execute Junit test ‘testMakeGetTrap’
Outcome	Pass / Fail

15.2.5 Trap list retrieval at storage component junit test

ID	T15.2.5
Test	Trap list retrieval at storage component junit test
Type	System Testing
Setup	TT_01, TT_02, TS_01, TH_01, TP_01
Start	InfluxDB, VGW, middleware platform running
Req.	[57], [110], [266], [273], [283]
Input	Execute Junit test ‘testGetTrapList’
Outcome	Pass / Fail

15.2.6 Sensor list retrieval at storage component junit test

ID	T15.2.6
Test	Sensor list retrieval at storage component junit test
Type	System Testing
Setup	TT_01, TT_02, TS_01, TH_01, TP_01
Start	InfluxDB, VGW, middleware platform running
Req.	[57], [110], [266], [273], [283]
Input	Execute Junit test ‘testGetMakeSensor’
Output	Pass / Fail

15.2.7 Trap information retrieval at virtual gateway connector component junit test

ID	T15.2.7
Test	Trap retrieval at virtual gateway connector component junit test
Type	System Testing
Setup	TT_01, TT_02, TS_01, TH_01, TP_01
Start	InfluxDB, VGW, middleware platform running
Req.	[47], [57], [110], [266], [273], [283]
Input	Execute Junit test ‘testGetCompleteTrap’
Outcome	Pass / Fail

15.2.8 Trap list retrieval at virtual gateway connector component junit test

ID	T15.2.8
Test	Trap list retrieval at virtual gateway connector component junit test
Type	System Testing
Setup	TT_01, TT_02, TS_01, TH_01, TP_01
Start	InfluxDB, VGW, middleware platform running
Req.	[47], [57], [110], [266], [273], [283]
Input	Execute Junit test ‘testGetCompleteTraps’
Outcome	Pass / Fail

T15.2.9 Measurement registration at middleware platform

ID	T15.2.9
Test	Trap registration at middleware platform
Type	System Testing
Setup	TT_02, TT_03, TS_03, TH_02, TP_02
Start	InfluxDB, VGW, middleware platform running
Req.	[15], [110]
Input	Connect a trap with new measurements Execute Postman call ‘Get All Entities’
Output	Registered measurement of the middleware platform
Outcome	Pass / Fail

T15.2.10 Measurement registration at influxDB

ID	T15.2.10
Test	Measurement registration at influxDB
Type	System Testing
Setup	TT_02, TT_04, TS_02
Start	InfluxDB, VGW, middleware platform running
Req.	[57], [154], [273]
Input	Register a new trap Open influx application Execute query “SELECT * FROM sensor”
Output	Registered measurements
Outcome	Pass / Fail

3.3.6.6 Test outcome overview

The following table will provide an overview of the test result of all the performed tests in this SAT.

Test	Description	Outcome
T15.1.1	Device identification at dispatcher component junit test	Pass / Fail
T15.1.2	Trap generation at storage component junit test	Pass / Fail
T15.1.3	Sensor generation at storage component junit test	Pass / Fail
T15.1.4	Trap registration at storage component junit test	Pass / Fail
T15.1.5	Trap registration at middleware platform	Pass / Fail
T15.1.6	Trap registration at influxDB	Pass / Fail
T15.2.1	Trap information retrieval at dispatcher component junit test	Pass / Fail
T15.2.2	Trap list retrieval at dispatcher component junit test	Pass / Fail
T15.2.3	Sensor list retrieval at dispatcher component junit test	Pass / Fail
T15.2.4	Trap information retrieval at storage component junit test	Pass / Fail
T15.2.5	Trap list retrieval at storage component junit test	Pass / Fail
T15.2.6	Sensor list retrieval at storage component junit test	Pass / Fail
T15.2.7	Trap retrieval at virtual gateway connector component junit test	Pass / Fail
T15.2.8	Trap list retrieval at virtual gateway connector component junit test	Pass / Fail
T15.2.9	Trap registration at middleware platform	Pass / Fail
T15.2.10	Measurement registration at influxDB	Pass / Fail
SAT Outcome		**Pass / Fail**

Table 50: Test outcome overview

3.3.6.7 Integration ethics and security

Introduction

This section addresses recommendations and proposed solutions to the ethical and security points of the integration.

SensHook

There are at least 3,528 species of mosquitoes. The majority are harmless to humans, but a few dozen species transmit diseases. SensHook addresses the major problem of disease-carrying Invasive Mosquito Species (IMS) that invade Europe due to climate change. The most threatening is the Asian Tiger Mosquito (Aedes albopictus), a vector that can transmit several serious and life threatening diseases.

3.3.7 Third Party: SOFOS

The imminent arrival of the Internet of Things (IoT), which consists of a vast variety of devices with heterogeneous characteristics, means that future networks need a new architecture to accommodate end-to-end IoT networking, dealing with: i) the expected increase in data generation, ii) the problems related to the end-to-end IP networking of the resource-constrained IoT devices, iii) the capacity mismatch between devices, and iv) the rapid interaction between services and infrastructure.

Software defined networking (SDN) and network function virtualization (NFV) are two technologies that promise to cost-effectively provide the scale and versatility necessary for IoT services in order to address efficiently the aforementioned challenges. Moreover, given that SDN and NFV are considered a fundamental component in the 5G landscape, since it is widely recognized that 5G networks will be software-driven and most components of future heterogeneous 5G architectures should be capable to support software-network technologies, both SDN and NFV are promising candidate technologies for a Software Defined Approach of end-to-end IoT Networking.

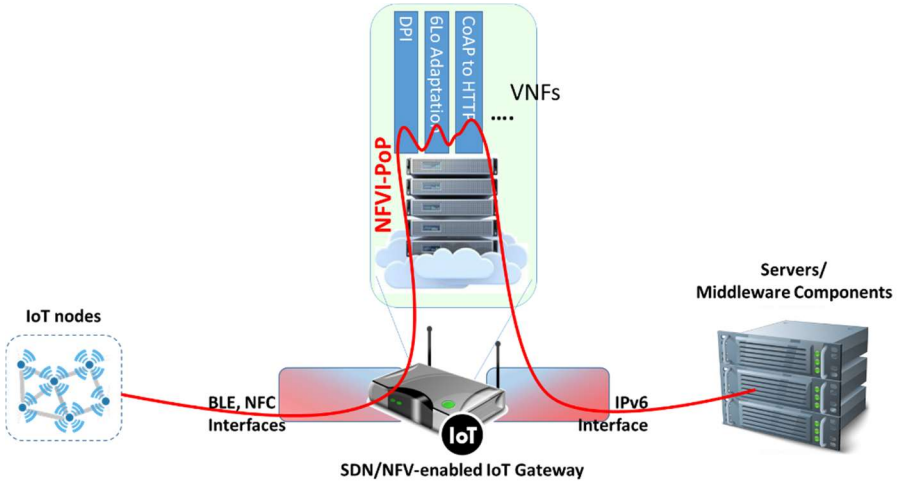


Figure 80: The proposed SDN/NFV end-to-end IoT Gateway overview

SOFOS aims at advancing the existing INTER-IoT framework with SDN and NFV functionalities towards a Software-defined end-to-end IoT infrastructure with IoT service chaining support. The main objective of the proposed SDN/NFV-enabled framework is to enhance the interoperability of the INTER-IoT framework in order to facilitate the interoperable management of a large number of diverse smart objects that currently operate utilizing a variety of different IoT protocols.

In this framework, specific objectives of the proposal include:

- To add SDN/NFV Automation and Verification in IoT Infrastructure
- To relocate various IoT functions from HW appliances to Virtual Machines (VMs) (i.e. Virtual Network Functions - VNFs).
- To enhance the interoperability support of the INTER-IoT platform by deploying VNFs that map IoT protocols (such as CoAP, MQTT) to standard IP networking
- To connect and chain the software-defined IoT functions (i.e. VNFs) together.
- To abstract the IoT's control plane by exploiting the SDN concept and advances.
- INTER-IoT Infrastructure with the proposed advances can be enhanced by means of NFV with integration of SDN, making it more agile and introducing a high degree of automation in service delivery and operation—from dynamic IoT service parameter exposure and negotiation to resource allocation, service fulfilment, and assurance.

3.3.7.1 Integration of IoT framework

SOFOS experiment considers that INFOLYSiS will deploy on top of INTER-IoT vGW modules that provide SDN/NFV Automation in IoT Infrastructure, such as the INFOLYSiS SDN/NFV Network Manager. By applying appropriate OPENFLOW commands, INFOLYSiS add-on will steer the data traffic from the INTER-IoT vGW to the various VNFs that will have been deployed in order to enhance the interoperability functions of INTER-IoT, allowing to the application layer to represent the received data in a unified way.

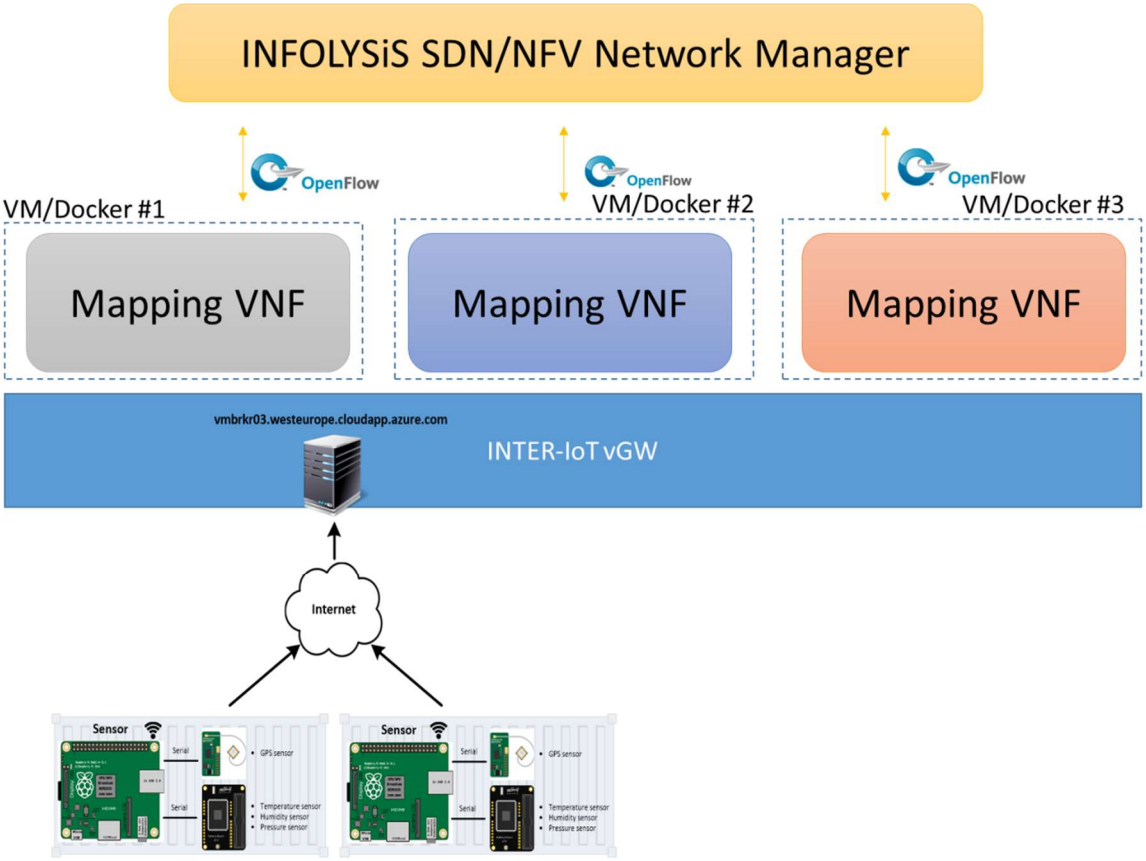


Figure 81: SOFOS Integration and Factory test setup overview.

Thus, with the mapping VNFs provided by INFOLYSiS and the support of the SDN/NFV techniques, the data provided by Raspberry and panStamp are mapped to a common protocol (e.g. HTTP). For the instantiation of the virtual functions, an SDN-compatible (i.e. OpenFlow compliant) cloud computing platform is considered at the MW layer for enterprise users, such as the Docker approach of the INTER-IoT framework.

The container-based topology of the Factory test setup overview, is depicted in the following figure, where each box represents a container on top of the Docker-based virtualization.

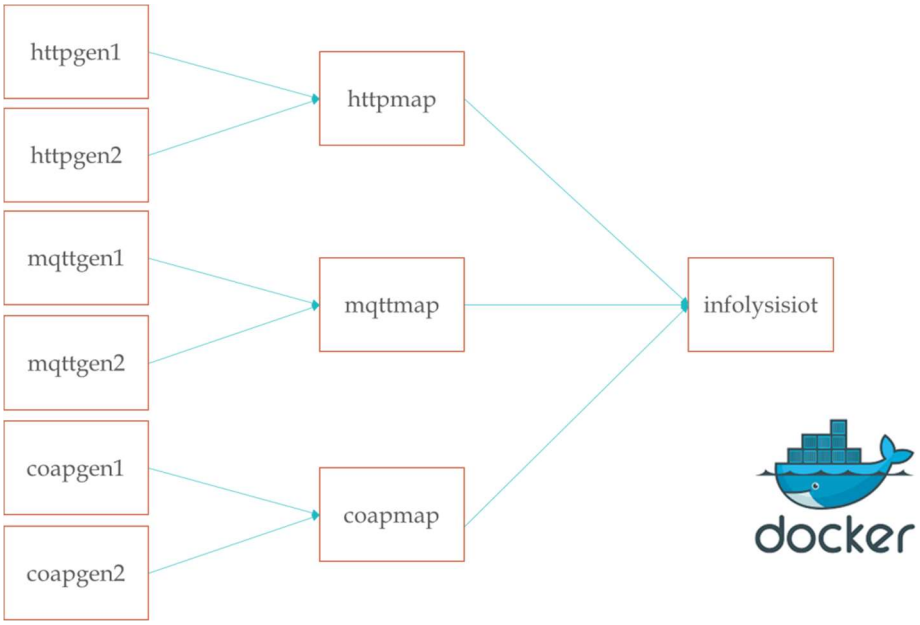


Figure 82: SOFOS Integration and Factory test setup logical topology.

The IoT node generators for the factory setup (i.e. httpgen1/2, mqttgen1/2, coapgen1/2) will be based on data provided by Raspberry and panStamp units of the INTER-IoT platform or other HW-based IoT nodes that will be available by INTER-IoT system during the early integration phase.

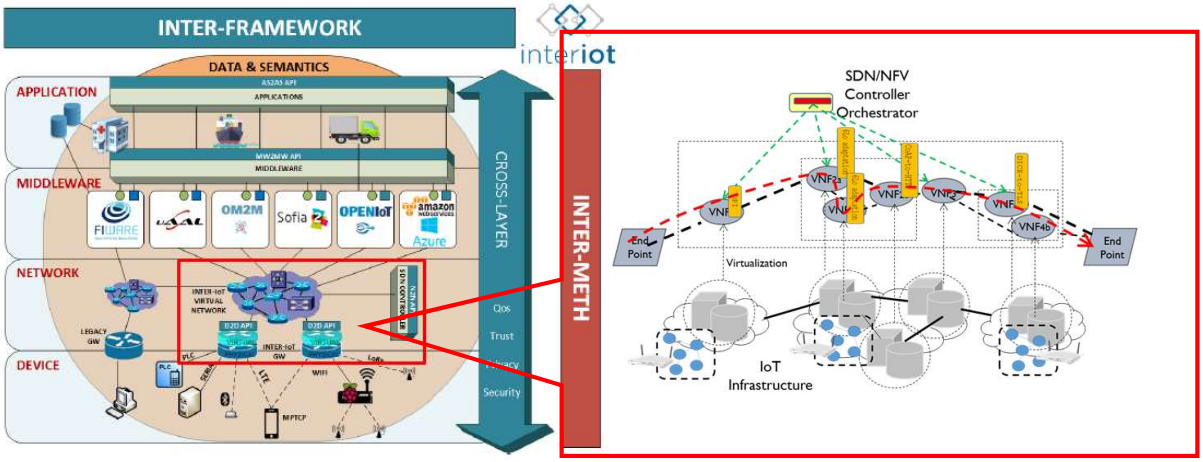


Figure 83: Collaboration approach SDN/NFV infrastructure in INTER-IoT architecture

SOFOS experiment considers that INFOLYSiS will deploy the necessary mapping VNFs (i.e. proxies) on the top of the relevant virtual INTER-IoT GWs. The necessity for the SDN management on top of each testbed it is depicted in the Figure 2, where it is shown that without the proposed SOFOS SDN/NFV-based IoT system, the IoT nodes forwards the data traffic directly to the respective IoT GW, which is not capable to understand the different protocols and therefore communication is not achieved. With SOFOS SDN/NFV-based IoT system, by applying appropriate OPENFLOW commands via the SDN Controller, the data traffic from the IoT nodes will be routed (1) to the SDN-node/switch, then (2) will be diverted/routed/steered by the SDN switch of the testbed (due to appropriate Openflow commands/programming) towards the mapping function in order to be translated to the "interoperable" protocol (e.g. HTTP in the example) and (3) then (once it has been translated) will be further forwarded/routed back to the SDN switch and finally (4) from there to the original destination i.e. the IoT vGW.

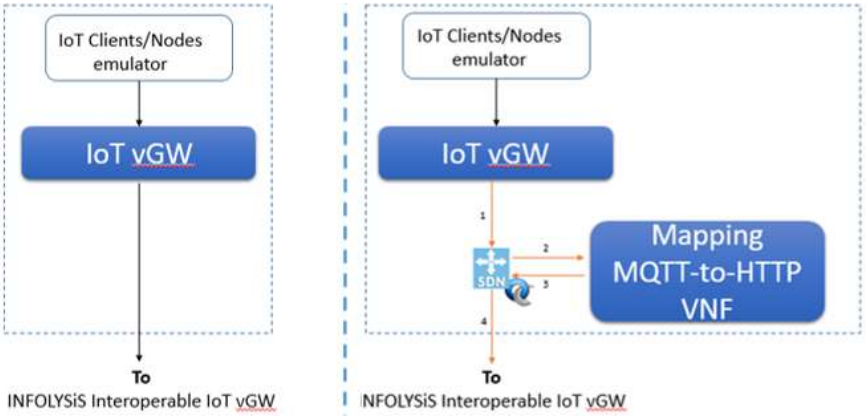


Figure 84: Detailed approach of SDN applicability on top of INTER-IoT

3.3.7.2 Deliverables and version overview

The following table contains a deliverable list which need to be signed of before SAT testing commences.

ID	Description	Check
Documents		
1	Validation and Test reports of the IoT system components	
2	Validation and Test reports of the Pilot system components	
Hardware		
3	UDP port GW of the INTER-IoT	
Tools		
4	OpenVPN	
5	TCPDUMP	
6	OVS-OFCTL – SDN flows monitoring tool	
7	BMON – Bandwidth monitoring in interfaces	
8	DOCKER STATS – Resource usage of docker containers	

Table 51: Deliverable checklist

The following table shows the software components and version of which the system release version 1.0 consists of.

ID	Description	Version	Check
IoT Physical Gateway			
1	AN Controller	V1.0.3	
IoT Virtual Gateway			
3	Fiware	V4.2.3	
4	Docker		
5	Ubuntu		
UniversAAL container			
6	UniversAAL REST API	V3.2.1	

Table 52: Component version overview

3.3.7.3 Requirements, scenarios and use cases

S1 - Accident at the port area: Fire, Explosion, Extreme weather conditions

This scenario considers an emergency situation that happens at the port and is related to incidents related to fire, explosion and/or extreme weather conditions. The implementation and use of the SDN paradigm by SOFOS will speed up IoT connections, provide interoperability among different IoT devices and centralize the management at the port domain. Moreover, the SDN applicability will allow the prioritization of IoT data flows using traffic engineering, achieving a general overview of the whole network at any time. SOFOS pilot will provide at the first responders' commander, who will coordinate the emergency/rescue teams, a unified view of IoT data visualisation. More specifically, the proposed SDN/NFV-enabled IoT GW will be used to provide interoperability between port IoT systems on the different locations with the coordination center at the port with scope to provide a common unified view of the accident, its type and the location of the available rescue teams. For this purpose, a virtual mapping function that implements an existing interoperability standards commonly used in information systems will be deployed by the SDN/NFV orchestrator, offering interoperable and continuous data transmission, allowing to the coordinator to allocate at each available rescue unit the appropriate incident, degree of severity and its location.

Interoperability in this scenario is required to connect IoT devices providing IoT port data related to: fire (using temperature, humidity data), explosion (using noise, temperature, wind data) and also about extreme weather conditions (using weather related data such as wind speed, wind direction, pressure, rain, humidity etc): .

The resulting service will be obtained by the integration of:

- the historical data from the port IoT platform and port IoT sensors
- SOFOS SDN-based interoperable vGW

The following table provides an overview of the relation between the requirements and the test(s) that validate their implementation.

ID	Description	Covered by
2	Scalability. Design	T6, T7, T8, T9
13	Extensibility	T6, T7, T8, T9
15	Support of common IoT communication protocols	T1, T2, T3, T4
21	Real time output	T1, T2, T3, T4, T7, T8, T9
27	System security	T5
28	System privacy	T5
70	Easy-to-use user interface	T4, T7, T8, T9
78	Automatic and dynamic selection of communication protocol	T1, T2, T3
95	Robustness, resilience and availability	T6
229	SDN capabilities	T1, T2, T3, T4, T6, T7, T8, T9
231	Network function virtualization	T1, T2, T3, T4, T6, T7, T8, T9
244	Gateway virtualization	T4, T5, T6
281	Publish data stream into a platform	T5, T7, T8, T9

Table 53: Requirements vs. test mapping

The following table provides an overview of the relation between the scenarios and the test(s) that validate their implementation.

ID	Scenario name	Covered by
9.1	Accident at the port area - Fire	T1, T2, T3, T4, T5, T6, T7
9.2	Accident at the port area - Explosion	T1, T2, T3, T4, T5, T6, T8
9.3	Accident at the port area - Extreme weather conditions	T1, T2, T3, T4, T5, T6, T9

Table 54: Scenario vs test mapping

3.3.7.4 Test environment

Introduction

To test the functionality of the SOFOS in combination with the IoT framework a representative test system is needed. The test system needs to approach the “real world” as much as possible. The pilot setups must be recreated and proven. This chapter will describe this environment and the used hardware, software, tools and platforms.

Test environment

This paragraph describes the test setups, hooks and probes used in the system used during this SAT.

Test tools, hooks and probes

This paragraph describes the test setups, tools, hooks and probes used in the by this document defined tests.

A test setup defines a setup used for a test, and describe the used system parts, interface etc.

Tools define the used COTS tooling that will be used during testing e.g. Logic analyzers, packet sniffer like e.g. Whireshark, developed scripts, etc.

Hooks are used to inject data or control parts of the system used to get the system to handle a needed scenario. This can be hidden, debug or service interfaces which contain the needed functionality.

Probes are used to gather logging from parts of the system under test to provide system feedback and to prove that the system is handling the scenario as it should.

The following paragraphs will define the used setups, tools, hooks and probes which will be referred to by the test descriptions.

Test setup

The following topology refers to the test setup that will be used for verifying the SOFOS solution. The IP addresses of the figure refer to the ones that have been already used for testing at INFOLYSiS testbed environment. Appropriate ones will be used, while the IoT nodes/emulators that are currently used for testing will be replaced by actual HW-based IoT nodes provided by INTER-IoT ecosystem. INFOLYSIS Automated Alarm System (IAAS) will detect and communicate to INTER-IoT GWs the port emergency incidents as per the scenarios IDs 9.1, 9.2 and 9.3.

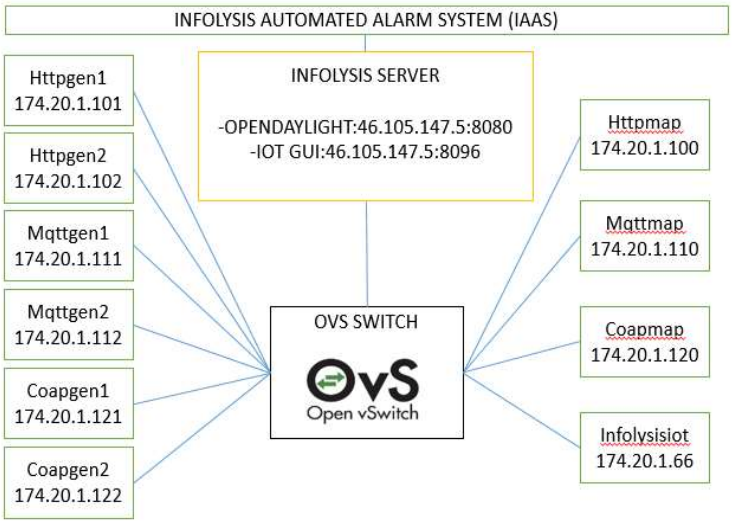


Figure 85: Test setup of SOFOS solution.

Test tools

Below are described the four test tools that are used for the SOFOS SAT.

TT_01 Test tool TCPDUMP

TCPDUMP - Packet sniffers can be used for verifying the IoT protocols that are entering to the SOFOS systems and the ones that are translated when are existing the system.

TT_02 Test tool OVS-OFCTL

OVS-OFCTL – SDN flows monitoring tool

TT_03 Test tool BMON

BMON – Bandwidth monitoring in interfaces

TT_04 Test tool DOCKER STATS

DOCKER STATS – Resource usage of Docker containers

Test probes

TP_01 Test probe INFOLYSiS IoT vGW

INFOLYSiS IoT vGW provides a detailed monitoring interface, which shows in real time the data that are translated by the mapping functions

TP_02 Test probe INFOLYSiS VNFs

VNFs mapping functions translating data of various IoT protocols (CoAP, MQTT, HTTP) to generic UDP streams, proving that the system is handling the scenario as it should.

3.3.7.5 Test description

SOFOS SAT scenario considers an emergency situation that happens at the port and is related to emergency incidents related to fire, explosion and/or extreme weather conditions (Scenario IDs 9.1, 9.2 and 9.3).

In specific, real data are provided by the port IoT sensors related to temperature values, sound values and wind. These data are fetched to INFOLYSiS IoT protocol generators/aggregators (MQTT, CoAP, HTTP) and then to INFOLYSiS VNFs mapping functions in order to be translated into UDP streams and achieve IoT interoperability before publishing them in real time to the INTER-IoT platform under UDP protocol.

As a real word case scenario, INFOLYSiS IoT vGW analyzes the received port data and notifies INTER-IoT platform in a unified way (interoperability achieved by the provision of the processed data under UDP protocol) for emergency cases of fire, explosion and/or extreme weather conditions that are taking place at the port. Please find below the details of the planned SAT test.

T1-SOFOS Interoperability and Port Emergency Alarms for Scenario IDs 9.1, 9.2 and 9.3

ID	T1
Test	Real Port Sensors data testing and identification of Emergency incidents
Type	IoT Interoperability and Port Sensor Data testing
Setup	MQTT generator, MQTT mapping VNF, CoAP generator, CoAP mapping VNF, HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	MQTT, CoAP and HTTP generators produce and send data
Req.	[2],[13],[21],[70],[229],[231],[281]
Input	MQTT, CoAP, HTTP port sensor data (temperature, sound, wind etc)
Output	Alarm of emergency incidents
Logs	Logs in dedicated folder
Outcome	Pass / Fail

Within the framework of the SOFOS SAT planning and initial testing, 3 main categories of tests have been performed:

- SOFOS Infrastructure readiness and sensitivity tests
- Emergency detection tests (Scenario IDs 9.1, 9.2, 9.3)
- SOFOS Add-on features and SOFOS Components failure tests

SOFOS Infrastructure readiness and sensitivity tests

In total 7 tests were performed as part of this SOFOS SAT category of technical level tests in order to verify the readiness of SOFOS infrastructure before proceeding to the port use case scenarios testing (alarms for port emergency cases as per scenarios IDs 9.1, 9.2 and 9.3). Please find below the details of each performed test along with its outcome:

T1

Test	MQTT mapping to UDP-based real port data
Type	VNF testing using real port data
Setup	MQTT generator, MQTT mapping VNF, Infolysis IoT GW, OpenVSwitch

Start	MQTT generator produces and sends data
Req.	[15],[21],[78],[229],[231]
Input	Real port MQTT sensor data
Output	UDP-based real port sensor data
Outcome	Pass / Fail

T2

Test	CoAP mapping to UDP-based real port data
Type	VNF testing using real port data
Setup	CoAP generator, CoAP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	CoAP generator produces and sends data
Req.	[15],[21],[78],[229],[231]
Input	Real port CoAP sensor data
Output	UDP-based real port sensor data
Outcome	Pass / Fail

T3

Test	HTTP mapping to UDP-based real port data
Type	VNF testing using real port data
Setup	HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	HTTP generator produces and sends data
Req.	[15],[21],[78],[229],[231]
Input	Real port HTTP sensor data
Output	UDP-based real port sensor data
Outcome	Pass / Fail

T4

Test	Real time monitoring from INFOLYSiS IoT GW
Type	System testing using real port data
Setup	MQTT generator, MQTT mapping VNF, CoAP generator, CoAP mapping VNF, HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	MQTT, CoAP and HTTP generators produce and send data
Req.	[15],[21],[70],[229],[231],[244]
Input	MQTT, CoAP, HTTP real port sensor data
Output	Real time graphs of port sensor data in INFOLYSiS IoT GW
Outcome	Pass / Fail

T5

Test	Real time publishing of real port data in INTER-IOT platform with UDP
Type	Integration testing using real port data
Setup	MQTT generator, MQTT mapping VNF, CoAP generator, CoAP mapping VNF, HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	UDP data arrive at INFOLYSiS IoT GW
Req.	[27],[28],[244],[281]
Input	UDP-based real port sensor data
Output	Successful delivery of UDP messages to INTER-IoT platform
Outcome	Pass / Fail

T6

Test	Handling more than one data flows from each protocol
Type	System testing using real port data
Setup	MQTT generator, MQTT mapping VNF, CoAP generator, CoAP mapping VNF, HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	MQTT, CoAP and HTTP generators produce and send data
Req.	[2],[13],[229],[231],[244]
Input	MQTT, CoAP, HTTP port sensor data
Output	Real time UDP port sensor data in INFOLYSiS IoT GW
Outcome	Pass / Fail

T7

Test	Sensitivity testing to define the threshold values per data type (temperature, sound, wind) above which emergency alarms should be triggered
Type	Sensitivity testing – Alert values defined – Real port data are used
Setup	MQTT generator, MQTT mapping VNF, CoAP generator, CoAP mapping VNF, HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	MQTT, CoAP and HTTP generators produce and send data
Req.	[2],[13],[229],[231],[244]
Input	MQTT, CoAP, HTTP real port sensor data
Output	Threshold values above which alarm is created
Outcome	Pass / Fail

Emergency detection tests

In total 3 tests were performed as part of this SOFOS SAT category of tests in order to indicate the cases at which the provided port sensors real data are above the set limits of T7

and an alarm should be communicated to INTER-IoT platform denoting emergency cases of fire, explosion, extreme weather conditions. Please find below the details of each performed test along with its outcome. Real port data have been used which for the purpose of the tests have been modified accordingly in order alarms to be triggered:

T8

Test	Identifying fire from real port sensor data (Scenario ID 9.1)
Type	Temperature Port Sensor data testing
Setup	MQTT generator, MQTT mapping VNF, CoAP generator, CoAP mapping VNF, HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	MQTT, CoAP and HTTP generators produce and send data
Req.	[2],[13],[21],[70],[229],[231],[281]
Input	MQTT, CoAP, HTTP port temperature sensor data
Output	Alarm of fire since values above set thresholds have been identified
Outcome	Pass / Fail

T9

Test	Identifying explosions from real port sensor data (Scenario ID 9.2)
Type	Sound Port Sensor data testing
Setup	MQTT generator, MQTT mapping VNF, CoAP generator, CoAP mapping VNF, HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	MQTT, CoAP and HTTP generators produce and send data
Req.	[2],[13],[21],[70],[229],[231],[281]
Input	MQTT, CoAP, HTTP port sound sensor data
Output	Alarm of explosion since values above set thresholds have been identified
Outcome	Pass / Fail

T10

Test	Identifying extreme weather conditions from real port sensor data (Scenario ID 9.3)
Type	Wind port Sensor data testing
Setup	MQTT generator, MQTT mapping VNF, CoAP generator, CoAP mapping VNF, HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	MQTT, CoAP and HTTP generators produce and send data
Req.	[2],[13],[21],[70],[229],[231],[281]
Input	MQTT, CoAP, HTTP port wind sensor data
Output	Alarm of extreme weather conditions since values above set thresholds have been identified
Outcome	Pass / Fail

SOFOS Add-on features and SOFOS components failure tests

In order to verify the functionality of additional features of the SOFOS solution, 6 extra tests were performed in order to verify add-on features (such as Disable alarm options and No Port Data received cases).

These add-on features are also useful for cases at which a component of SOFOS system fails (mapping VNFs or INFOLYSIS IoT vGW failure) and actions should be taken in order the system to keep up functioning efficiently (eg disable the faulty component) or alarms need to be triggered for the occurring malfunctions.

In particular, tests T11-T13 refer to cases in which due to faulty sensors functionality or faulty received data which may trigger continuously false alarms or simply because we want the system to offer this add-on feature (shutting down an alarm), we test malfunction incidents where we can disable on demand specific functionalities/alarms (eg disable fire alarm etc) and the rest system keeps up running.

Similarly, in T14, we test the case where due to an unexpected reason (faulty sensors, break of communication, power failure etc), no data are received or no data are processed by the INFOLYSiS VNFs and/or IoT vGW. Again in this case we trigger a warning/alarm that no data are received/processed by SOFOS system.

Finally, tests T15-T16 refer to potential malfunction cases of SOFOS mapping VNFs or SOFOS IoT vGW and the corresponding warnings/alarms that should be triggered to the INFOLYSIS Management GUI in order corrective actions (eg contingency plans) to be taken immediately into force.

T11

Test	Disable fire alarm
Type	Temperature Port Sensor data testing and inactivation of fire alarm notifications
Setup	MQTT generator, MQTT mapping VNF, CoAP generator, CoAP mapping VNF, HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	MQTT, CoAP and HTTP generators produce and send data
Req.	[2],[13],[21],[70],[229],[231],[281]
Input	MQTT, CoAP, HTTP port temperature sensor data
Output	Alarm of fire is not triggered although corresponding data may exist
Outcome	Pass / Fail

T12

Test	Disable explosion alarm
Type	Sound Port Sensor data testing and inactivation of explosion alarm notifications
Setup	MQTT generator, MQTT mapping VNF, CoAP generator, CoAP mapping VNF, HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	MQTT, CoAP and HTTP generators produce and send data
Req.	[2],[13],[21],[70],[229],[231],[281]
Input	MQTT, CoAP, HTTP port sound sensor data

Output	Alarm of explosion is not triggered although corresponding data may exist
Outcome	Pass / Fail

T13

Test	Disable extreme weather conditions alarm
Type	Port Sensor data testing and inactivation of extreme weather conditions alarm notifications
Setup	MQTT generator, MQTT mapping VNF, CoAP generator, CoAP mapping VNF, HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	MQTT, CoAP and HTTP generators produce and send data
Req.	[2],[13],[21],[70],[229],[231],[281]
Input	MQTT, CoAP, HTTP port wind sensor data
Output	Alarm of extreme weather conditions is not triggered although corresponding data may exist
Outcome	Pass / Fail

T14

Test	Identifying cases where no port data are received by INFOLYSiS IoT vGW
Type	Port Sensors data, SOFOS VNFs, and INFOLYSIS IoT vGW data testing – Lack of data/Communication Error
Setup	MQTT generator, MQTT mapping VNF, CoAP generator, CoAP mapping VNF, HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	MQTT, CoAP and HTTP generators produce and send data
Req.	[2],[13],[21],[70],[229],[231],[281]
Input	MQTT, CoAP, HTTP real port sensor data
Output	Alarm of no port data received for an X defined period of time
Outcome	Pass / Fail

T15

Test	Identifying cases where SOFOS mapping VNFs malfunction and do not provide UDP processed data to INFOLYSIS IoT vGW
Type	SOFOS mapping VNFs malfunction testing – Lack of UPD data from specific VNF(s)
Setup	MQTT generator, MQTT mapping VNF, CoAP generator, CoAP mapping VNF, HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	MQTT, CoAP and HTTP generators produce and send data
Req.	[2],[13],[21],[70],[229],[231],[281]
Input	MQTT, CoAP, HTTP real port sensor data
Output	Alarm of no UDP data received from X SOFOS mapping VNF
Outcome	Pass / Fail

T16

Test	Identifying cases where INFOLYSIS IoT vGW malfunctions and cannot produce appropriate output for the INTER-IoT system
Type	INFOLYSIS IoT vGW malfunction testing – Error processing/output
Setup	MQTT generator, MQTT mapping VNF, CoAP generator, CoAP mapping VNF, HTTP generator, HTTP mapping VNF, Infolysis IoT GW, OpenVSwitch
Start	MQTT, CoAP and HTTP generators produce and send data
Req.	[2],[13],[21],[70],[229],[231],[281]
Input	MQTT, CoAP, HTTP real port sensor data
Output	Master Alarm of not appropriate operation of INFOLYSIS IoT vGW
Outcome	Pass / Fail

3.3.7.6 Test outcome overview

The following table will provide an overview of the test result of all the performed tests in this SAT.

The following table will provide an overview of the test result of all the performed tests.

Test	Description	Outcome
T1	MQTT mapping to UDP-based real port data	Pass / Fail
T2	CoAP mapping to UDP-based real port data	Pass / Fail
T3	HTTP mapping to UDP-based real port data	Pass / Fail
T4	Real time monitoring from INFOLYSiS IoT GW	Pass / Fail
T5	Real time publishing of port data in INTER-IOT platform with UDP	Pass / Fail
T6	Handling more than one data flows from each protocol	Pass / Fail
T7	Sensitivity testing to define the threshold values per data type (temperature, sound, wind) above which emergency alarms should be triggered	Pass / Fail
T8	Identifying fire from real port sensor data	Pass / Fail
T9	Identifying explosions from real port sensor data	Pass / Fail
T10	Identifying extreme weather conditions from real port sensor data	Pass / Fail
T11	Disable fire alarm	Pass / Fail
T12	Disable explosion alarm	Pass / Fail
T13	Disable extreme weather conditions alarm	Pass / Fail
T14	Identifying cases where no port data are received by INFOLYSiS IoT vGW	Pass / Fail
T15	Identifying cases where SOFOS mapping VNFs malfunction and do not provide UDP processed data to INFOLYSIS IoT vGW	Pass / Fail
T16	Identifying cases where INFOLYSIS IoT vGW malfunctions and cannot produce appropriate output for the INTER-IoT system	Pass / Fail
SAT Outcome		**Pass / Fail**

Table 55: Test outcome overview

3.3.7.7 Integration ethics and security

Introduction

This section addresses recommendations and proposed solutions to the ethical and security points of the integration.

SOFOS will give prompt attention to any ethical issues that may arise as a result of project activities, and will address them in a professional way following established and upcoming EU regulations and the corresponding national laws about data protection, digital and property rights issues and protection of minors very closely.

SOFOS

Since the port IoT sensors are available by INTER-IoT project for SOFOS, then the execution of SOFOS experiment involves no human participants. In general, human participants are not involved and therefore Ethics/Privacy aspects are not considered.

If by any case, human participants are indirectly involved or affected, established procedures will be followed that respect all pertinent laws (Directive 95/46/EC and General Data Protection Regulation) and ethics standards, in particular related to contacting individuals, providing comprehensive and clear information about the objectives of the conducted research and the use of the collected data, obtaining their consent and not sharing any of the collected personal data with third parties.

The precise documentation that will be communicated to users participating in the pilot studies/case studies, if human participation is requested, will be made available and will be included as appendices to the final deliverable.

3.3.8 Third Party: ACHILLES

Access control and endpoint authentication in the IoT is a challenging problem. Things are usually small devices with limited storage capacity, power, energy, and processing capabilities, in order to be inexpensive and practical. In many cases Things are “exposed” to tampering, whereas in many application scenarios, after Things are deployed, it is not easy to access them. Things usually are not able to perform “heavy” tasks, such as complex cryptographic operations. Storing user credentials or any other sensitive information in a Thing creates security risks, adds storage overhead, and makes security management an impossible task. When it comes to interoperable applications, Things (or even gateways) cannot interpret complex business roles and processes. Moreover, companies are not willing to share sensitive information about their users with a Thing (or a gateway), even if this information is required by an access control mechanism, neither do they want to invest in yet another security system.

The ACHILLES project overcomes these limitations by allowing the delegation of security operations to a third party, referred to as the Access Control Provider (ACP), which can be implemented by a trusted separate entity, or even the service provider itself. The ACHILLES concept is depicted in Figure 1.

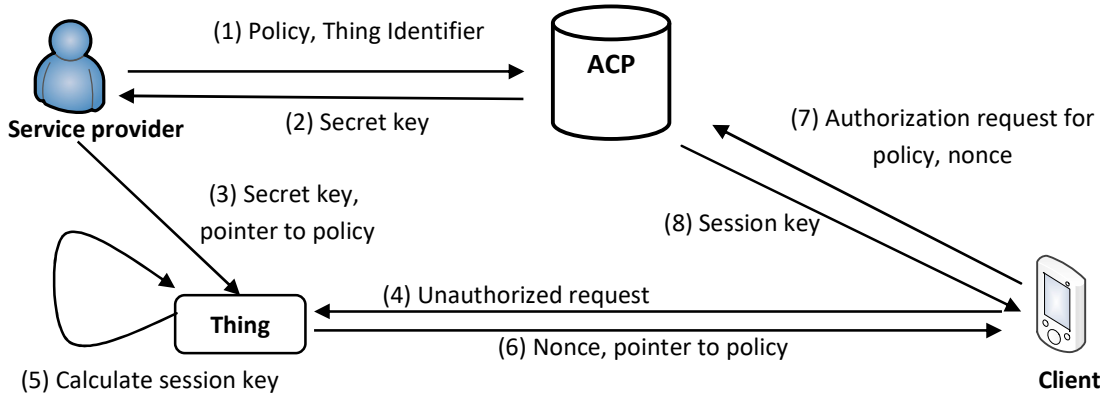


Figure 86: The concept of the ACHILLES project

The main idea of the ACHILLES concept is that IoT service providers store access control policies in ACPs and in return ACPs generate secret keys which are stored in Things (steps 1-2). These keys are generated, during a setup phase, using a secure hash with input the Thing identifier. Additionally, Things are configured with pointers (e.g., a URL that points to an ACP and a particular file) to the access control policies that protect sensitive resources (step 3). Every time a client requests access to a protected resource (step 4) the Thing uses a secure hash function to generate a session key (step 5). The secret key used by that function is the key generated by the ACP and the hash inputs are: (a) the pointer to the policy that protects the resource and (b) a random nonce. The Thing transmits the nonce and the pointer to the client (step 6), which in return requests authorization from the appropriate ACP (over a secure channel) (step 7). The ACP has all the necessary information required to calculate the session key: if the client is authorized, the ACP calculates the session key and transmits it back to the client (step 8). Providing that: (i) the Thing has not lied about its identity and (ii) the messages exchanged between the client and the Thing have not been modified, the Thing and the client end up sharing a secret key. This key can be used for securing subsequent communications (e.g., by using DTLS).

Testing system

Our testing system is illustrated in Figure 2. This system is more advanced compared to the system used for the FATs.

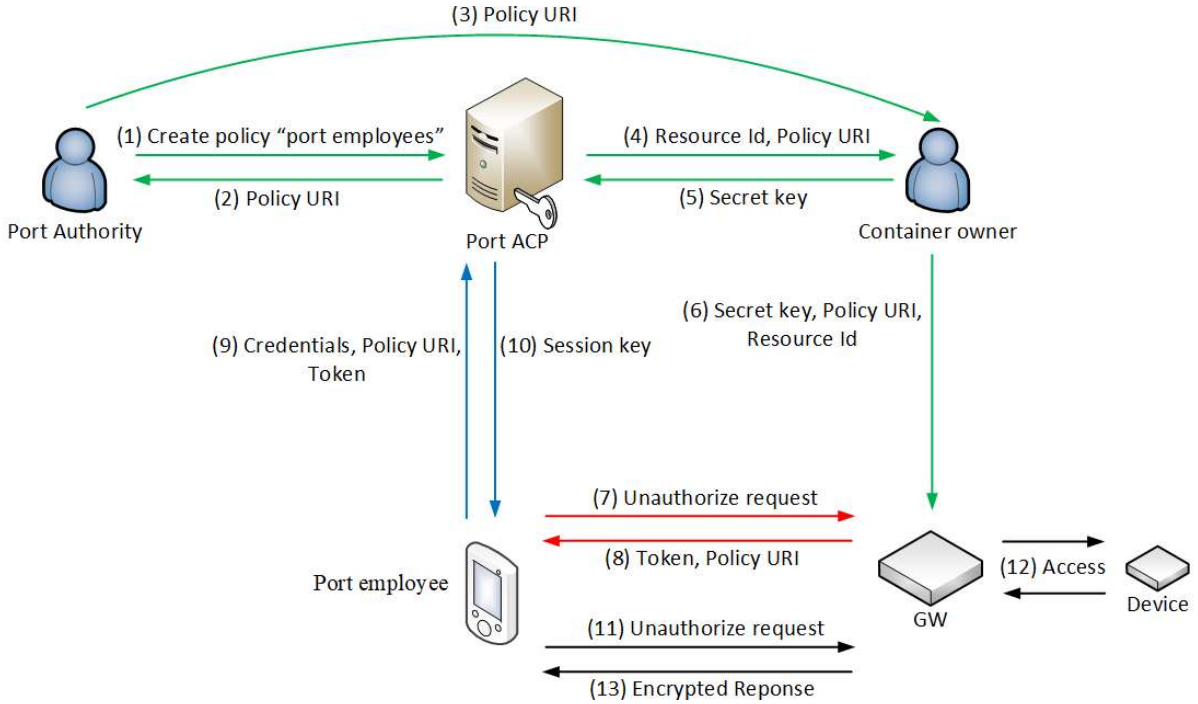


Figure 87: Testing system.

This system represents the use case of a "smart port". In this use case, port employees want to access information provided by Things embedded in a container, through an INTER-IoT gateway. Container owners want to make sure that only port employees can access this information. On the other hand, the port authority does not want to give access to its user management system. In order to overcome this problem, the port authority creates an access control policy in its ACP and receives back a URI to this policy; then it sends this URI to container owners (Steps 1-3). Each container owner "registers" the available resources to the ACP and receives back a secret key which is installed in the INTER-IoT gateway (steps 4-6). This key, which is only used for generating session keys, should be kept protected; in case of a breach this setup process should be repeated. With these the "setup" phase is completed. A port employee initially sends an "unauthorized request" and receives back a token and the URI of the policy that protects the desired resource (steps 7,8). Then the employee authenticates him/herself and obtains a session key (steps 9,10). The GW has also calculated the same key, which can then be used for retrieving securely the desired resource (steps 11-13). In the following we provide more technical details related to each phase of our system.

System entities

Our system considers the following entities: *Gateways* (GW), resource *owners*, resource *clients*, *authorities*, and *Access Control Providers* (ACPs). The goal of a resource owner is to provide a *resource* only to clients authorized by an authority, using an *authenticated* GW, over a secured communication channel. Access to a resource is regulated by an *access control policy*. Each access control policy is stored in an ACP and maps the authority-specific identification data of a client to a Boolean output (true, false) and to a policy specific identifier referred to as ID_{client}. The length of ID_{client} is 16 bits, limiting the number of users that can be

managed by an access control policy to 2^{16}. When the output of an access control policy is true, the client is considered authorized. Ideally, an access control policy should not be GW/resource specific, for example, an access control policy may output "true" for any employee of a particular company, this way access control policies become re-usable.

Although by design ACPs and resource owners are two distinct entities, in reality there can be cases where these roles are held by the same real world entity.

System setup

In the following we assume that each resource is identified by a pair of identifiers, namely URI_{host} and URI_{path}. For example, a resource name could be company1.iot/temperature where company1.iot is the URI_{host} and temperature is the URI_{path}.

Our system assumes an out-of-band and secured setup phase. During this phase, each ACP generates and securely stores a *Root Secret Key* (RSK). In addition, access control policies are created and stored in ACPs by authorities. For each policy, a *Uniform Resource Identifier* (URI) is generated. These URIs are of the form "*ACP location/access control policy name*". A policy URI, henceforth denoted as URI_{policy}, may be used by many resource owners: a resource owner does not have to be aware of the rules and the implementation details of an access control policy; the only information that a resource owner needs in order to protect a resource is a URI_{policy}.

Every resource owner that wants to use a policy to protect an URI_{host} issues a *secret key request* to an ACP. The ACP uses a secure HMAC and the RSK to compute a secret key. This secret key, denoted as $SK_{acp,host}$, is computed simply by hashing URI_{host} using the HMAC function and the RSK as the hash function key. The computed secret key is then securely delivered to the resource owner. Then, the resource owner configures the appropriated GW with URI_{policy} and $SK_{acp,host}$. Hence, each GW maintains an *Access Table* that contains tuples of the form [URI_{host}, URI_{policy}, $SK_{acp,host}$].

Figure 3 illustrates the system setup phase (ACP's RSK generation is omitted). In the illustrated example, an access control policy (Policy1) is installed in an ACP by an authority. Then, a resource owner requests a secret key for the compnay1.iot URI_{host}, the ACP generates this key (that is, $SK_{ACP,company1.iot}$) by using its RSK and an HMAC, and sends it back to the resource owner. Finally, the resource owner configures the GW with the appropriate information.

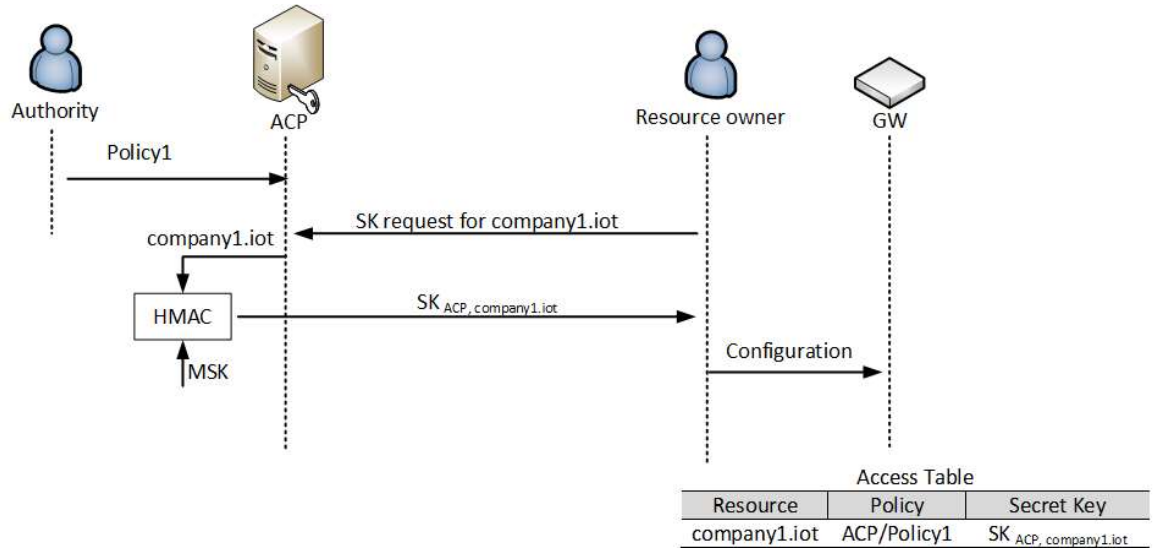


Figure 88: The setup phase.

It should be noted here that this phase takes place out-of-band and all operations are secured.

Non-authorized request

A client wishing to access a resource stored at URI_{host} initially sends a non-authorized request to the GW. A non-authorized request is transmitted unprotected, i.e., it is transmitted over an unprotected communication channel, hence 3rd parties can view it, even modify it. A non-authorized request includes the URI_{host} and a random number. Upon receiving a non-authorized request, a GW retrieves URI_{policy} and $SK_{acp,host}$ from the Access Table and generates a *token*. A token is a (public) variable unique among all sessions of that specific Thing. Then, the GW updates a "Connections Table" that contains tuples of the form [Connection Id, Expire, Parameters] where Connection Id an identifier for that connection (for example, client's IP address and port), Expires is the expiration time of the token, and parameters are the client's random number, URI_{host}, and URI_{policy}. Finally, the GW responds to the client with the URI_{policy} and the token. Figure 4 illustrates a non-authorized request. In this example, the client requests a resource stored at company1.iot. The GW generates a token, and responds to the client with ACP/Policy1 (found in the Access Table) and the token

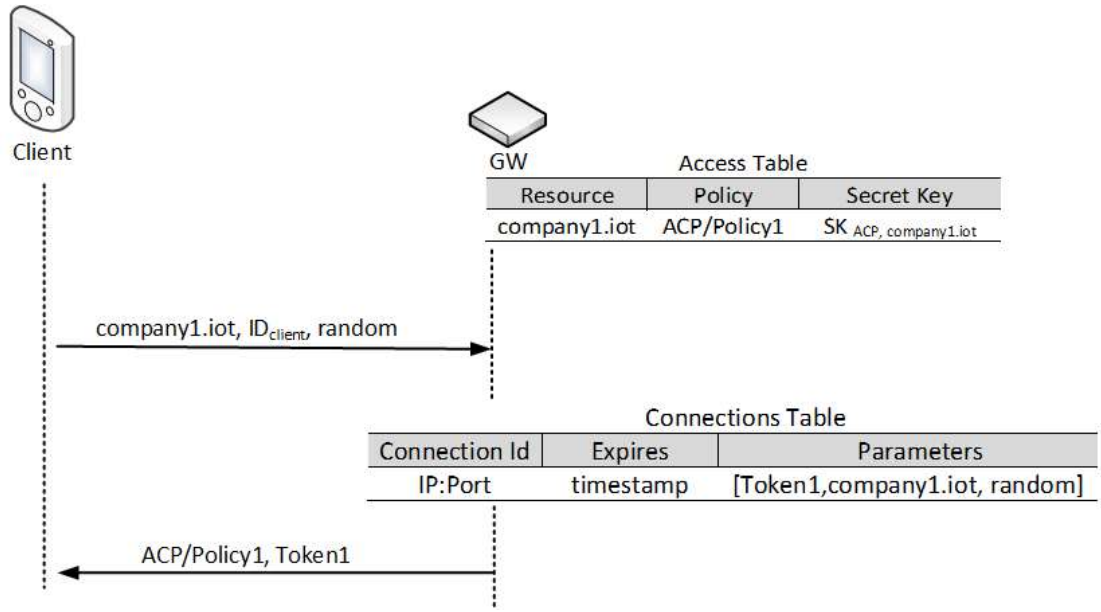


Figure 89: Non-authorized request

Client authentication and authorization

Upon receiving the GW's response, a client sends an authentication request to the appropriate ACP, which is found using the URI_{policy}. The semantics of this request, which is transmitted over a secure communication channel, are ACP specific. This request must contain the URI_{policy}, and the token received from the Thing, as well as, the URI_{host} and the client's random number included in the non-authorized request.

The ACP should authenticate the client, examine if he abides by the access control policy.If this is true, the ACP retrieves $SK_{acp,host}$ (this is the key generated during the setup phase) and uses it as key to an HMAC function with inputs URI_{policy}, ID_{client}, the token, and the client's random number. We refer to the output of this function as the ACP generated session key. Finally, the ACP securely transmits the ACP generated session key and the ID_{client} back to the client.

Final phase

Upon receiving the ACP generated session key and the ID_{client} back, the client sends the ID_{client} to the GW. Then, the GW uses $SK_{acp,host}$ and an HMAC to hash URI_{policy}, ID_{client}, the token, and the client's random number. The hash output is referred to as the GW generated session key. GW and ACP generated keys match (hence we will refer to them simply as session keys) if the following conditions hold:

- The GW really knows $SK_{acp,host}$ (hence it is authorized by the resource owner to offer the resource in question)
- No transmitted parameters have been modified.

Figure 5 illustrates an example of the client authentication and authorization, and finals phases. In this example, in which it is assumed that the phases illustrated in Figs. 3 and 4 have taken place, the client sends her identification in an ACP along with the URI_{host} she wants to access and the random number included in the non-authorized request, as well as the URI_{policy}, and the token she received from the GW. The ACP first performs client authentication, by examining the identification data and then client authorization, by examining if the client is authorized to access the resource in question. If both checks are correct, the ACP computes an HMAC with the appropriate input and sends the result back to the client.

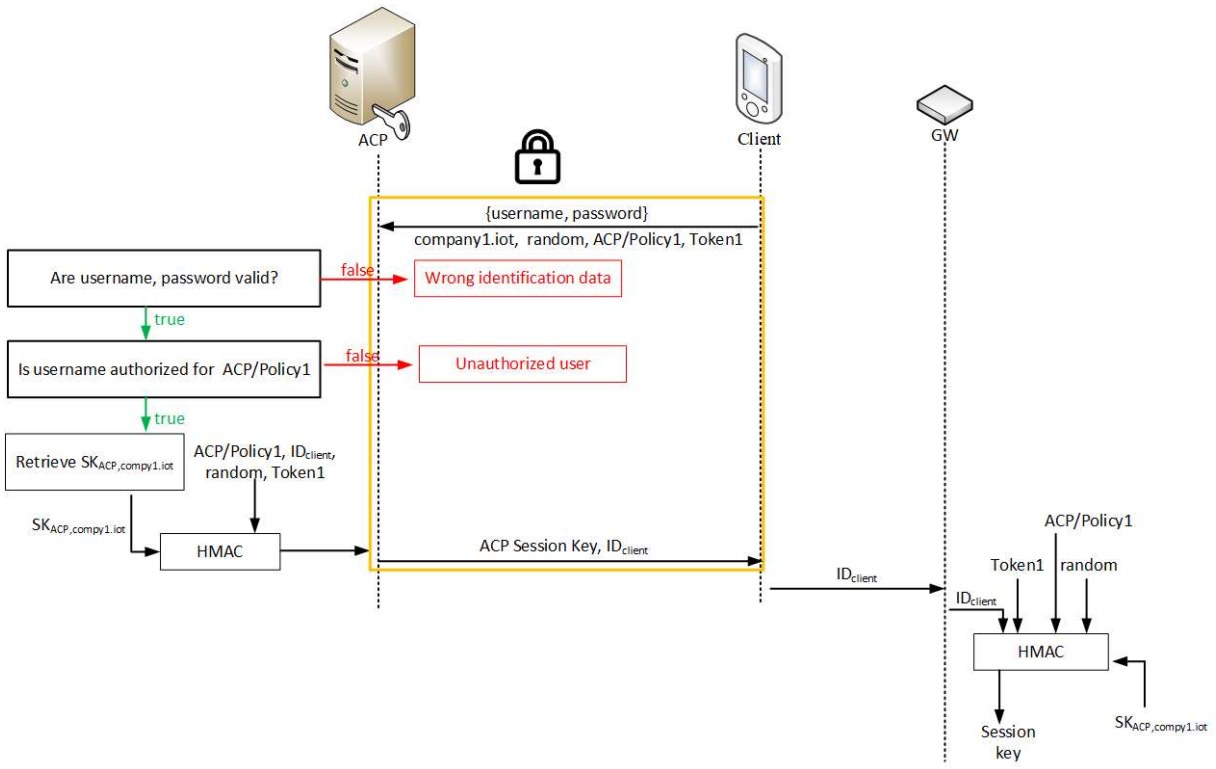


Figure 90: Client authentication and authorization, and final phases.

Security considerations

Providing that secret keys are protected, the proposed solution offers protection against tampering, eavesdroppers, as well as against man-in-the-middle attacks. Moreover, the key generation process guarantees that Things are authorized to host a resource, as well as that users are authorized to access it. For a thorough analysis of the security and privacy properties of our solution interested readers are referred to “N. Fotiou, et. al, “Access Control for the Internet of Things”, Proc. of the International Workshop on Secure Internet of Things 2016”.

Secret key breach

The setup phase described previously does not consider the case of a secret key breach. If a SKacp,host is revealed then all already deployed keys must be updated. For this reason, we propose the use of a counter per URIhost that should be maintained by the ACP. The counter will be used as an input to the hash function that generates the SKacp,host and it will be incremented every time this key has to be updated. Clients and Things do not have to be aware of this counter.

3.3.8.1 Integration of IoT framework

The following figure illustrates how ACHILLES has been integrated with the INTE-IoT framework.

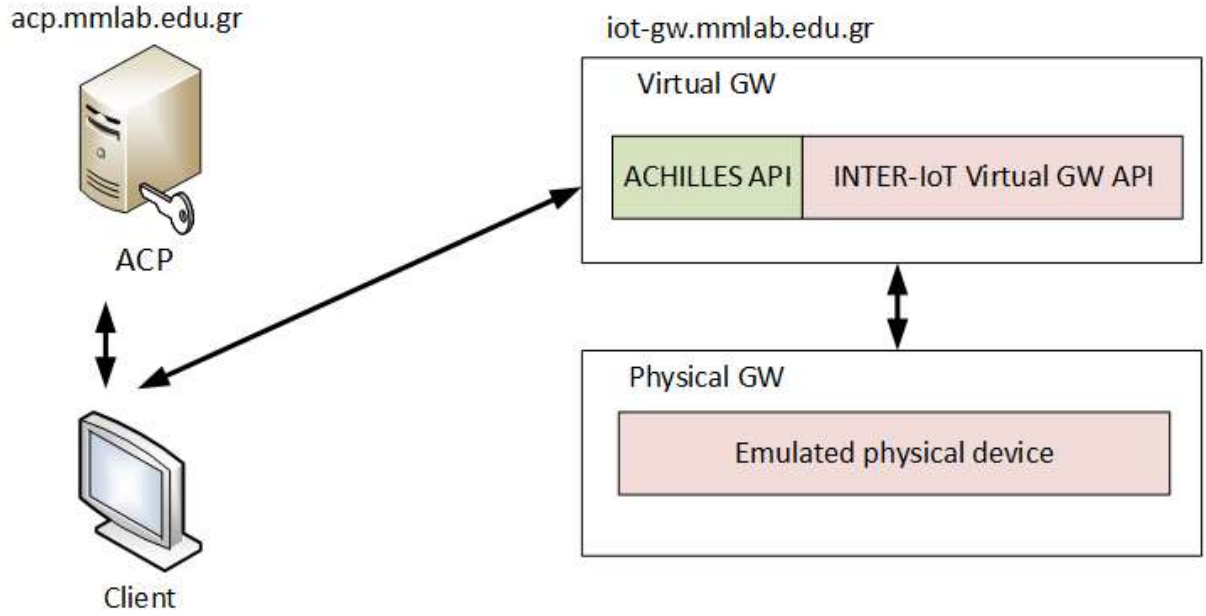


Figure 91: ACHILLES-INTER-IoT integration

An instance of the INTER-IoT GW has been installed at the location “iot-gw.mmlab.edu.gr”. The virtual part of the GW has been extended to include ACHILLES API (as it can be seen form the figure below).

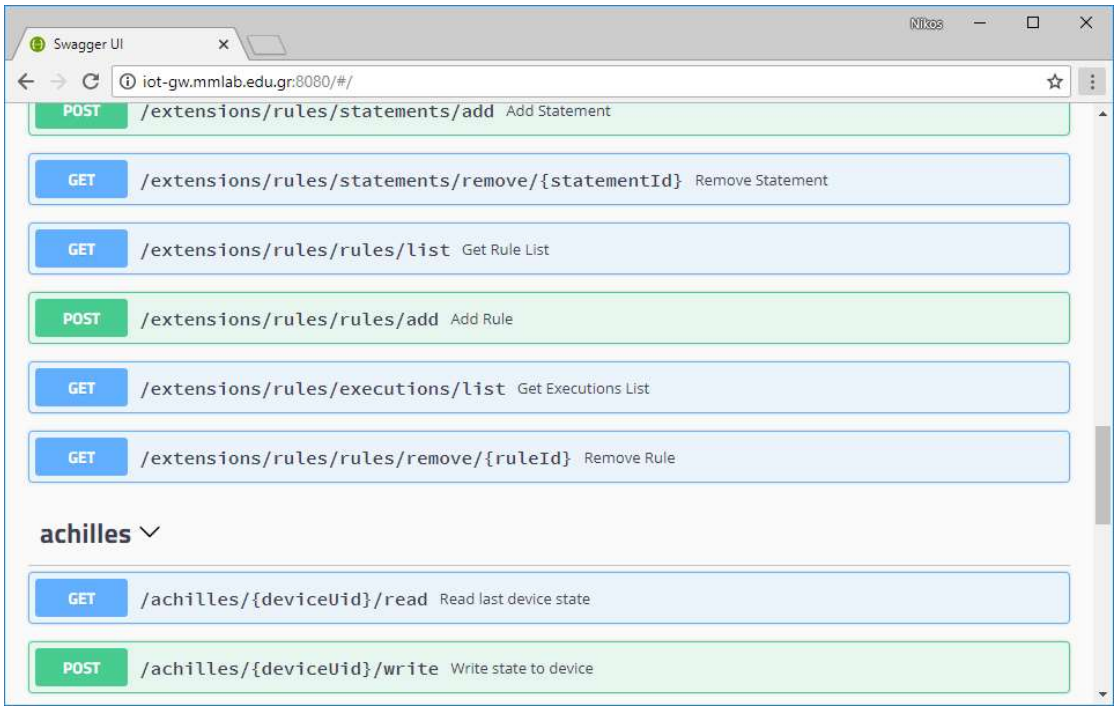


Figure 92: ACHILLES API as a component of the virtual GW

This extension implements ACHILLES functionality and configuration files, and it is able to perform read and write calls to an (unmodified) emulated physical device provided by the IoT framework.

Furthermore, a Java client (Figure 8) is able to communicate with the INTER-IoT gateway, as well as interact with an ACP located at the location “acp.mmlab.edu.gr”.

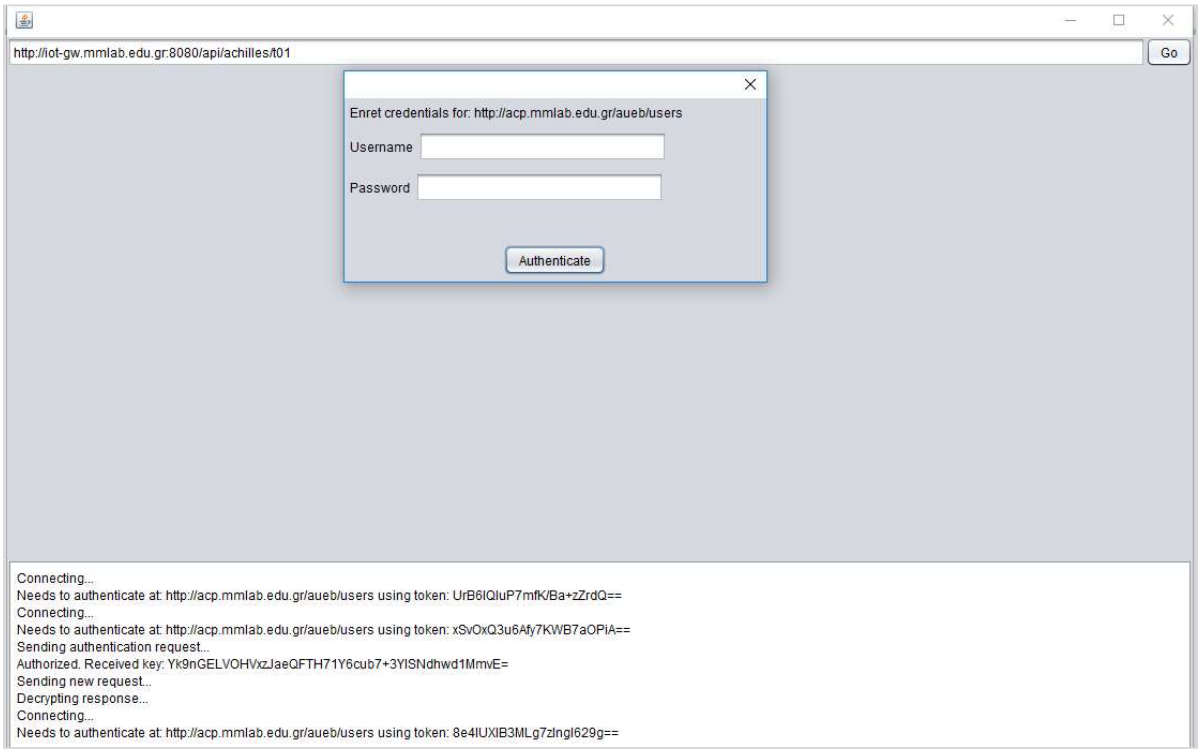


Figure 93: ACHILLES Client

3.3.8.2 Deliverables and version overview

The following table contains a deliverable list which need to be signed of before SAT testing commences.

ID	Description	Check
Documents		
1	Validation and Test reports of the IoT system components	
2	Validation and Test reports of the Pilot system components	
3		
Tools		
7	Wireshark	

Table 56: Deliverable checklist

The following table shows the software components and version of which the system release version 2.0 consists of.

ID	Description	Version	Check
IoT Physical Gateway			
1	Logging	VX.X.X	
2	Emulated device	VX.X.X	
IoT Virtual Gateway			
3	Virtual Gateway API		
ACHILLES			
4	ACP	V2.0.0	
5	ACHILLES Client	V2.0.0	
6	GW Module	V2.0.0	

Table 57: Component version overview.

3.3.8.3 Requirements, scenarios and use cases

The following table provides an overview of the relation between the requirements and the test(s) that validate their implementation.

ID	Description	Covered by
Architecture		
2	Scalability. Design	T1.1.1, T2.1.1
6	Efficiency of the processing of information	T1.1.1, T2.1.1
Communications		
14	Platform independent	T1.2.2, T2.2.2
15	Common IoT communication protocols must be supported.	T1.2.2, T2.2.2
Functionality		
11	Addressability and reachability	T1.1.1, T2.1.1
22	Unique identifier	T1.1.1, T2.1.1
API		
243	Gateway access API	T1.1.1, T2.1.1
Interoperability		
13	Extensibility	T2.1.2
Legality		
76	Interoperability between things from different administrative/management domains	T2.2.2
Performance		

72	Communication should be done using protocols that are efficient in terms of amount of exchanged information over message size	T1.2.2, T2.2.2
Security		
27	System security	T 1.2.1, T 2.2.1, T3.1.1, T3.2.1, T3.2.2
28	System privacy	T3.1.1, T3.2.1, T3.2.2
95	Robustness, resilience and availability	T3.1.1, T3.2.1, T3.2.2
98	Data provenance	T1.1.1, T2.1.1

Table 58: Requirements vs test mapping.

The following table provides an overview of the relation between the scenarios and the test(s) that validate their implementation.

ID	Scenario name	Covered by
1	IoT Data sharing	T1.1.1, T1.2.1, T1.2.2
2	B2B Services	T2.1.1, T2.2.1, T2.2.2
3	System under attack	T3.1.1, T3.2.1, T3.2.2

Table 59: Scenario vs. test mapping.

3.3.8.4 Test environment

Introduction

To test the functionality of the ACHILLES project in combination with the IoT framework a representative test system is needed. The test system needs to approach the “real world” as much as possible. The pilot setups must be recreated and proven. This chapter will describe this environment and the used hardware, software, tools and platforms.

Test environment

Our testing environment is composed of the following components:

- A virtual machine that includes the following INTER-IoT GW
  - An INTER-IoT virtual gateway that includes ACHILLES components (running at iot-gw.mmlab.edu.gr)
  - An INTER-IoT physical gateway, connected to an emulated device
- A Java-based client
- An ACP hosted at acp.mmlab.edu.gr

Test setups, tools, hooks and probes

TS_01 Test setup 1

This setup is used by all tests that involve a single ACP. During this setup the ACP is configured with username-password pairs. Moreover, the ACP generates and securely stores a Master Secret Key (MSK). In addition, access control policies are created and stored in ACPs, and a Uniform Resource Identifier (URI) for each policy is generated. The ACP uses a secure HMAC and the MSK to calculate a secret key as follows: $SK_{acp,GW} = HMAC_{MSK}(ID_{GW})$. The calculated SK is then configured to the GW. Finally, each CoAP client is configured with an ACP username and password as well as with the URI of the desired resources.

TT_01 Packet sniffer

In order to visualize the exchanged messages, we will use the Wireshark network sniffer.

TH_01 Test configurator

This hook is used for configuring the ACP with the appropriate parameters, as well as for injecting into the INTER-IoT GW the generated secret keys and proper configuration files. The configuration files contain entries related to the available CoAP resources.

TH_02 Session re-player

This hook is used for replaying requests from authorized users with or without modifications and it is used for testing the security properties of the system.

TP_01 GW Dumper

This probe will output all state related to ACHILLES maintained by the INTER-IoT GW. This includes configuration files, access control tables, as well as Token Tables. Test description

3.3.8.5 Test description

S1 - IoT data sharing

The objective of this scenario is to enable a resource owner to share measurement data with other authorized users. In this scenario resource owners have Things they own connected to an INTER-IoT GW. These Things perform various measurements. Measurements are grouped based on the Thing location and can be accessed in real-time using the appropriate resource URIs (e.g., http://iot-gw.mmlab.edu.gr:8080/t01). Resource owners define access control policies in the ACP (e.g., “Friends”, “Family”) and define in the GW the access control policy that protects each group of measurements (e.g., “t01 can be accessed by Friends”).

U1: New measurement group creation

The resource owner creates a new group of measurements and registers them in the GW, providing at the same time a pointer to the access control policy that protects them.

T1.1.1 New measurement group creation

ID	T1.1.1
Test	Registration of a new group of measurements
Type	System testing
Setup	Needs setup TS_01
Start	Access Table in GW is empty
Req.	[2],[6],[11],[22],[243],[98]
Input	Resource owner invokes the resource registration API call
Output	Access Table is updated
Logs	Folder “`T1_Output`”, prefix “`T1.1.1_achilles`” >
Outcome	Pass / Fail

U2: User request

A user is interested in a receiving a measurement protected under a specific access control policy. The user performs an initial request (an unauthorized request) to learn all information required for authorization. Then, it authenticates himself in the appropriate ACP and obtains an authorization token. The latter is used for performing an authorized request.

T1.2.1 Unauthorized request

ID	T1.2.1
Test	Request from an unauthorized user for a protected resource
Type	System Testing
Setup	Needs setup TS_01
Start	Access Table contains some entries
Req.	[27]
Input	A request from an unauthorized user
Output	• Check if the resource is included in the Access Table • Generate session key • Generate token • Respond to the user with the ACP URI and the token
Logs	Folder "`T1_Output`", prefix "`T1.2.1_achilles`" >
Outcome	Pass / Fail

T1.2.2 Authorized request

ID	T1.2.2
Test	Request from an authorized user for a protected resource
Type	System Testing
Setup	Needs setup TS_01
Start	Access Table and Token Table contains some entries
Req.	[14],[15],[72],[76]
Input	A request from an authorized user
Output	• Check if the resource is included in the Access Table • Check if the Token is included in the Token Table and it is still valid • Perform a read/write operation to the appropriate device • Encrypt the response and send it back to the user
Logs	Folder "`T1_Output`", prefix "`T1.2.2_achilles`" >
Outcome	Pass / Fail

S2 - B2B services

The objective of this scenario is to enable protected resources for multiple groups of authorized users belonging to diverse administrative domains. In this a scenario, a resource owner owns actuators connected to an INTER-IoT GW. These actuators can accept. Various stakeholders define access control policies in their corresponding ACP (e.g., “Employees”, “Managers”). Moreover, the resource owner defines in the GW the access control policies that protect each operation (e.g., “switch1 can be turned on by the “Employees” of the company that has business relationships with ACP A, or the “Employees” of the company that has business relationships with ACP B).

For this scenario, a replicated instance of our ACP is running at the location acp2.mmlab.edu.gr

U1: New operation creation and management

The resource owner defines an operation that can be performed on an actuator and provides pointers to the policies that protect this operation. Moreover, later on, the resource owner can modify the list of the pointers to policies by adding or removing a pointer.

T2.1.1 New operation registration

ID	T2.1.1
Test	Registration of a new resource protected by multiple policies
Type	System testing
Setup	Needs setup TS_02
Start	Access Table in GW is empty
Req.	[2],[6],[11],[22],[243],[98]
Input	Resource owner invokes the resource registration API call
Output	Access Table is updated
Logs	Folder “`T1_Output`”, prefix “`T2.1.1_achilles`” >
Outcome	Pass / Fail

T2.1.2 List of policies modification

ID	T2.1.2
Test	Add or remove a pointer to an access control policy
Type	System testing
Setup	Needs setup TS_02
Start	Access Table in GW has some entries
Req.	[13]
Input	Resource owner invokes the resource registration API call
Output	Access Table is modified
Logs	Folder “`T1_Output`”, prefix “`T2.1.2_achilles`” >
Outcome	Pass / Fail

U2: User request

A user is interested in triggering an actuator protected by some access control policies. The user performs an initial request (an unauthorized request) to learn all information required for authorization. Then it authenticates himself in the appropriate ACP and obtains an authorization token. The latter is used for performing an authorized request.

T2.2.1 Unauthorized request

ID	T2.2.1
Test	Request from an unauthorized user for a protected actuator
Type	System Testing
Setup	Needs setup TS_02
Start	Access Table contains some entries
Req.	[27]
Input	A request from an unauthorized user
Output	• Check if the resource is included in the Access Table • Generate session key • Generate token • Respond to the user with the ACP URIs and the token
Logs	Folder "`T1_Output`", prefix "`T2.2.1_achilles`" >
Outcome	Pass / Fail

T2.2.2 Authorized request

ID	T2.2.2
Test	Request from an authorized user for a protected resource
Type	System Testing
Setup	Needs setup TS_02
Start	Access Table and Token Table contains some entries
Req.	[14],[15],[72],[76]
Input	A request from an authorized user
Output	• Check if the resource is included in the Access Table • Check if the Token is included in the Token Table and it is still valid • Perform a read/write operation to the appropriate device • Encrypt the response and send it back to the user
Logs	Folder "`T1_Output`", prefix "`T2.2.1_achilles`" >
Outcome	Pass / Fail

S3 - System under attack

The objective of this scenario is to evaluate the security of the integrated platform in the presence of malicious users.

U1-New sessions

An attacker is able to capture and record successful sessions. He then replays the messages in order to gain access to a protected resource.

T3.1.1 Replay attack

ID	T3.1.1
Test	Emulate an attacker that repeats captured sessions
Type	Security Test
Setup	Needs setup TS_01 or TS_02 and Test hook 1
Start	Access Table and Token Table contains some entries
Req.	[27],[28],[95]
Input	A request that appears from an authorized user
Output	• Check if the resource is included in the Access Table • Check if the Token is included in the Token Table and it is still valid • Reply with an error
Logs	Folder "`T1_Output`", prefix "`T3.1.1_achilles`" >>
Outcome	Pass / Fail

U2-Tampering with existing sessions

An attacker is able to intercept the communication between an authorized user and a Thing. His goal is to modify the transmitted packets in way that will give him access to protected resources.

T3.2.1 Packet modification attack

ID	T3.2.1
Test	Emulate an attacker that modifies transmitted packets
Type	Security Test
Setup	Needs setup TS_01 or TS_02 and Test hook 1
Start	Access Table and Token Table contains some entries
Req.	[27],[28],[95]
Input	A CoAP request that appears to be from an authorized user
Output	• Check if the resource is included in the Access Table • Check if the Token is included in the Token Table and it is still valid • Reply with an error
Logs	Folder "`T1_Output`", prefix "`T3.2.1_achilles`" >>
Outcome	Pass / Fail

T3.2.2 Man-in-the-middle attack

ID	T3.2.2
Test	Emulate an attackers that perform man-in-the-middle attack
Type	Security Test
Setup	Needs setup TS_01 or TS_02 and Test hook 1
Start	Access Table and Token Table contains some entries
Input	A CoAP request that appears to be from an authorized user
Output	• Check if the resource is included in the Access Table

	• Check if the Token is included in the Token Table and it is still valid • Reply with an error
Logs	Folder "`T1_Output`", prefix "`T3.2.2_achilles`" >>
Outcome	Pass / Fail

3.3.8.6 Test outcome overview

The following table will provide an overview of the test result of all the performed tests in this SAT.

Test	Description	Outcome
T1.1.1	New measurement group creation	Pass / Fail
T1.2.1	Unauthorized request	Pass / Fail
T1.2.2	Authorized request	Pass / Fail
T2.1.1	New operation registration	Pass / Fail
T2.1.2	List of policies modification	Pass / Fail
T2.2.1	Unauthorized request	Pass / Fail
T2.2.2	Authorized request	Pass / Fail
T3.1.1	Replay attack	Pass / Fail
T3.2.1	Packet modification attack	Pass / Fail
T3.2.2	Man-in-the-middle attack	Pass / Fail
SAT Outcome		**Pass / Fail**

Table 60: Test outcome overview

3.3.8.7 Integration ethics and security

Introduction

This section addresses recommendations and proposed solutions to the ethical and security points of the integration.

ACHILLES

The site acceptance plan presented in this document does not raise any ethical issue, since the data that will be used during the tests will be artificially created and will not affect any real entity. Moreover, all experiments will be conducted in a closed system that does not interact with the external world, hence no concerns should be raised. When it comes to the actual deployment of ACHILLES, various risks should be taken into consideration. ACHILLES ACPs may have access to sensitive user information, including user names, passwords, and access control policies. Moreover, ACHILLES ACPs may have access to the "seed" used by the Things to generate session secret keys. For these reasons, a security incident handling plan should be considered.

3.3.9 Third Party: Inter-HINC

The conceptual design of INTER-HINC is described in INTER-HINC D2 and in related papers[21]. It will be updated in the final INTER-HINC D3. Figure 94 describes key components, without internal details, for SAT plan. Three parts of INTER-HINC:

- Resource Slice Interoperability Hub (rsiHub): includes services, software artifacts and algorithms to ensure resource slice interoperability.
- Pizza-CLI: is the client CLI including various features for the user to perform tasks for interoperability with resource slices.
- External ResourceProviders: including those from third parties, those integrating INTER-IoT existing components and those newly developed for the INTER-IoT project.

The main services within the architecture are:

- rsiHub LocalService: to interface to IoT, network function and cloud providers.
- rsiHub GlobalService: for the application and other middleware to control IoT devices, networks and services and acquire IoT data, for checking and finding bridges for interoperability and presenting workflows of configuration and provisioning of resources for interoperability, slice management by provisioning and configuring slices.

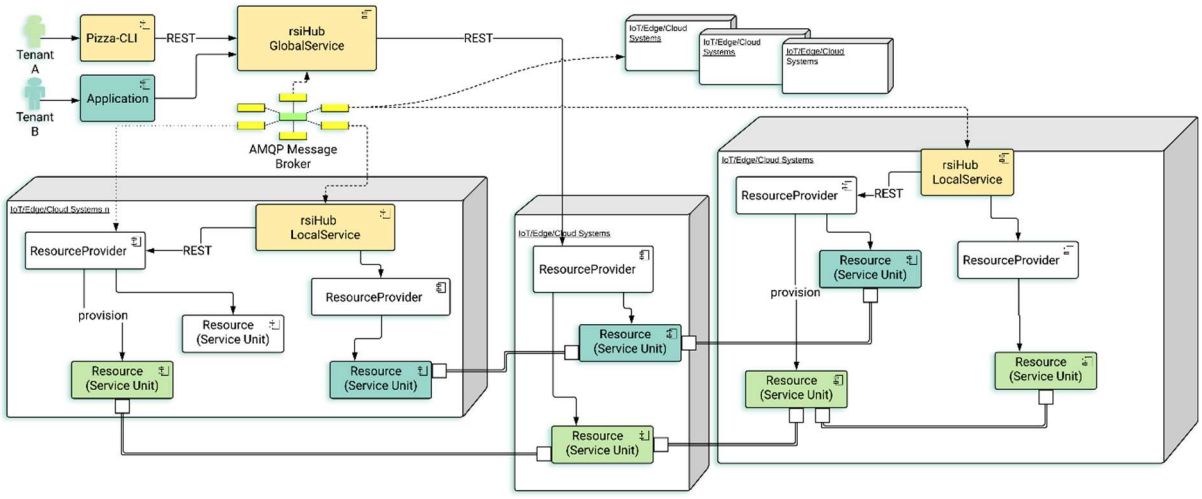


Figure 94: Component View of INTER-HINC

All the above-mentioned services provide standard REST APIs for any clients to use INTER-HINC services and for applications to program calls to INTER-HINC services.

The current rsiHub deployment allows a single deployment for rsiHub with various microservices and multiple rsiLocalService. rsiLocalService deployments are highly dependent and integrated with the number of IoT/Cloud providers.

[21] Such as: Hong Linh Truong: *Towards a Resource Slice Interoperability Hub for IoT*. IC2E 2018: 310-316; Hong-Linh Truong, Lingfan Gao, Michael Hammerer, *Service Architectures and Dynamic Solutions for Interoperability of IoT, Network Functions and Cloud Resources*, 12th European Conference on Software Architecture, September 24-28, 2018, Madrid, Spain

The source code, detailed documents, deployment, tests, etc. are available in GitHub under two git repositories: rsiHub and IoTCloudSamples

- rsiHub: https://github.com/sincconcept/HINC. The main services are part of rsiHub, and due to their importance, by default they are deployed in cloud environments, whereas the additional Local Service(s) can be deployed elsewhere, depending from the individual requirements of resource providers.
- IoTCloudSamples: https://github.com/rdsea/IoTCloudSamples. Several Services and their providers are provided. Some are integrated wiith INTER-IoT Services (e.g., Middleware). Some are emulating services who sources are not available. Furthermore, some units have been developed for new scenarios. We will explain them in Section 4.

Due to the richness and complexity of various services in rsiHub and IoTCloudSamples. The test is therefore focusing on testing these components.

3.3.9.1 Integration of IoT framework

The integration is seen from two aspects: various Providers have been provided for various purposes or for integration with other INTER-IoT, or for emulation. We outline the main services we provide here but the complete list should be referred to our Git repositories.

3.3.9.1.1 IoT, Network Functions and Cloud Providers

As part of Inter-HINC, many services are deployed in order to run specific providers. Some Providers can be used in the production/development for INTER-IoT. However, some are just emulators for certain important services that we cannot access the source/software. Below, listed are the most important providers that can be used by INTER-IoT. Therefore, not only we test them within the INTER-HINC but also test them separately.

Node-RED Datatransformer Provider

INTER-HINC provides a provider for data transformation using Node-RED (source code: nodered-datatransformer-provider). This service provider is not like a single Node-RED installed statically: the provider offers multi-tenant, separated Node-RED instances for different stakeholders on the fly: any stakeholders (e.g. port authority, crane companies, vessel company) can dynamically make a request to INTER-HINC to provision a Node-RED instance and then deploy workflows (for analytics, data transformation, etc.) on demand. The provider is developed atop Kubernetes that enables scalable data transformation and analytics in INTER-IoT. This provider can wrap specific Node-RED components from INTER-IoT. We have test it with Google cloud platform and minukube[22] in a medium server size. Currently, this provider is being integrated with the INTER-IoT Framework.

MQTT Provider

INTER-HINC provides an MQTT provider (similar to the case of Node-RED) for multi stakeholders and multitenant scenarios. The MQTT broker can be instantiated on the fly (source code: mosquitt-mqtt-provider).This provider can be used to provide MQTT instances on-demand. It is especially useful for INTER-IoT to support multi-tenant in sharing and exchanging information among various users based on different platforms. The Provider offers MQTT atop Kubernetes container platforms, tested with Google cloud platform and minikube.

[22] https://kubernetes.io/docs/setup/minikube/

Generic Lightweight IoT Provider

There are many cases when an IoT function (e.g. Broker, Gateway, Firewall, etc.) has to be executed in a lightweight machine. This means, the machine can only run basic operating system features, VMs, or dockers, but cannot have a complex distributed system, like Kubernetes. For such a case, a function is wrapped into a unit, which will be run as a process within the machine. The Generic Lightweight IoT Provider will support this kind of units in a generic way. Currently the integration with INTER-IoT Gateway is ongoing.

BigQuery Provider

This service providers offer service instances wrapping Google BigQuery; it enables the requirements for setting up the storage service using the Google BigQuery.

Sensor Provider

This Provider offers to run different types of sensors for emulating real sensors. Emulating sensors reflect real sensor based on realistic dataset and they are needed to test INTER-IoT scenarios as the access to many types of data are not given. Emulating sensors mostly read real dataset and replace the data

Video Camera Provider

We assume that we could also access cameras in INTER-IoT scenarios. This service enables a provider through which one can obtain real camera video. In the context of SAT, there is no such real camera from INTER-IoT available. Therefore, we use the service in the emulation mode to obtain real camera from other places (e.g., in a city).

3.3.9.1.2 Connectors to other INTER-IoT Components

For integration with other services of INTER-IoT, we are also building connectors for receiving real data from these services. Note that these connectors are still in testing as the integration with various real INTER-IoT services is still in progress.

- INTER-IoT Middleware for receiving port weather information & truck access gateways
- INTER-IoT Middleware for controlling lights

Furthermore, it is also planned that other INTER-IoT components will use INTER-HINC services. For example, we are currently working on how the Front-end in the INTER-IoT Framework would utilize the Node-RED datatransformer provider to support multi-tenant Node-RED and flows processing.

3.3.9.1.3 Port Application Services

INTER-HINC also integrates with other services emulating applications. Note that this is currently doing so tests are also developed:

- Port Control Services: to emulate a feature of port control service to share information for vessels in the emergency situation
- Alarm Service: emulates possible alarms that should be used to trigger actions for vessels, trucks, etc.

3.3.9.1.4 Resource Slices

A resource slice reflects an integration view for INTER-IoT. It shows how different components related to INTER-IoT and Interoperability can be created by INTER-HINC.

Various scenarios are detailed in papers[23],[24] and rsiHub GitHub[25]. Here we describe two main example resource slices for testing. The list of resource slices will be extended and revised during the course of the testing and implementation. Note that the number of resource slices and resource slices are not fixed. It is up to the developer to use INTER-HINC to create suitable resource slices integrating various components and services for different scenarios.

3.3.9.1.5 Accessing Sensor Data in seaport

This resource slice can be used to show middleware interoperability, protocol interoperability and data interoperability with different solutions.

Example of resource slice: a tenant in the seaport wants to access sensor data in the seaport with the condition of no-sharing middleware. The resource slice will be created, including:

- Sensors: for sensor data.
- MQTT brokers for exchanging data

We then have different situations for resource slice reconfiguration

- First situation: the consumer wants to process data from the broker using a separate workflow engine within the seaport. The slice is reconfigured with a new Node-RED instance and the consumer adds a workflow into Node-RED.
- Second situation: Another consumer wants to access the sensor data from the broker but finds that the data is in CSV, thus the consumer wants to deploy a resource to transform CSV data to JSON. Two possible solutions:
  - a new component is deployed that takes data from MQTT and transforms the CSV to the JSON. This component is based on (https://github.com/rdsea/IoTCloudSamples/tree/master/IoTCloudUnits/csvToJson
  - a new Node-RED instance is created and a workflow for data transformation is pushed into the instance.
- Third situation
  - Similar to the second situation but the consumer is outside the seaport. Thus, cloud services are used and Network Function is also enabled.

Data Exchange and Control in Emergency situation

We assume there are alarms occurring in a seaport. The alarms are propagated through an MQTT broker. Usually, there are some analytics applications listening the alarms queues to react to the alarms. One of such alarms analytics programs finds alarms related to terminals in the port. It queries a PortControlService (PCS) to obtain the list of vessels approaching the port. We have the resource slice creating dynamically:

- MQTT broker for alarms
- A REST PCS emulating the Port Control. The PCS has APIs for querying vessels and for updating vessels positions. PCS has the back-end database as MongoDB.
- A set of vessel emulators (python/nodejs) emulate the movement of vessels. A vessels emulator subscribes information from its providers via a queue.
- A set of vessel service providers. Each providers accept a different format of data (JSON/CSV with different structures) and use different protocols (MQTT, AMQP and REST).

[23] Hong-Linh Truong, Lingfan Gao, and Michael Hammerer. 2018. Service Architectures and Dynamic Solutions for Interoperability of IoT, Network Functions and Cloud Resources. 12th European Conference on Software Architecture, September 24-28, 2018, Madrid, Spain.

[24] Hong Linh Truong: Towards a Resource Slice Interoperability Hub for IoT. IC2E 2018: 310-316

[25] https://github.com/SINCConcept/HINC/tree/master/scenarios

The above-mentioned resource slice can be extended to cover:

- Cranes and trucks are similar vessels with cranes/trucks and their providers
- Vessels/trucks/cranes/cameras have their GPS positions so that geohash can be used to query them.

Note that while our slices show various integration with many components, many components are not accessible thus, we have to emulate them, such as PortControlService and Vessels.

3.3.9.2 Deliverables and version overview

The tests presented in this document are performed before the INTER-HINC D3. Therefore, there are no documents that one needs to be signed. Table 3 shows the tools.

ID	Description	Check
Documents		
1	rsiHub online document: https://github.com/sincconcept/HINC	
2	IoTCloudSamples online document: https://github.com/rdsea/IoTCloudSamples	
Hardware/Infrastructure		
3	Google Cloud (BigQuery, Compute, Kubernetes)	
4	Raspberry PI/Typical Computer	
5	MongoDB (local installation, mlab.com instance, or MongoDB Atlas(https://www.mongodb.com/cloud/atlas))	
6	Docker container	
7	MQTT (mosquito, https://mosquitto.org/)	
8	RabbitMQ (https://www.rabbitmq.com/)	
Tools		
9	Postman	
10	NodeJS	
11	Python 3 and 2	

Table 61:Tool checklist

The following table shows the software components and version of INTER-HINC used in the tests. Note that there are many other components in the Git that we do not mention.

ID	Description	Version	Check
IoTCloudSamples			
1	GenericLightweightIoTProvider	0.1.0	
2	nodered-datatransformer-provider	0.1.0	
3	mosquitt-mqtt-provider	0.1.0	
4	bigQueryProvider	0.1.0	
rsiHub			
5	rsiGlobalService	0.1.0	
6	rsiLocalService	0.1.0	
7	Pizza-CLI	0.1.0	
8	Port control vessel provider plugin	0.1.0	
9	Alarm client provider plugin	0.1.0	
10	Valencia sensor plugin	0.1.0	

Table 62: Component version overview

3.3.9.3 Requirements, scenarios and use cases

The following table provides an overview of the relation between the requirements and the test(s) that validate their implementation.

ID	Description	Covered by
2	Scalability. Design	All tests
3	Ability to be enlarged to accommodate growth. Computing resources	All tests
13	Extensibility	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
14	Platform independency	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
15	Support of common IoT communication protocols	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
31	Tools / libraries to support design	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
39	Heterogeneous gateway	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
43	Service discoverability	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
47	API for third-party developers	All tests
52	API REST	All tests
54	High responsiveness	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1, T2.1.1, T2.1.2, T2.1.3
57	Monitoring and Self-Awareness of the system	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
81	Quality of Service	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1, T2.1.1, T2.1.2, T2.1.3
87	Online documentation in INTER-IoT new tools and services	All code/tests
95	Robustness, resilience and availability	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1 T1.5.1, T1.5.2, T1.5.3
108	Open Source	All code/tests
123	Use of standards	All code/tests (REST, AMQP, MQTT)
125	Adaptability	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
132	Portability	All code/tests
159	Development support for systematic IoT platforms integration/interconnection	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1, T2.1.1, T2.1.2, T2.1.3
198	Capacity to achieve a heterogeneous computing platform environment	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
199	IDEs and APIs for rapid new applications development	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1, T2.1.1, T2.1.2, T2.1.3
226	API for network services	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
231	Network function virtualization	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
232	Fault tolerance	T1.1.1, T1.1.2, T1.1.3, T1.2.1,

		T1.3.1, T1.4.1 T1.5.1, T1.5.2, T1.5.3
234	Provide connectors to middleware standards	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
236	Support of main Internet of Things platforms	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
237	API Middleware for interoperability between different platforms	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1, T2.1.1, T2.1.2, T2.1.3
244	Gateway virtualization	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
248	Create new services to access different platforms	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
251	Ability of IoT platforms to coordinate with emergency systems	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1, T3.1.1, T3.1.2
266	API allows resources/capabilities discovery	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
282	Map publish/subscription between platforms	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
283	Manage a sensor or actuator	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1

Table 63: Requirements vs. test mapping

The following table provides an overview of the relation between the scenarios and the test(s) in INTER-HINC. Note that the integration is still on the progress, while the scenarios are covered, the test description is given.

ID	Scenario name	Covered by
2	IoT support for transport planning and execution	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
6	Dynamic lighting in the port	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
9	Accident at the port area	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1, T3.1.1, T3.1.2
13	IoT interoperability for Vessel Arrivals	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1, T3.1.1, T3.1.2
19	Transport on truck breaks down or is hijacked	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
20	Damage or problems to the container during shipment	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
29	Reliable control of robotic cranes and trucks in port terminals	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
30	IoT access control, traffic and operational assistance	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1
32	Third party developer using INTER-FW to access data from two different platforms	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1, T2.1.1, T2.1.2, T2.1.3
33	Heterogeneous Platforms Methodology-driven Integration	T1.1.1, T1.1.2, T1.1.3, T1.2.1, T1.3.1, T1.4.1, T2.1.1, T2.1.2, T2.1.3

Table 64: Scenario vs test mapping

3.3.9.4 Test environment

Introduction

This chapter describes the test environments based on our current development and infrastructure access that we have.

Test environment and deployment models

From Figure 94, the test environment includes 3 parts:

- The global deployment of rsiHub: it includes the rsiHub Global Service and the message broker for rsiHub.
- Various local resource management services and corresponding providers
- The Client emulating the users which run tests

Note that “global” or “local” here are from the view of the design. Local deployments for Providers and rsiHubLocalService are actually dependent on the Providers which are available in the cloud as well.

Configuration	rsiHub Global		rsiHubLocal Service	Providers
Test environment	one rsiHubGlobal Service in Google Cloud Platform	One RabbitMQ from http://cloudamqp.com instance with	One rsiHub Local Management service in Google Cloud Platform (different data centers) One rishHubLocalManagementService in TU Wien	Providers: in Google Cloud Platform (different data centers), TU Wien

Table 65: A basic configuration of the tests

With respect specific configurations of machines, we have:

- A configuration of a Local Server as an Edge Server: 4 CPUs, 2 cores per CPU, Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz, 8GB memory, running with Ubuntu 16.04.4 LTS, docker, VirtualBox, and minikube.
- A configuration of cloud edge server (for using the production cloud to install edge servers): using machine type as g1-small (1 vCPU, 1.7 GB memory) in different zones
- A configuration for high-end edge servers using Kubernetes: includes 3 nodes with Google Kubernetes in GCP. Each node is with machine type as n1-standard-1 (1 vCPU, 3.75 GB memory) and all are in us-central1-a
- A configuration for cloud brokers: a RabbitMQT instance is from cloudamqp.com. The instance is a shared RabbitMQ, for development with max 100 queues, maximum 10 000 queued messages, maximum 1M messages/month, and max 20 concurrent connections.
- A configuration for MongoDB: from https://www.mongodb.com/cloud/atlas, is a free 3-node replica set in Amazon, with shared RAM and 512 MB storage.

All of these configurations are based on production environments but they are limited capabilities to serve for the tests under minimum conditions (and partially avoid that a larger configuration incurs a high cost).

Test tools, hooks and probes

We have two types of tests: basic tests and scenarios. Test scripts are available in the GitHub of corresponding services in INTER-HINC project, including:

- rsiHub for resource slice creation, management, monitoring, interoperability recommendation and management
- IoTCloudSamples: for various providers for interoperability

The following table shows the components and links where the tests scripts should be accessed.

Repositories	Services	Links
IoTCloudSamples	mqtt-provider	https://github.com/rdsea/IoTCloudSamples
	Nodered-datatransformer-provider	
	MQTT-Provider	
rsiHUB	GlobalManagementService	https://github.com/SINCConcept/HINC/
	LocalManagementService	
	ArtifactRepositoryService	

Table 66: Services and source code. Test scripts are within the Git repositories.

We use 3 types of tools for testing:

- Easy-to-use testing tool for the user: we use Postman[26], which is a very popular tool for testing web services. This tool allows various tests and the user can download it. Furthermore, it allows to sync and share test scripts and requests through the cloud. With this tool, the user can easy test INTER-HINC services in any place.
- Complex testing scripts: we develop testing scripts to test different scenarios. The scripts are mainly written in Python and Javascript using well-known libraries. Therefore, the tests can be executed easily in various environments.
- Behavior driven development (BDD) and Test Driven Development (TDD): we use Chai[27] to develop BDD/TDD tests for certain services.

In general, we provide various such scripts that can be used within the above-mentioned tools.

TS_01 Test setup

The test setup is based on configurations described above. For cloud services productions that are used for running rsHub or IoT/network functions/cloud providers, we registered free tie services or pay services/compute instances and deploy software in corresponding services. Software installation and configuration are documented in the source repository.

TT_01 Test tool Postman

The detail of Postman can be accessed from https://www.getpostman.com/. The setup of this tool is quite straightforward. Using this tool, one can add test collections and modify the URL of services accordingly based on the deployment. For example, Figure 95 presents a snapshot of Postman tool. With this tool, we can setup test scripts and run tests for various

[26] https://www.getpostman.com/
[27] http://www.chaijs.com/

services. Using this tool, we allow the user to modify test constraints: e.g. max time, so that they see the results suitable for their environment.

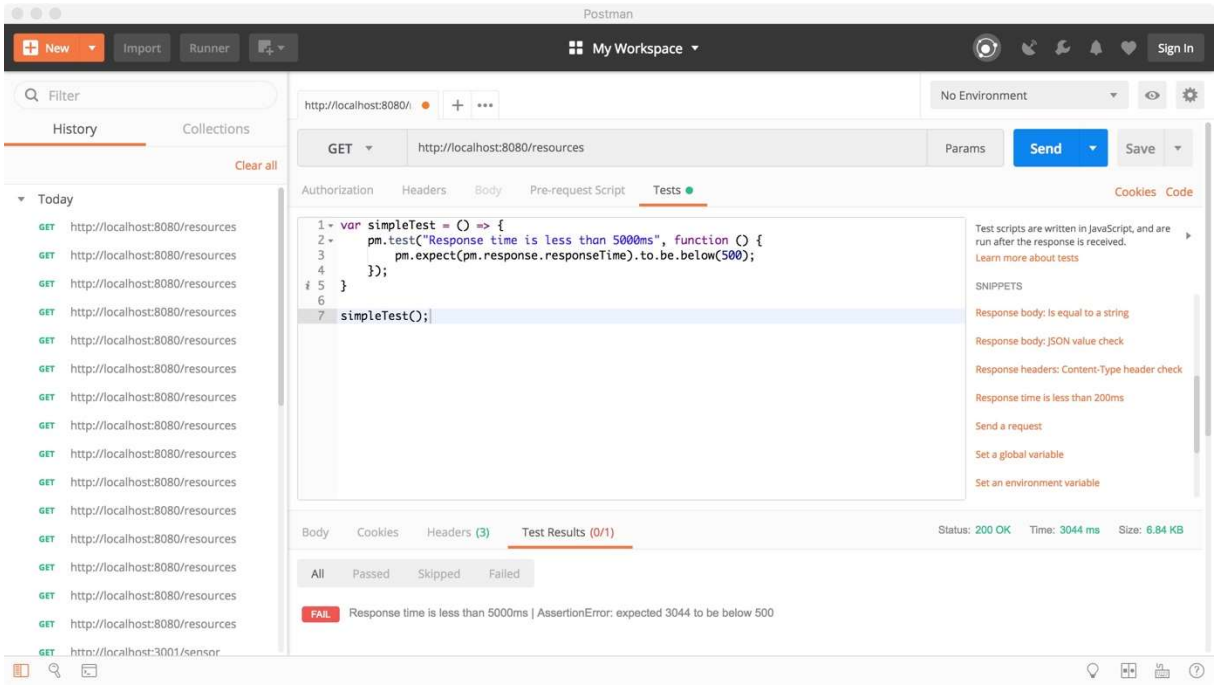


Figure 95: Snapshot of Postman tool for testing INTER-HINC

Figure 96 shows the Postman tool with test script to check the response time and if the information returned (for a nodered instance) has an URL.

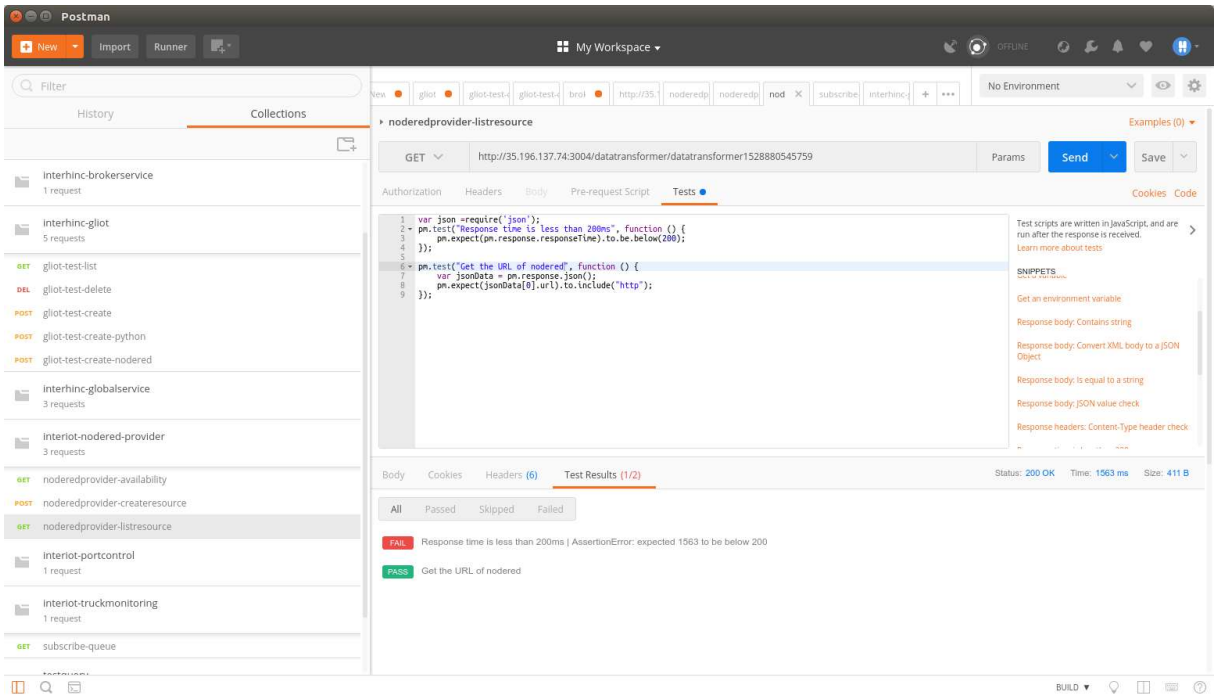


Figure 96: Example of setting up Postman for testing

TT_02 Test tool Python and Javascript

For this tool, one needs to setup the right environment and run the script. Most of the cases:

- Python: we use both Python2 and Python3. Related packages should be installed using pip
- Javascript: we use NodeJS (https://nodejs.org/en/). For tests, npm is used to download required packages described in the package.json

The following script presents an example of a python-based test that perform a test which is a part of three steps for a simple resource slice: (1) create a Node-RED resource, (2)upload a flow to the Node-RED resource, (3) remove the flow and (4) remove the resource. The example script just shows the upload and removal of flows into a Node-RED. A flow here can be used for IoT data processing or transformation.

```
parser = argparse.ArgumentParser()

parser.add_argument('--nodered_provider', help='URL of the Data
Transformer Provider')

parser.add_argument('--workflow_file',help='workflow_file')


args = parser.parse_args()

post_workflow_data =json.load(open(args.workflow_file))

headers = {'content-type': 'application/json'}

def test_upload_workflow(nodered_url):


workflow_post_response=requests.post(nodered_url+"/flows",data=jso
n.dumps(post_workflow_data),headers=headers)

    print(workflow_post_response.text)

    if (workflow_post_response.text):

        json_workflow_response                                    =
json.loads(workflow_post_response.text)

        workflow_id=json_workflow_response['id']


delete_workflow_response=requests.delete(nodered_url+"/flow/"+work
flow_id)

        print(delete_workflow_response)


test_upload_workflow(args.nodered_provider)
```

TT_03 Test tool Chai

We also use chai for some BDD and TDD. Installation of Chai is available at: http://www.chaijs.com/ and https://pypi.org/project/chai/

For BDD and TDD, for example, the following script tests if configuration for a real deployment is corrected or not.

```
var expect = require('chai').expect;

import  deployTemplate from '../configTemplates/deployTemplate';

import GLIoTFunction from '../data/models/gliotfunction';

var gliotFunctions = JSON.parse(JSON.stringify(deployTemplate));

expect(gliotFunctions.functions).to.have.lengthOf(3);

expect(gliotFunctions.functions[0].functionname).to.be.a('string')
;
```

3.3.9.5 Test description

Scenario Deploying Resource Slices in Edge Servers for Interoperability

In this scenario, resource slices including messaging brokers (MQTT/Mosquitto), workflow engine (Node-RED), data transformer, network functions (firewalls and deep packet inspection tools), etc. need to be deployed in edge servers, e.g., in the seaport for interoperability goals (e.g., protocol interoperability, data interoperability, IoT data platform interoperability). The resource slices can be deployed in a constrained edge server (resources are running as shared services for many users) or high-end edge server.

Use case create/list/delete a resource in an edge server

T1.1.1 Create, list and delete a resource in a light weighted edge server

ID	T1.1.1
Test	Test basic operations for resources in the edge server
Type	online response time, live environment, error handling
Setup	The GenericIoTservice is deployed in a virtual machine in Google Cloud. The virtual machine is g1-small (1 vCPU, 1.7 GB memory) in us-east1-b. Python script is running from another machine at TU Wien.
Start	Select Node-RED as the type of the resource for the test and run the script
Req.	
Input	a test script with a sequence of three service calls: create a node-red resource, get information about the newly created resource and removed the resource.
Output	Success of tasks and performance values
Logs	Summarized measurements are stored in GitHub. Logs are not stored as the tests can be easily rerun.
Outcome	Pass / Fail

T1.1.2 Create, list and delete a Node-RED resource in a high-end edge server

ID	T1.1.2
Test	Test basic operations for resources in an high-end edge server
Type	online response time, live data, live environment, error handling
Setup	The nodered-datatransformer-provider is deployed in a virtual machine in Google Cloud. The virtual machine is g1-small (1 vCPU, 1.7 GB memory) in us-east1-b. This provider will be tested to create Node-RED resources for clients in a Kubernetes cluster running in us-central1-a. The cluster has three nodes of n1-standard-1 (1 vCPU, 3.75 GB memory). A Python test script is running from another machine from a home in Vienna.
Start	Run the provider in Google cloud and then run the test
Req.	
Input	a test script with a sequence of three service calls: create a node-red resource, get information about the newly created resource and removed the resource.
Output	Success of tasks and performance values
Logs	Summarized measurements are stored in GitHub. Logs are not stored as the tests can be easily rerun.
Outcome	Pass / Fail

T1.1.3 Create, list and delete a Mosquitto MQTT Broker resource in a high-end edge server

ID	T1.1.3
Test	Test basic operations for resources in an high-end edge server
Type	online response time, live data, live environment, error handling
Setup	The mosquitt-mqtt-provider is deployed in a virtual machine in Google Cloud. The virtual machine is g1-small (1 vCPU, 1.7 GB memory) in us-east1-b. This provider will be tested to create Node-RED resources for clients in a Kubernetes cluster running in us-central1-a. The cluster has three nodes of n1-standard-1 (1 vCPU, 3.75 GB memory). A Python test script is running from another machine in TU Wien.
Start	Run the provider in Google cloud and then run the test
Req.	
Input	a test script with a sequence of three service calls: create a node-red resource, get information about the newly created resource and removed the resource.
Output	Success of tasks and performance values
Logs	Summarized measurements are stored in GitHub. Logs are not stored as the tests can be easily rerun.
Outcome	Pass / Fail

Use case Create a resource slice in an Edge system

T1.2.1 Create and delete a resource slice of Mosquitto MQTT Broker and Node-RED a lightweighted edge server

ID	T1.2.1
Test	Test basic operations for resources in an high-end edge server
Type	online response time, live data, live environment, error handling
Setup	The mosquitt-mqtt-provider is deployed in a virtual machine in Google Cloud. The virtual machine is g1-small (1 vCPU, 1.7 GB memory) in us-east1-b. This provider will be tested to create Node-RED resources for clients in a Kubernetes cluster running in us-central1-a. The cluster has three nodes of n1-standard-1 (1 vCPU, 3.75 GB memory). A Python test script is running from another machine in TU Wien.
Start	Run the provider in Google cloud and then run the test
Req.	
Input	a test script with a sequence of three service calls: create a node-red resource, get information about the newly created resource and removed the resource.
Output	Success of tasks and performance values
Logs	Summarized measurements are stored in GitHub. Logs are not stored as the tests can be easily rerun.
Outcome	Pass / Fail

Use case deploy workflow for data conversion

T1.3.1 Create a slice of Node-RED resource and deploy a workflow

ID	T1.3.1
Test	Test basic operations for resources in an high-end edge server
Type	online response time, live data, live environment, error handling
Setup	The mosquitt-mqtt-provider is deployed in a virtual machine in Google Cloud. The virtual machine is g1-small (1 vCPU, 1.7 GB memory) in us-east1-b. This provider will be tested to create Node-RED resources for clients in a Kubernetes cluster running in us-central1-a. The cluster has three nodes of n1-standard-1 (1 vCPU, 3.75 GB memory). A Python test script is running from another machine in TU Wien.
Start	Run the provider in Google cloud and then run the test
Req.	
Input	a test script with a sequence of three service calls: create a node-red resource, get information about the newly created resource and removed the resource.
Output	Success of tasks and performance values
Logs	Summarized measurements are stored in GitHub. Logs are not stored as the tests can be easily rerun.
Outcome	Pass / Fail

Use case deploy a program accessing INTER-IoT Middleware

T1.4.1 Create a resource and deploy a program obtaining gate access data

ID	T1.4.1
Test	Deploy a resource slice include a program obtaining gate access data from INTER-IoT Middleware
Type	online response time, live data, live environment, error handling
Setup	Assume that INTER-IoT Middleware available. A kubernetes platform is available.
Start	Run rsiHub and then run the test
Req.	
Input	a test script with a sequence of three service calls: search the artefact, deploy the artefact for the resource and remove the resource
Output	Success of tasks and performance values
Logs	Logs are not stored as the tests can be easily rerun.
Outcome	Pass / Fail

Use case check robustness of resource creation

T1.5.1 Check robustness of Node-RED resource creation

ID	T1.5.1
Test	Test if the resource creation is robust and elastic in Cloud.
Type	backup and recovery, stress testing, live environment, error handling
Setup	The nodered-datatransformer-provider is deployed in a Kubernetes cluster in Google Cloud Platform. The cluster has elasticity rules.
Start	Run the provider in Google cloud and then run the test
Req.	
Input	a test script makes a series of service calls creating node-red resource instances. The number of instances is increased, more than the initial number of nodes in the cluster. Another test script removes instances, the number of cluster nodes are also decreased
Output	Success of tasks and the automatic increase/decrease of cluster nodes to accommodate requests
Logs	Summarized measurements are stored in GitHub. Logs are not stored as the tests can be easily rerun.
Outcome	Pass / Fail

T1.5.2 Prevent misuse of API of Node-RED resource creation

ID	T1.5.2
Test	Test if the misuse of resource creation API is prevented.
Type	Interface bound checks, live environment, error handling
Setup	The nodered-datatransformer-provider is deployed in a high-end server but not in the cloud.
Start	Run the provider and then run the test
Req.	
Input	a test script makes a series of service calls creating node-red resource instances. The number of instances is increased but the service calls will be rejected when the number of instances reach a pre-defined limited.
Output	Success of tasks and the rejection of API calls when the limit is reached.
Logs	Summarized measurements are stored in GitHub. Logs are not stored as the tests can be easily rerun.
Outcome	Pass / Fail

T1.5.3 Check the failure of back-end MongoDB database in different resource providers and services

ID	T1.5.3
Test	Test if the failure of back-end MongoDB database causes problems for resource providers
Type	Interface bound checks, backup and recovery, live environment, error handling
Setup	The nodered-datatransformer-provider, mosquitt-mqtt-provider, and rsiGlobalService is deployed with MongoDB as the backend database. MongoDB is used with multiple nodes and replication
Start	Run the providers and services and then run the test
Req.	
Input	a test script kills a MongoDB instance and makes calls to providers and services, which operates normal.
Output	Success of tasks and no interruption of services and providers.
Logs	Summarized measurements are stored in GitHub. Logs are not stored as the tests can be easily rerun.
Outcome	Pass / Fail

Scenario Deploying Resource Slices IoT Cloud for Data Conversion

In this scenario, a resource slice includes a sensor provider, a broker, simple analytics program and big data storage. This resource slice reflects the case when, for example, a stakeholder activates sensors (e.g., monitoring trucks, containers, or temperatures of some objects) and requires sensor data to be sent to the queues to an analytic program which pushes the results to Sensors, MQTT and Ingest, BigQuery and then the new elements as mentioned in the paper.

A video is shown in youtube: https://youtu.be/_SCrK8Q3xBs

Use case Create the Resource Slice sensor-broker-analytics-bigquery

T2.1.1 Create a sensor-mqtt-analytics-bigquery slice

ID	T2.1.1
Test	Test a resource slice of sensor-mqtt-analytics-bigquery
Type	online response time, stress testing, live data, live environment, error handling
Setup	rsiHub setup. Providers for sensors, mqtt, analytics and BigQuery are setup
Start	Run rsiHub and providers. Then start to define the slice and use pizza commandline to perform the slice creation
Req.	
Input	Resource slice
Output	Success of tasks
Logs	Video in youtube.
Outcome	Pass / Fail

Use case Search for resources and artifacts for interoperability bridges

T2.1.2 Search for resources and artifacts suitable for interoperability bridges

ID	T2.1.2
Test	Search suitable resources and artifiacts for interoperability bridges
Type	online response time, live data, live environment, error handling
Setup	rsiHub is running, IoT providers running, artifacts are in the repository. Metadata about interoperability are associated with artifacts and resources
Start	Run rsiHub and provide a search commandline using pizza
Req.	
Input	Criteria for search.
Output	Success of tasks and return valid result
Logs	
Outcome	Pass / Fail

Use case Update sensor-broker-analytics-bigquery with new receiver

T2.1.3 Update the slice of sensor-broker-analytics-bigquery with new resources

ID	T2.1.3
Test	Update an existing resource slice with new resource
Type	online response time, live environment, error handling
Setup	rsiHub is running. Providers are running. Existing slice is running
Start	Define new resources and run the pizza command line for update
Req.	
Input	New resources defined.
Output	Success of tasks
Logs	
Outcome	Pass / Fail

Scenario resource Slice for Exchanging data between Port and Vessel in Emergency

We assume there are alarms occurring in a seaport. The alarms are propagated through an MQTT broker. Usually, there are some analytics applications listening the alarms queues to react to the alarms. One of such alarms analytics programs finds alarms related to terminals in the port. It queries a PortControlService (PCS) to obtain the list of vessels approaching the port. Based on the information about the vessels and the service providers of the vessels, the alarms analytics program creates new brokers as resources or connects to existing communication means of the vessel providers to share the information about the situations. The program can also send requests to ask vessels to stop or change the plan to arrive terminals.

Similarly, another analytics program can also inform other relevant objects around the terminals (e.g., by query trucks) and requests them to stop or change the plan. Another analytics program can request camera providers (for cameras close to the terminal, using geohash) to provide videos to separate channels that can be accessed by polices and other relevant third parties.

We have:

- AlarmSensor: emulates alarms
- AlarmService: the service captures alarms and handles alarm in the port
- A MQTT Broker Provider for messaging
- A Port Control Service: emulates the port control service which accepts information about vessels and trucks.
- VesselService: emulates a vessel.
- Vessel Provider: emulates the vessels registering to the port and approaching the port.
- TruckMonitoringProvider: monitors the entrance and existence of trucks by obtaining events from INTER-Miiddleware.

A basic resource slice for alarms will include: Port Control Service, AlarmService, VesselProvider and TruckMonitoringProvider.

For the use case of alarms, AlarmSensor will be activated: a new resource is added and thus existing vessels and trucks should get notifications about alarms.

Use case: Create a resource slice for alarm

T3.1.1 Create resource slices for alarms

ID	T3.1.1
Test	Test a resource slice for alarms
Type	online response time, live data, live environment, error handling
Setup	rsiHub and providers, truck information sent by INTER-IoT Middleware is emulated through a RabbitMQ with real dataset.
Start	Run rsiHub and providers; run the pizza commands to create the basic resource slice
Req.	
Input	Resource information and configuration
Output	Success of the slice creation
Logs	
Outcome	Pass / Fail

Use case: add new alarms and vessels into the alarm slice

T3.1.2 Integrate new vessels and alarms in resource slices

ID	T3.1.2
Test	Test new vessels and alarms in the resource slice for alarms
Type	online response time, live data, live environment, error handling
Setup	rsiHub and providers, truck information sent by INTER-IoT Middleware is emulated through a RabbitMQ with real dataset.
Start	Create new vessels, create new alarms
Req.	
Input	New vessels, new alarms
Output	New vessels receive alarms, new alarms are handled and propagated to vessels & trucks
Logs	
Outcome	Pass / Fail

3.3.9.6 Test outcome overview

The following table will provide an overview of the test result of all the performed tests in this SAT.

Test	Description	Outcome
T1.1.1	Create, list and delete a resource in a lightweighted edge server	Pass / Fail
T1.1.2	Create, list and delete a Node-RED resource in a high-end edge server	Pass / Fail
T1.1.3	Create, list and delete a Mosquitto MQTT Broker resource in a high-end edge server	Pass / Fail
T1.2.1	Create and delete a resource slice of Mosquitto MQTT Broker and Node-RED a lightweighted edge server	Pass / Fail
T1.3.1	Create a slice of Node-RED resource and deploy a workflow	Pass / Fail
T1.4.1	Create a resource and deploy a program access gate access data	Pass / Fail
T1.5.1	Check robustness of Node-RED resource creation	Pass / Fail
T1.5.2	Prevent misuse of API of Node-RED resource creation	Pass / Fail
T1.5.3	Check the failure of back-end MongoDB database in different resource providers and services	Pass / Fail
T2.1.1	Create a sensor-mqtt-analytics-bigquery slice	Pass / Fail
T2.1.2	Search for resources and artifacts suitable for interoperability bridges	Pass / Fail
T2.1.3	Update the slice of sensor-broker-analytics-bigquery with new resources	Pass / Fail
T3.1.1	Create resource slices for alarms	Pass / Fail
T3.1.2	Integrate new vessels and alarms in resource slices	Pass / Fail
SAT Outcome		**Pass / Fail**

Table 67: Test outcome overview

3.3.9.7 Integration ethics and security

In INTER-HINC we do not address ethics and security. When services deployed atop infrastructures, they rely on the infrastructure security.

3.3.10 Third Party: Semantic Middleware

Figure 97 depicts the overall architecture of the Semantic Middleware, focusing on its semantic information, integration and dispatching capabilities. The diagram outlines the components in charge of supporting it: Update Manager (UM) and Semantic Broker (SB), on its turn made up by the Subscription Manager (SM) and the Messaging System (MS) supported by a multi-agent System. Each sensor, after having gathered the information which oversees, affects the knowledge base (**GOIoTP**) hosted on the shared semantic repository (**RDF store**) by updating or deleting semantic assertions. This is done through a web service exposed by UM. On the other hand, information consumers (smart services and sensors) subscribes to the SM, providing their profile of interest.

SM is the component which is always listening on the queue that manages the new subscriptions, leveraging the Apache ActiveMQ (ActiveMQ) messaging system. Whenever SM receives a subscription request from a network client, it activates server-side an agent (the ClientAgent) which takes care of the client interests. Namely, SM records this interest activating an agent in charge of signaling emerging new information to the consumer. If such agent already exists, SM simply notifies the new consumers' interest. In each moment, the consumer can unsubscribe by cancelling the request. Each time a sensor authors new knowledge, the UM informs the SB of the occurred event so that, in a continuous query processing fashion, the SB can evaluate emerging information and notifies it to the consumer through the MS.

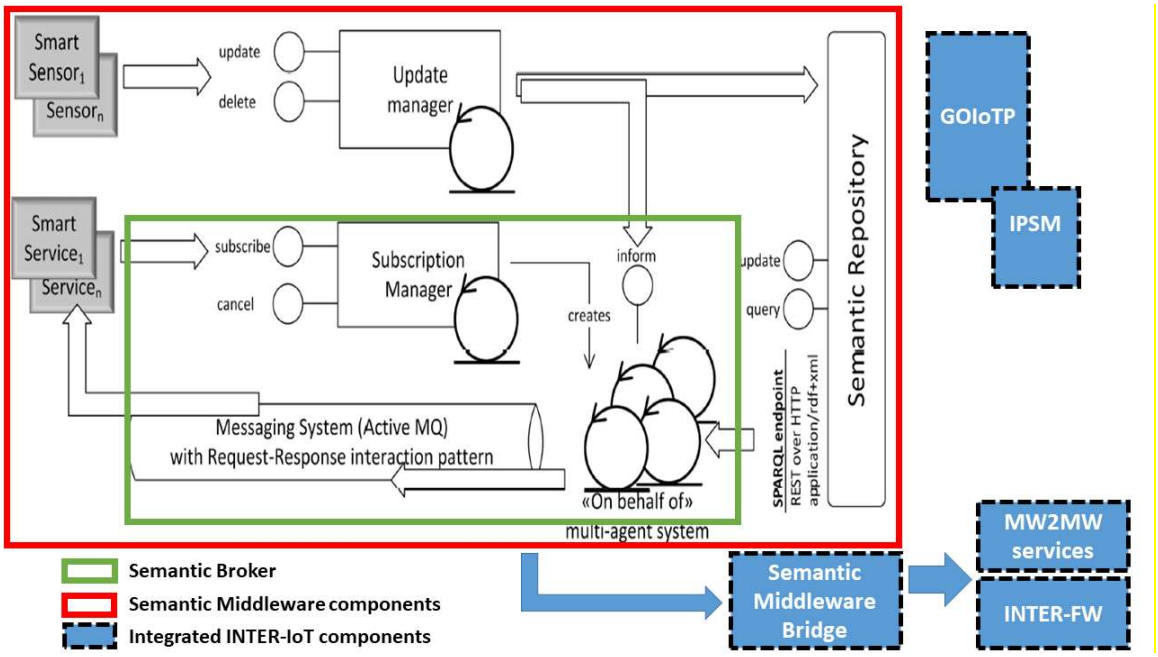


Figure 97: Overall architecture and its interaction with INTER.IoT

The client starts the interaction with the server through a subscription message containing the description of the information of interest (specified in a query) with the minimum refresh rate in milliseconds, together with a unique identifier of the request (req-id) and a reference (pointer, address, etc.) of the client (client-ref.) to which forward the discovered information.

The query transmitted from the client is expressed through the SPARQL 1.1 syntax and may be a SELECT, ASK or CONSTRUCT that refers to semantic model contained in the

repository. The server processes the subscription request and decides whether to accept it. If it is rejected, the repository sends to the client the rejection condition ending the interaction. If it is accepted, at each interval of minimum refresh rate, the server updates the evaluation of the subscribed queries and transmits an information message (inform-result to the client) containing the result of the executed query (query-result) if the result is not empty (SELECT or CONSTRUCT query type) or positive (ASK query type), according to the chosen response format.

The server continues to broadcast type messages (inform-result) as long as one of the following conditions happen:

1. the client deletes the subscription request by the cancellation request (see next section);
2. an error occurs for which the server is no longer able to communicate with the client or to process queries.

All interactions are identified by a unique identifier other than zero (req-id) assigned by the initiator of the protocol and valid for it (client-ref). This allows stakeholders to manage their communication strategies and activities. Moreover, since it can be important to preserve the sequence of the messages, the transport layer has to preserve the order of the messages (reliable transport layer). Thanks to the uniqueness of the req-id, each client can participate in multiple signaling at the same time. Figure 98 reports the overall workflow of the subscription process.

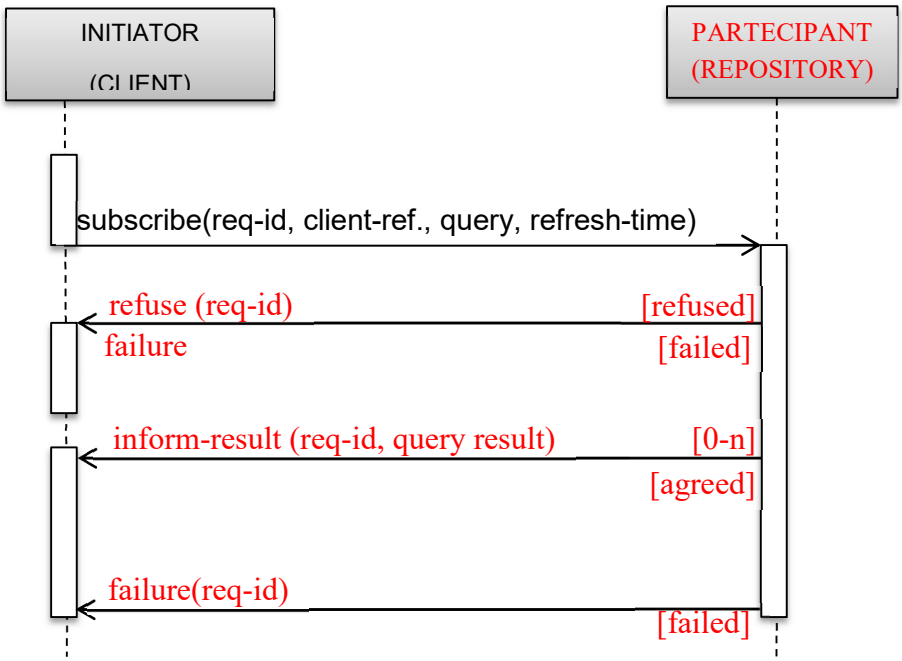


Figure 98. Subscription and notification workflow

At any time, the client may cancel a subscription request by transmitting a cancel request to the server. In such a request, the parameters *req-id-ref* and *client* identify the interaction to be stopped (Figure 103). The server inform then the client if the interruption succeeded (done) or that it was not possible to break the interaction due to an error (failure).

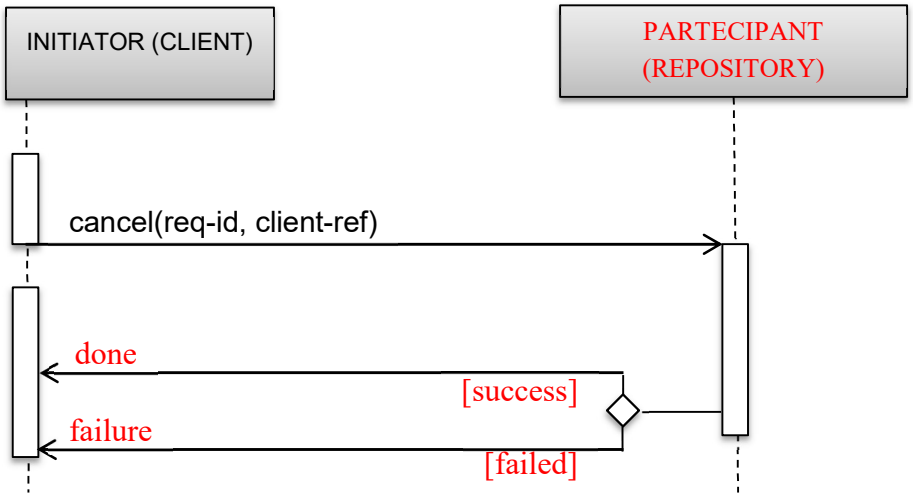


Figure 99. Workflow of the cancellation of the subscription

3.3.10.1 Integration of IoT framework

Internet-of-Things it is not just a matter of connecting things to the Internet; but more importantly linking things in a synergistic networks of virtual and physical components able to find, access, manage each other and work together with a higher degree of interoperability. In order to enable such interoperability, it is needed to make things smarter, i.e., provide them with the capabilities of exchange information, also at a semantic level, by providing them the functions for register and follow alerts of changes information.

The scope of this work is to describe our experience in integrating the CNR-ITIA IoT platform *Virtual Factory* with other Internet-of-Things (IoT) platforms leveraging the methodology and framework of the INTER-IoT European project. *Virtual Factory* is a platform leveraging Semantic Middleware to allow distributed manufacturing resources to interoperate and collaborate with each other exploiting the IoT paradigm.

One of the main challenges for integrating different IoT platforms is to put in communication these platforms equipped with heterogeneous communication protocols and technologies in a way as smooth as possible, which guarantees a thorough interoperability in each architectural levels of the platform, thus promoting the synergistic usage of existing IoT platforms. This objective is in line with the INTER-IoT project, whose aim is to design, implement and test a framework for inter-platform communication. So far, various existing IoT platforms have been plugged into the framework ranging from various application fields, thus demonstrating the advantages of such an integration platform disregarding the specific application context. In this paper, we propose and discuss how *Virtual Factory* platform can be integrated into INTER-IoT. From such integration, we expect to benefit from all the advantages brought by the INTER-IoT project, mostly related with 1) the possibility offered by the integration platform to exchange information between the plugged platforms in a seamless and smart way; 2) the easing of the communication process between client and *Virtual Factory* back-end services, by exploiting tested and robust communication mechanisms provided within INTER-IoT implementation.

This section explores the integration of the *Semantic Middleware* with INTER-IoT components \ products. The idea is implementig a **"Virtual Factory Bridge"** which handles the connection of our *Virtual Factory* with all the underlying platforms, with the **MW2MW** services and with the **INTER-FW** (Figure 97, Figure 100 and Figure 102). Under these conditions, the *Virtual Factory* can be considered as a new platform. In this way, *Virtual Factory* can ask information requests to the the other underlying platforms. The requests can concern information about specific values. They are expressed composing a message queue as specified in D3.1 (using Kafka protocol). As soon as the *Virtual Factory* receives the answer, it elaborates the received information. The realization of the platform will be based on "the generic interface which provides a structured template to easily develop new bridges." (D3.1). In addition, the *Virtual Factory* will expose its functionalities according to the structure provided by the "generic interface".

In addition to **Bridge**, another INTER-IoT product that *Semantic Middleware* will use is **GOIoTP** (Figure 97 and Figure 101). Indeed, *Semantic Middleware* is agnostic to the meta-model (TBOX) of the IoT platform ontology, i.e. the behavior of the *Semantic Middleware* does not depend on a specific semantic structure of the ontology. Under these conditions, the *Semantic Middleware* uses the **GOIoTP** paired with an application ontology specific of the analyzed scenario (Figure 101). **GOIoTP** is taken as global common semantic model that all the devices share and it will be used to represent general concepts of our scenario. Thus, a link between our application ontology and **GOIoTP** will be studied and implemented (see

integration with the **IPSM** below). The ontology model will be stored into the RDF store included into the Semantic Middleware through its provided SPARQL engine.

In addition, since it is essential to reconcile and mediate the application ontologies handled by applications and devices and the core ontology shared within the whole platform, it should be evaluated the integration of *Semantic Middleware* with **Inter Platform Semantic Mediator (IPSM)** component proposed in INTER-IoT.In particular, the latter allows the alignment of the commonalities (overlapping concepts) between the domain ontology (RDF model) used in a specific IoT platform and the core ontology (RDF model) defined within the INTER-IoT. It will be necessary to define the rules of the mapping for each domain ontology that has to be aligned with the core ontology. Thus, **IPSM** will be exploited to translate and link the meta-model (TBOX) of our application ontology with **GOIoTP**.

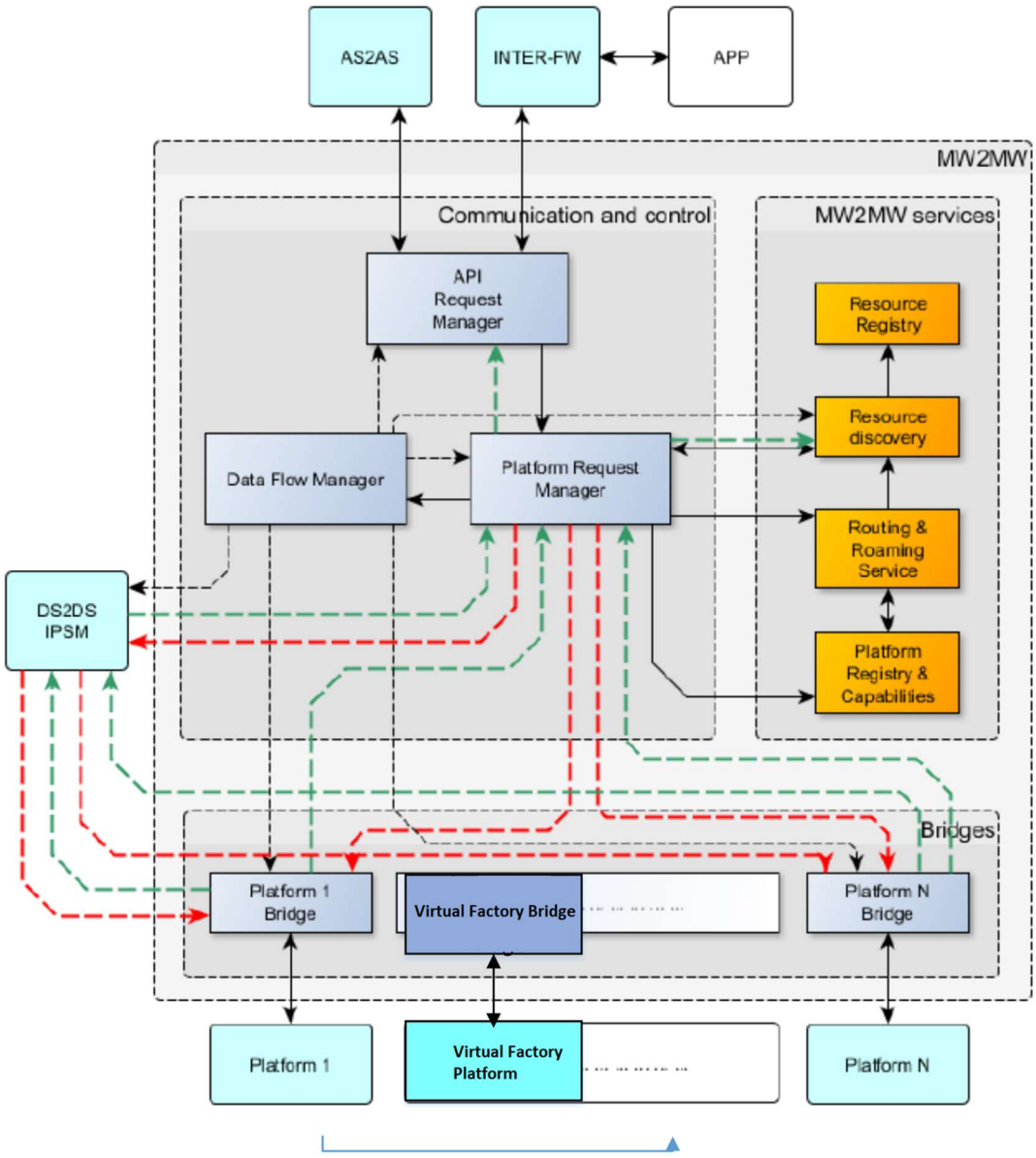


Figure 100. Semantic Middleware Bridge

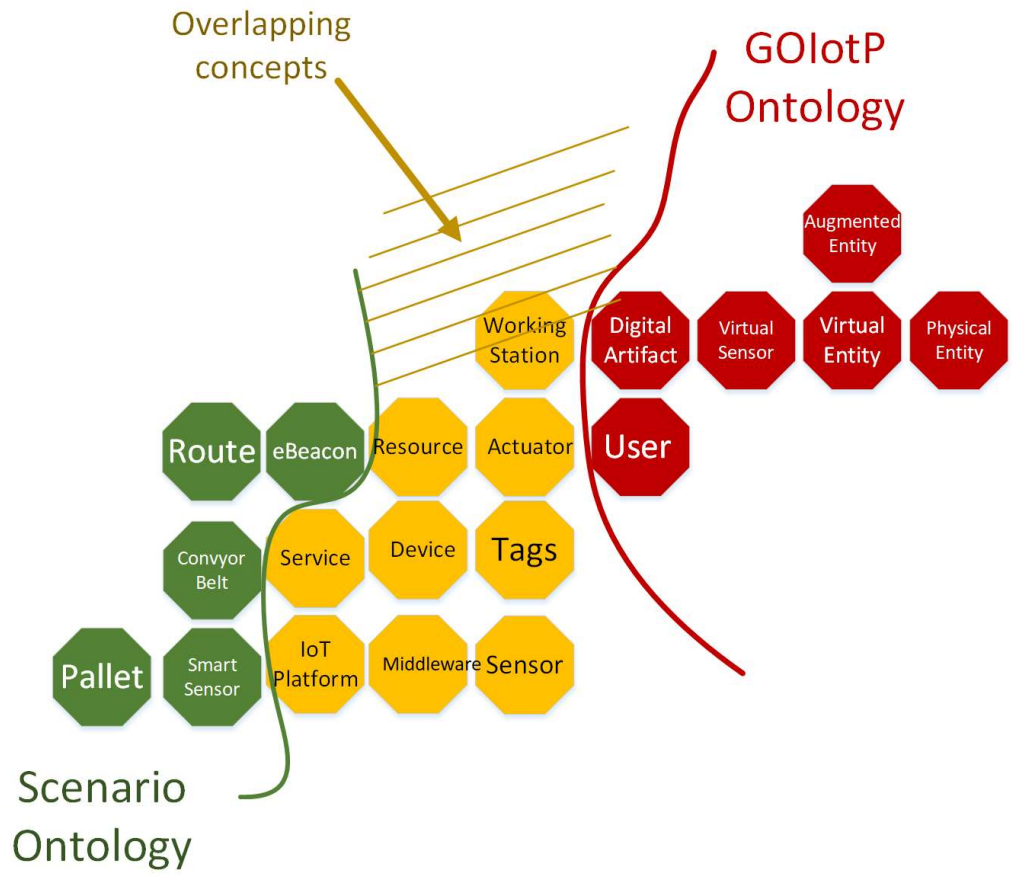


Figure 101. Overlapping between application ontology and GIoTP

Components	Interface overview of the used IoT components	Tests
Virtual Factory Bridge	The generic common interface as a structured template to easily develop new bridges (D3.1)	T34.62.5
GOIoTP	Various entities defined in the Semantic Model	All
IPSM	Invoking the function of IPMS Aligner (dashboard : http://grieg.ibspan.waw.pl:3000/translation)	All
INTER-MW	INTER-IoT rest API (Figure 6)	T34.62.5

Table 68:Components and interface overview

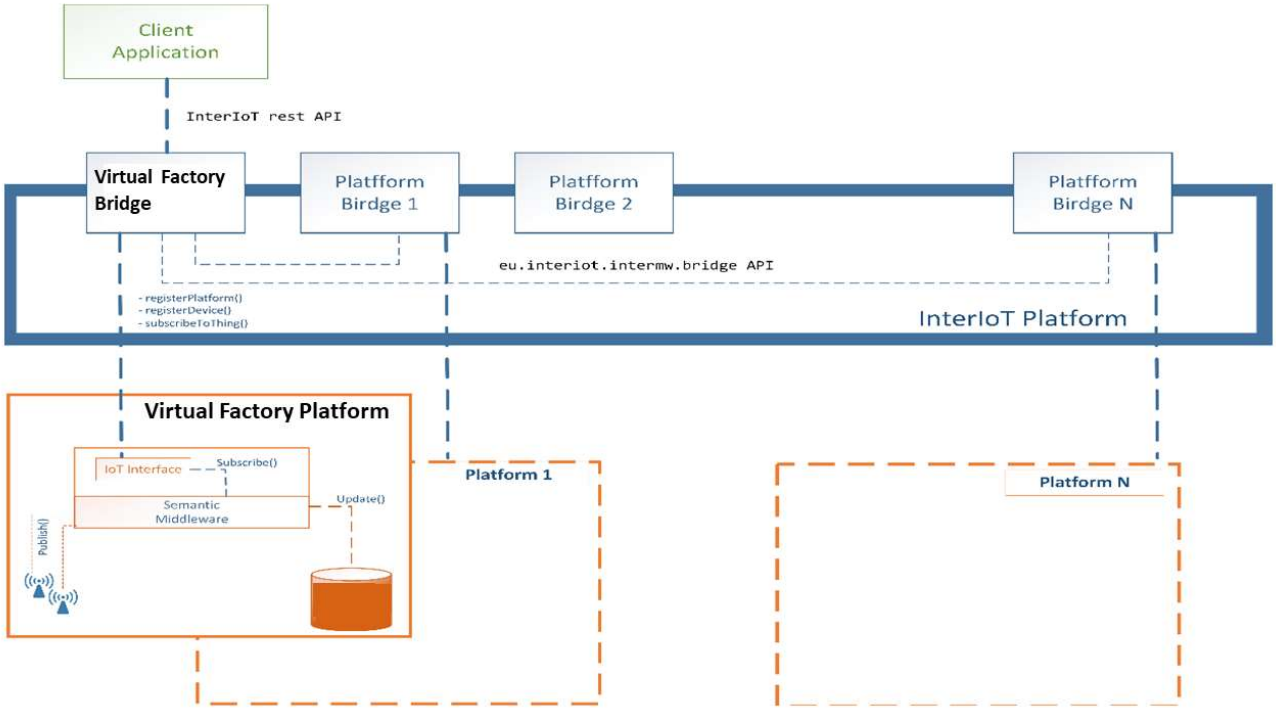


Figure 102. Integration of the Semantic Middleware with the IoT framework

3.3.10.2 Deliverables and version overview

The following table contains a deliverable list which need to be signed of before SAT testing commences.

ID	Description	Check
Documents		
1	Validation and Test reports of the Semantic Middleware components	
2	Validation and Test reports of the Pilot system components	
3		
Hardware		
4	Workstation server \ Cloud (server which hosts Semantic Middleware server + database)	
5	PC client (PC which hosts the Virtual Factory monitoring application)	
6		
Tools		
7	Semantic Middleware as a central component of the Virtual Factory Platform	
8	Virtual Factory Platform	
9	Virtual Factory Bridge Interface component (packed as Java Archive File)	

Table 69: Deliverable checklist

The following table shows the software components and version of which the system release version 1.0 consists of.

ID	Description	Version	Check
IoT Physical Gateway			
1	Kafka		
2	Rabbit		
3	ActiveMq		

4	Mqtt		
IoT Virtual Gateway			
5	Virtual Factory Bridge	1.0	
6	GOIoTP	0.9	
7	IPSM dashboard (http://grieg.ibspan.waw.pl:3000/)		
8	INTER-MW	2.0.1	
9	Docker container	1.13.1	
10	Parliament		
Semantic Middleware			
11	Semantic Broker	1.0	
12	Update Manager	1.0.0	
13	Client Semantic Middleware Library	1.0.0	
14	RDF store (Stardog[28] free version)	5.0.0	
15	Publish-Subscribe middleware (ActiveMQ[29])	5.15.1	
16	Semantic data model (Application Ontology)	1.0.0	
Virtual Factory Interface			
17	Virtual Factory monitoring Application	0.1	
18	Virtual Factory Platform	0.1	

Table 70: Component version overview

3.3.10.3 Requirements, scenarios and use cases

The following table provides an overview of the relation between the requirements and the test(s) that validate their implementation.

ID	Description	Covered by
42	Heterogeneous information representation	T34.62.7
75	The interaction between IoT endpoints may follow M2M concept	T34.62.1, T34.62.2, T34.62.3, T34.62.4
96	Enable (automated or semi-automated) linking of relevant data models	T34.62.7
178	Inter Platform Semantic Mediator provides data and semantic interoperability functionality	T34.62.6
179	Inter Platform Semantic Mediator supports platform communication	T34.62.6
180	Syntactic and semantics interoperability - Data format and semantics translation	T34.62.6
237	API Middleware for interoperability between different platforms	T34.62.5
270	API allows subscription to data streams/queues	T34.62.1, T34.62.2, T34.62.3, T34.62.4
282	Map publish/subscription between platforms	T34.62.5

Table 71: Requirements vs test mapping

[28] https://www.stardog.com/

[29] http://activemq.apache.org/

The following table provides an overview of the relation between the scenarios and the test(s) that validate their implementation.

ID	Scenario name	Covered by
32	Third party developer using INTER-FW to access data from two different platforms	T34.62.5 T34.62.6 T34.62.7
33	Heterogeneous Platforms Methodology-driven Integration	T34.62.5 T34.62.6 T34.62.7
34	Virtual Factory: Position and Optimization of the pallets **(NEW SCENARIO)**	T34.62.1, T34.62.2, T34.62.3, T34.62.4, T34.62.5 T34.62.6 T34.62.7

Table 72: Scenario vs test mapping

S34: Position and Optimization of the pallets

The sensors monitoring the pallet position will play the role of publisher as they will send the information concerning the pallet position through the middleware (Step 1); this information is expressed under the form of a SPARQL UPDATE. Also the working stations will publish their availability status (Step 2). This information will be then consumed by the simulation tool (Optimizer) which has previously subscribed to the changes applied to the pallet position (Step 3) and the availability status of the working stations (using a proper SPARQL query) with the goal to identify the optimized pallet route. In addition, the information concerning the route is then published (Step 4) and in its turn consumed by the IoT actuators which allow to change the route of the pallets along the conveyor belt (Step 5).

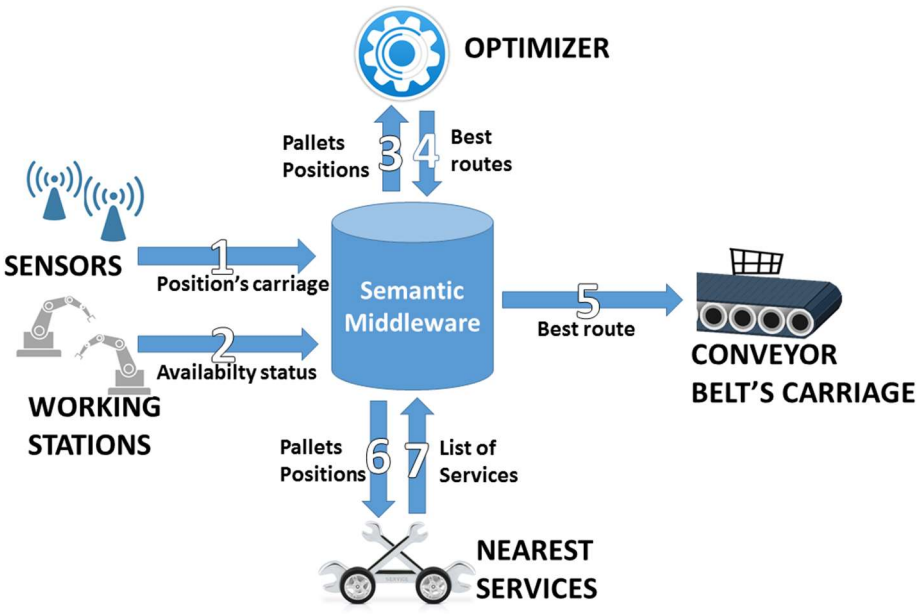


Figure 103. Workflow of the scenario

The use cases reported in the following sections involve various components integrated with the *Semantic Middleware*. The components are the following:

- The Semantic Model. The Semantic model used in these SATs represents knowledge concerning sensors and their corresponding measures, pallets and their real and optimized positions.
- An applications (*Virtual sensor*) that mocks and simulates the behavior of an Ebeacon sensor tracking the position of the pallets. These sensors will publish, through an

UPDATE SPARQL (**Query 1**), the position of the pallets P1 and P2 within the knowledge base, according to a proper domain ontology.
To support the SAT, *Virtual Sensor* provides a GUI that allows to set a position for a specific pallet, thus generating the corresponding UPDATE SPARQL query.

- *Optimizer*, a simulation tool which uses the pallet position and the availability status of the working stations with the goal to identify the optimized pallet route. It consumes data published by the various tracking sensors, while it publishes optimized routes for the pallets.
  To support the SAT, *Optimizer* is virtualized and provides a GUI that allows to start to listen and consume information concerning position of a specific pallet, thus generating the corresponding SELECT SPARQL query (**Query 2**). In addition, *Virtual Optimizer* allows to set the new best route for the pallet, simulating the behaviour of the real application.
- *Virtual carriages* (simulating the real carriages) each one transporting a pallet. They consume data published by the simulator in order to follow a specific route.
  To support the SAT, *Virtual carriage* provides a GUI that allows to start to listen and consume information concerning route for a specific pallet, thus generating the corresponding UPDATE SPARQL query (**Query 3**).

3.3.10.4 Test environment

Introduction

To test the functionality of the integrated Semantic middleware in combination with the IoT framework representative test hooks in the system are needed. This chapter will describe this environment and the used hardware, software, tools and platforms.

Test environment

This paragraph describes the test setups, hooks and probes used in the system used during this SAT.

Test tools, hooks and probes

This paragraph describes the test setups, tools, hooks and probes used in the by this document defined tests.

A test setup defines a setup used for a test, and describe the used system parts, interface etc.

Tools define the used COTS tooling that will be used during testing e.g. Logic analyzers, packet sniffer like e.g. Whireshark, developed scripts, etc.

Hooks are used to inject data or control parts of the system used to get the system to handle a needed scenario. This can be hidden, debug or service interfaces which contain the needed functionality.

Probes are used to gather logging from parts of the system under test to provide system feedback and to prove that the system is handling the scenario as it should.

The following paragraphs will define the used setups, tools, hooks and probes which will be referred to by the test descriptions.

TS_01 SemanticMiddlewareComponents

The server components, which are Update Manager (UM) and Semantic Broker (SB), are deployed on a server. The client component (Client Semantic Middleware Library) is deployed on the client application (typically included in its setup package).

TS_02 IPSM

The IoT IPSM is up and running.

TS_03 GOIoTP

The GOIoTP ontology. The file owl corresponding to GOIoTP has been created starting from the corresponding ttl file.

TS_04 MW2MW

It includes the following components:

1. The middleware service (v. 2.0.1) running on a Docker container (docker version 1.13.1 build 092cba3) installed in a Ubuntu (r. 16.04) server available in a LAN used for the present integration test;
2. The Java Archive file (mw.bridge.testBridge-0.0.1-SNAPSHOT.jar) containing all the classes and resources files implementing the Bridge;
3. The TestedPlatform Emulator running on a different host connected to the INTER-MW service via the LAN.

TS_05 Virtual Factory platform and its bridge

The middle component acting as a request handler mediator between the INTER-MW (interfaced with Virtual Factory Bridge) and the client application.

TT_01 RDF store

An RDF store (we propose the version trial of Stardog: http://www.stardog.com/, but any other kind of RDF store can be used).

TT_02 ActiveMq

A publish-subscribe middleware. We propose Apache ActiveMQ which is one of the most popular open source messaging middleware (ActiveMQ) and supports the most well-known standard protocols for messaging. Namely, Apache ActiveMQ supports MQTT (MQTT), a lightweight messaging protocol built on top of TCP and characterized by low bandwidths as needed by IoT ecosystems, and Openwire (ActiveMQ OpenWire), the Apache ActiveMQ default message oriented protocol designed for performances optimization and supporting private peer to-peer queues.

TH_01 Semantic Model

A Semantic Model, reported as owl file, which can be imported by the RDF store. This model represents the knowledge concerning the scenario environments.

The Semantic model is paired with a list of SPARQL queries to subscribe a change into the knowledge base and a list of SPARQL queries that updates the knowledge base in correspondence of the subscription.

In Section 0 it is reported some hints about the semantic model used in these FAT and about involved SPARQL queries (e.g. Query 1, etc.).

TP_01 FeedbackWithinGUI

Feedback of the tests will be shown within the GUI of the client applications.

TP_02 Virtual Factory monitoring Client Application

A GUI-based client application allowing the continuous monitoring of the smart factory environment by visualizing various sensor measures request through the INTER-MW communication protocol.

No specific tools will be used for site acceptance testing. On the contrary, a monitoring application will be used to demonstrate the correct behaviour of the Virtual Factory platform while interfacing to the INTER-MW via the Virtual Factory bridge. Different debug messages inside the bridge and in the client application will be used to validate the test process.

TP_03 LogFile

A log file is provided which reports the auditing of some operations.

3.3.10.5 Test description

S34: Position and Optimization of the pallets

U62 – Device (sensor) triggers information

A device, typically a sensor, triggers an event sending determined information to the gateway in order to be stored on the platform Cloud or server or in order to generate a response for an actuator (being handled by the rules engine).

This use case involves these requirements: [75], [163], [270].

T34.62.1 Information published by Virtual Sensor are persisted

ID	**T34.62.1**
Test	Virtual Sensor publishes information and this information is persisted into the RDF store
Type	System testing
Setup	Need test setup TS_01 SemanticMiddlewareComponents (Query 1, etc.) Need test tool TT_01 RDF store Need test tool TT_02 ActiveMQ Need test hook TH_01 Semantic Model (Query 1, etc.)
Start	Information to be published are not yet persisted
Req.	[75], [163], [270]
Input	Enable the sensor within range of the physical gateway
Output	• The result of a SPARQL query on the RDF store
Logs	The output log is stored in Folder LOG, in a file with prefix "`T34.62.1_log`"
Outcome	Pass / Fail

Test output:

- Access the RDF store and verify through a SPARQL query (SELECT) if the information has been stored

T34.62.2 Information updated by Virtual Sensor are received by the subscribed clients

ID	T34.62.2
Test	Information updates are received by the subscribed clients
Type	System testing
Setup	Need test setup TS_01 SemanticMiddlewareComponents Need test tool TT_01 RDF store Need test tool TT_02 ActiveMQ Need test hook TH_01 Semantic Model (Query 1, etc.)
Start	Information to be published are not yet persisted A client (Optimizer) is subscribed to the updated information
Req.	[75], [163], [270]
Input	Information published by Virtual Sensor. It concerns the position of the pallet.
Output	• Check if the subscriber (Optimizer) receives the information updates (postion of the pallet). It has to receive the information updated by the Virtual Sensor. • Check if other subscribers (Virtual Carriage), which are not subscribed to the updated information, does not receive the updated information.
Logs	The output log is stored in Folder LOG, in a file with prefix "`T34.62.2_log`"
Outcome	Pass / Fail

Test output:

- Feedback of the tests will be shown within the GUI of the client applications (TP_01 FeedbackWithinGUI).
- Also a log file can be provided (TP_03 LofFile)

T34.62.3 Information updated by Optimizer are received by Virtual Carriage

ID	T34.62.3
Test	Information updates are received by the subscribed clients
Type	System testing
Setup	Need test setup TS_01 SemanticMiddlewareComponents Need test tool TT_01 RDF store Need test tool TT_02 ActiveMQ Need test hook TH_01 Semantic Model (Query 2 and Query 3)
Start	Information to be published are not yet persisted A client (Virtual carriage) is subscribed to the updated information
Req.	[75], [163], [270]
Input	Information published by Optimizer (new route of the pallet).
Output	• Check if the subscriber (Virtual carriage) receives the information updates. The have to receive the updates triggered by the Optimizer changes.
Logs	The output log is stored in Folder LOG, in a file with prefix "`T34.62.3_log`"
Outcome	Pass / Fail

Test output:

- Feedback of the tests will be shown within the GUI of the client applications (TP_01 FeedbackWithinGUI).
- Also a log file can be provided (TP_03 LofFile)

T34.62.4 Updates concerning information on which no client is subscribed

ID	T34.62.4
Test	Updates concerning information on which no client is subscribed
Type	System testing
Setup	Need test setup TS_01 SemanticMiddlewareComponents Need test tool TT_01 RDF store Need test tool TT_02 ActiveMQ Need test hook TH_01 Semantic Model (Query 1, etc.)
Start	Information to be published are not yet persisted A couple of clients (Virtual carriage and Optimizer) are subscribed to various information. The latter are not linked with the updated information
Req.	[75], [163], [270]
Input	Information published by Optimizer
Output	• Check if the subscribers (Virtual carriage) receives or not the information updates: They do not have to receive the updates.
Logs	The output log is stored in Folder LOG, in a file with prefix "`T34.62.4_log`"
Outcome	Pass / Fail

Test output:

- Feedback of the tests will be shown within the GUI of the client applications (TP_01 FeedbackWithinGUI).
- Also a log file can be provided (TP_03 LofFile)

T34.62.5 (TEST of INTEGRATION with IoT components) The connection with the Semantic Middleware Bridge (reading a device steraming data flow)

ID	T34.62.5
Test	Testing the system response to a simulated reduction of the temperature in the tested environment
Type	Integration with the various components of the system
Setup	The test needs the *Virtual Factory monitoring Client Application* set up (TP_02) and running with an established connection with the Virtual Factory Platform (TS_05), this one connected to INTER-MW through the Virtual Factory Bridge
Start	The Client Application has established a valid connection with the Virtual

	Factory Platform component
Req.	The following requirements are expected in order to execute the test: • The Application client has successful registered to the INTER-MW as a client with specific `ClientID`; • The Virtual Factory Platform has successfully registered with a specific `platformId` for the Virtual Factory platform. • The temperature sensor has successfully recognized within the INTER-MW as a device to which the client can subscribe.
Input	Subscribe to registered temperature sensor (`tempSensor1`) via a application/json packet continaing the subscribe request, the `platformID` and the `thingId`.
Output	A stream data channel is opened as a result of a Virtual Factory Platform connection returned to the callback address. Temperature sensor data flaws from the Platform to the client with a frequency parametrized in the subscribe request.
Logs	Virtual Factory bridge log are stored in the INTER-M log file as debug messages. The Platform log are displayed in the console, while the client app log visualized as GUI messages (status bar messages, dialog windows, etc.) of TP_2.
Outcome	Pass / Fail

T34.62.6 (TEST of INTEGRATION with IoT components) The connection with the Semantic Middleware Bridge (actuating over a controlled device)

ID	T34.62.6
Test	Testing the system response to a new temperature setting in the tested environment
Type	Integration with the various components of the system
Setup	The test needs the *Virtual Factory monitoring Client Application* set up (TP_02) and running with an established connection with the Virtual Factory Platform (TS_05), this one connected to INTER-MW through the Virtual Factory Bridge
Start	The Client Application has established a valid connection with the Virtual Factory Platform component
Req.	The following requirements are expected in order to execute the test: • The Application client has successful registered to the INTER-MW as a client with specific `ClientID`; • The Virtual Factory Platform has successfully registered with a specific `platformId` for the Virtual Factory platform. • The actuator device responsible for the setting of the shop floor indoor temperature has successfully recognized within the INTER-MW as a device to which the client can send an actuate request.
Input	Sending of an actuate request to actuator (`tempEnv1`) via an application/json packet containing the actuate request, the `platformID` and the `thingId`.
Output	The actuation instructions are successfully sent to the actuator device otherwise an exception is thrown.
Logs	Virtual Factory bridge log are stored in the INTER-M log file as debug messages. The Platform log are displayed in the console, while the client app

	log visualized as GUI messages (status bar messages, dialog windows, etc.)
Outcome	Pass / Fail

T34.62.7 (TEST of INTEGRATION with IoT components) The connection with a ThirdParty platform connected to the INTER-MW (reading a device streaming data flow)

ID	T34.62.7
Test	Testing the system response to a simulated reduction of the outdoor temperature affecting indoor temperature parameter setting.
Type	Integration with the various components of the system
Setup	The test needs the *Virtual Factory monitoring Client Application* set up (TP_02) and running with an established connection with the Virtual Factory Platform (TS_05, this one connected to INTER-MW through the Virtual Factory Bridge. The test also needs a ThirdParty platform connected to INTER-MW via a ThirdParty Bridge able to measure outdoor temperature through a connected device (`externalTempSensor`).
Start	The Client Application has established a valid connection with the Virtual Factory Platform component
Req.	The following requirements are expected in order to execute the test: • The Application client has successful registered to the INTER-MW as a client with specific `ClientID`; • The Virtual Factory Platform has successfully registered with a specific `platformId` for the Virtual Factory platform. The same for the ThirdParty Platform. • The external temperature sensor has successfully recognized within the INTER-MW as a device to which the client can subscribe.
Input	Subscribe to registered temperature sensor (`externalTempSensor`) via an application/json packet containing the subscribe request, the `platformID` and the `thingId`. The platfomID is that of the ThirdParty platform.
Output	A stream data channel is opened as a result of a Virtual Factory Platform connection returned to the callback address. Outdoor temperature sensor data flows from the Third-party Platform to the client with a frequency parametrized in the subscribe request.
Logs	Virtual Factory bridge log are stored in the INTER-M log file as debug messages. The Platform log are displayed in the console, while the client app log visualized as GUI messages (status bar messages, dialog windows, etc.)
Outcome	Pass / Fail

T34.62.8 (TEST of INTEGRATION with IoT components) The ontology alignment test through IPSM

ID	T34.62.8
Test	The ontology alignment test between our Application Ontology (AO) and the GOIoTP ontology will be performed according to the general structure of the INTER-IoT alignment format (also called IPSM alignment format) [Ref1]. The alignment element

describes a uni-directional set of translation rules comprised of independent mapping cells, each of which has an “input” and “output” entity descriptions. Elements <onto1> and <onto2> describe respectively the AO ontology and the GOIoTP ontology of the alignment, by giving their URIs and specifying the formalism used for their definition (in our case AO adopt the OWL Lite formalism).

The following alignment is used:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE Alignment
        [
                <!ENTITY sripas "http://www.INTER-IoT.eu/sripas#">
                <!ENTITY sosa "http://www.w3.org/ns/sosa/">
                <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
                <!ENTITY iiot "http://INTER-IoT.eu/GOIoTP#">
                <!ENTITY iiotex "http://INTER-IoT.eu/GOIoTPex#">
                <!ENTITY medex "http://INTER-IoT.eu/medex#">
                <!ENTITY time "http://www.w3.org/2006/time#">
                <!ENTITY sweet_units
"http://sweet.jpl.nasa.gov/2.3/reprSciUnits.owl#">
                <!ENTITY bc "http://itia.cnr.it/SemanticMiddleware#">
                <!ENTITY healthMeasurement
"http://ontology.UniversAAL.org/HealthMeasurement.owl#">
                <!ENTITY personalHealthDevice
"http://ontology.UniversAAL.org/PersonalHealthDevice.owl#">
                ]
        >
<Alignment
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:sripas="http://www.INTER-IoT.eu/sripas#"
        xmlns="http://www.INTER-IoT.eu/sripas#"
        xmlns:sosa="http://www.w3.org/ns/sosa/"
        xmlns:iiot="http://INTER-IoT.eu/GOIoTP#"
        xmlns:iiotex="http://INTER-IoT.eu/GOIoTPex#"
        xmlns:time="http://www.w3.org/2006/time#"
        xmlns:medex="http://INTER-IoT.eu/medex#"
        xmlns:bc="http://itia.cnr.it/SemanticMiddleware#"
        name="SemanticMiddleware_CO_align" version="1.0.11"
creator="ITIA-CNR" description="Alignment between Semantic middleware
application ontology and INTER-IoT central ontology.">
    <onto1>
        <Ontology about="http://itia.cnr.it/SemanticMiddleware#">
            <formalism>
                <Formalism name="OWL2.0"
uri="http://www.w3.org/2002/07/owl#"/>
            </formalism>
        </Ontology>
    </onto1>
    <onto2>
        <Ontology about="http://INTER-IoT.eu/GOIoTPex#">
            <formalism>
```

```
                <Formalism name="OWL2.0"
uri="http://www.w3.org/2002/07/owl#"/>
            </formalism>
        </Ontology>
    </onto2>
    <steps>
        <step order="1" cell="1_observation"/>
        <step order="2" cell="2_observation"/>
        <step order="3" cell="3_observation"/>
    </steps>
    <map>
        <Cell id="1_observation">
            <entity1>
                <sripas:node_CTX>
                    <rdf:type rdf:resource="&bc;Sensor" />
                </sripas:node_CTX>
            </entity1>
            <entity2>
                <sripas:node_CTX>
                    <rdf:type rdf:resource="&iiot;IoTDevice" />
                </sripas:node_CTX>
            </entity2>
            <relation>=</relation>
        </Cell>
        <Cell id="2_observation">
            <entity1>
                <sripas:node_CTX>
                    <rdf:type rdf:resource="&bc;Machining" />
                </sripas:node_CTX>
            </entity1>
            <entity2>
                <sripas:node_CTX>
                    <rdf:type rdf:resource="&iiot;Service" />
                </sripas:node_CTX>
            </entity2>
            <relation>=</relation>
        </Cell>
        <Cell id="3_observation">
            <entity1>
                <sripas:node_CTX>
                    <rdf:type rdf:resource="&bc;ProductionResource"
/>
                            <bc:hasResourceComponent>
                                <sripas:node_CTZ>
                              <rdf:type
rdf:resource="&bc;ResourceComponent" />
                                </sripas:node_CTZ>
                            </bc:hasResourceComponent>
                </sripas:node_CTX>
            </entity1>
            <entity2>
                <sripas:node_CTX>
                    <rdf:type rdf:resource="&iiot;SoftwarePlatform"
```

	/> <rdf:type rdf:resource="&iiot;IoTDevice" /> <iiot:hasComponent> <sripas:node_CTZ> <rdf:type rdf:resource="&iiot;PlatformComponent" /> </sripas:node_CTZ> </iiot:hasComponent> </sripas:node_CTX> </entity2> <relation>=</relation> </Cell> </map> </Alignment>
Type	TEST of INTEGRATION with IoT components
Setup	Need test setup TS_02 IPSM Need test hook TH_01 Semantic Model (a subset of this model must be aligned)
Start	Invoking the proper function of IPMS Aligner (dashboard : http://grieg.ibspan.waw.pl:3000/translation) and passing it the mapping in the form presented above
Req.	[178], [179], [180]
Input	The following message: { "@graph": [{ "@graph": [{ "@id" : "http://itia.cnr.it/SemanticMiddleware#testSensor", "@type" : "http://itia.cnr.it/SemanticMiddleware#Sensor" }, { "@id" : "http://itia.cnr.it/SemanticMiddleware#testMachining", "@type" : "http://itia.cnr.it/SemanticMiddleware#Machining" }, { "@id" : "http://INTER-IoT.eu/GOIoTP#testPlatformComponent", "@type" : "http://INTER-IoT.eu/GOIoTP#PlatformComponent" }, { "@id" : "http://itia.cnr.it/SemanticMiddleware#testProduction

```
Resource",
  "@type" :
"http://itia.cnr.it/SemanticMiddleware#ProductionReso
urce"
},


{
  "@id" :
"http://itia.cnr.it/SemanticMiddleware#testResourceCo
mponent",
  "@type" :
"http://itia.cnr.it/SemanticMiddleware#ResourceCompon
ent"
},


        {
          "@id":
"http://itia.cnr.it/SemanticMiddleware#testProduction
Resource",
          "@type": [

"http://itia.cnr.it/SemanticMiddleware#ProductionReso
urce"
          ],

"http://itia.cnr.it/SemanticMiddleware#hasResourceCom
ponent": {
                "@id":
"http://itia.cnr.it/SemanticMiddleware#testResourceCo
mponent"
          }
        },



{
  "@id" : "http://itia.cnr.it/Sensor",
  "@type" : [ "http://www.w3.org/2002/07/owl#Class" ]
},
{
  "@id" :
"http://itia.cnr.it/SemanticMiddleware#Machining",
  "@type" : [ "http://www.w3.org/2002/07/owl#Class" ]
},
{
  "@id" : "http://INTER-IoT.eu/GOIoTP#PlatformComponent",
  "@type" : [ "http://www.w3.org/2002/07/owl#Class" ]
}
```

	]
	"@id": "INTER-IoTMsg:payload"
	}
	],
	"@context": {
	"ns": "http://ontology.UniversAAL.org/PhThing.owl#",
	"owl": "http://www.w3.org/2002/07/owl#",
	"INTER-IoTMsg": "http://INTER-IoT.eu/message/",
	"INTER-IoTInst": "http://INTER-IoT.eu/inst/",
	"rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
	"xsd": "http://www.w3.org/2001/XMLSchema#",
	"rdfs": "http://www.w3.org/2000/01/rdf-schema#",
	"INTER-IoT": "http://INTER-IoT.eu/",
	"ns2": "http://ontology.UniversAAL.org/Measurement.owl#",
	"ns1": "http://ontology.UniversAAL.org/Context.owl#",
	"ns4": "http://ontology.UniversAAL.org/Device.owl#",
	"ns3":
	"http://ontology.UniversAAL.org/HealthMeasurement.owl#"
	}
	}

Output	We expect to obtain the following message in order to accept the alignment test: <map><Cell> <entity1 rdf:resource="http://www.opengis.net/gml/Point"/> <entity2 rdf:resource="http://www.w3.org/2003/01/geo/wgs84_pos#Point"/> <measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.99</measure> <relation>=</relation> </Cell></map> The correct response returns the URI of the entities that have been aligned and two fundamental information: the logical relation existing between them (<relation>) and the relative confidence of such relation (<measure>).
Outcome	Pass / Fail

Test output:

- The result is reported in the GUI of the dashboard:

```
{
```

```
  "@graph" : [ {
    "@id" : "INTER-IoT:GOIoTP#PlatformComponent",
    "@type" : "owl:Class"
  }, {
    "@id" : "INTER-IoT:GOIoTP#testPlatformComponent",
    "@type" : "INTER-IoT:GOIoTP#PlatformComponent"
  }, {
    "@id" : "http://itia.cnr.it/SemanticMiddleware#Machining",
    "@type" : "owl:Class"
  }, {
    "@id" : "http://itia.cnr.it/SemanticMiddleware#testMachining",
    "@type" : "INTER-IoT:GOIoTP#Service"
  }, {
    "@id" : "http://itia.cnr.it/SemanticMiddleware#testProductionResource",
    "@type" : [ "INTER-IoT:GOIoTP#IoTDevice", "INTER-IoT:GOIoTP#SoftwarePlatform"
],
    "INTER-IoT:GOIoTP#hasComponent" : {
      "@id" : "http://itia.cnr.it/SemanticMiddleware#testResourceComponent"
    }
  }, {
    "@id" : "http://itia.cnr.it/SemanticMiddleware#testResourceComponent",
    "@type" : "INTER-IoT:GOIoTP#PlatformComponent"
  }, {
    "@id" : "http://itia.cnr.it/SemanticMiddleware#testSensor",
    "@type" : "INTER-IoT:GOIoTP#IoTDevice"
  }, {
    "@id" : "http://itia.cnr.it/Sensor",
    "@type" : "owl:Class"
  } ],
  "@id" : "INTER-IoTMsg:payload",
  "@context" : {
    "ns" : "http://ontology.UniversAAL.org/PhThing.owl#",
    "owl" : "http://www.w3.org/2002/07/owl#",
    "INTER-IoTMsg" : "http://INTER-IoT.eu/message/",
    "INTER-IoTInst" : "http://INTER-IoT.eu/inst/",
    "rdf" : "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
```

```
    "xsd" : "http://www.w3.org/2001/XMLSchema#",
    "rdfs" : "http://www.w3.org/2000/01/rdf-schema#",
    "INTER-IoT" : "http://INTER-IoT.eu/",
    "ns2" : "http://ontology.UniversAAL.org/Measurement.owl#",
    "ns1" : "http://ontology.UniversAAL.org/Context.owl#",
    "ns4" : "http://ontology.UniversAAL.org/Device.owl#",
    "ns3" : "http://ontology.UniversAAL.org/HealthMeasurement.owl#"
  }
}
```

T34.62.9 (TEST of INTEGRATION with IoT components) Application Ontology imports GOIoTP

ID	T34.62.9
Test	GOIoTP ontology are imported within the Application Ontology used to represent knowledge for the scenario S34.
Type	TEST of INTEGRATION with IoT components
Setup	Need test setup TS_03 GOIoTP Need test tool TT_01 RDF store Need test hook TH_01 Semantic Model
Start	The following subsequent steps will be carried out: 1. Add the import directive in the AO ontology directed to the GOIoTP ontology, so that all the statements of the latter are imported in the former ontology; 2. After the concept alignment has been executed between AO and GoIoTP, the alignment results are used in order to create logical relation axioms within the integrated ontology (e.g., equivalentClass axioms, subClassOf, etc.).
Req.	[42], [96]
Input	Instantiate an ontological individual as an instance of a specific class of the AO ontology, which is equivalent to a class imported from the GoIoTP ontology.
Output	Test if ontological individual inherits the features of the equivalent class.
Outcome	Pass / Fail

Test output:

3.3.10.6 Test outcome overview

Test outcome

The following table will provide an overview of the test result of all the performed tests in this SAT.

Test	Description	Outcome
T31.62.1	Information published by Virtual Sensor are persisted	Pass / Fail
T31.62.2	Information updated by Virtual Sensor are received by the subscribed clients	Pass / Fail
T31.62.3	Information updated by Optimizer are received by Virtual Carriage	Pass / Fail
T31.62.4	Updates concerning information on which no client is subscribed	Pass / Fail
T31.62.5	(TEST of INTEGRATION with IoT components) Connection with the Bridge (reading a device steraming data flow)	Pass / Fail
T31.62.6	(TEST of INTEGRATION with IoT components) The connection with the Semantic Middleware Bridge (actuating over a controlled device)	Pass / Fail
T31.62.7	(TEST of INTEGRATION with IoT components) The connection with a ThirdParty platform connected to the INTER-MW (reading a device steraming data flow)	Pass / Fail
T31.62.8	(TEST of INTEGRATION with IoT components) The ontology alignment test through IPSM	Pass / Fail
T31.62.9	(TEST of INTEGRATION with IoT components) Ontology import Text	Pass / Fail
FAT Outcome		**Pass / Fail**

Table 73. Test outcome overview

3.3.10.7 Integration ethics and security

Introduction

This section addresses recommendations and proposed solutions to the ethical and security points of the integration.

For each pilot the ethics is discussed in paragraphs 8.2 until 8.5. The security aspects of each layer is discussed in paragraph 8.7 and 8.8.

The information for the pilots for both ethics and security comes from the partners and may be included in other documents as well.

Semantic Middleware

The various steps needed for the execution of these FAT are not connected with major ethical concerns. In particular, the components that make up the proposed semantic middleware and data sets managed in these tests do not deal with sensitive personal data. Therefore, just a limited attention should be given to the drawn of a code of ethics.

Under a generic perspective, Semantic Middleware allows a new way of combining and integrating different data streams. As analyzing and acting on insights from these data can introduce new classes of risks of unethical or even illegal use of insights, it is necessary to tackle ethical challenges that can emerge, analyzing a proper risk mitigation. The idea is creating a code of data ethics leveraging the Universal principles for data ethics—Guidelines introduced in [Ref2].

Concerning the security, no particular attention is required. RDF STORE and Semantic Middleware expose an authenticated access and for this reason test credentials will be provided to access RDF store and to access Semantic Middleware.

3.3.11 Third Party: SecurIoTy

Concerns about information security are one of the main reasons for companies and private individuals not to adopt cloud services and to be sceptical about IoT systems. On top of concerns regarding physical- and cyber-attacks, international corporations additionally carry legal attacks in their threat model. However, cloud services are essential in improving efficiency and cost structures.

SecurIoTy addresses that gap. SecurIoTy combines a number of security mechanisms to protect data and to address all relevant security dimension such as confidentiality, integrity and availability. We use CloudRAID, fragmentation and encryption. CloudRAID means that data is fragmented, the fragments are encrypted and the encrypted fragments are redundantly distributed to multiple independent storages. SecurIoTy is storage agnostic, i.e. the data fragments may be distributed across multiple jurisdictions adding additional security.

SecurIoTy solves security and compliance issues when sending and sharing data via public networks like the internet and when storing data in cloud services, thus enabling companies to use cloud based IoT services, which they would not use without SecurIoTy protection.

SecurIoTy is crypto proxy technology and as such won't interfere with the user experience or with processes. SecurIoTy users will get to keep their established usage pattern and processes and also keep their legacy infrastructure. SecurIoTy aims to integrate seamlessly. SecurIoTy can be operated off-premise, on-premise or hybrid.

SecurIoTy is based on DocRAID® - a storage system which offers distributed, secured storage and data transfer. To operate SecurIoTy we maintain a geo-redundant high availability computing cluster. We operate storage capacities spread among different European computing centres, among others in Germany and France. SecurIoTy provides highly secure storage based on the principles of fragmentation and encryption. This means no one (1) storage knows all the information to recompile a document.

Within the INTER-IoT framework we identify

(1) technical,
(2) legal and
(3) organizational

challenges. In the technical category, we can further identify challenges at the

(1) networking,
(2) middleware,
(3) application,
(4) interoperability and
(5) security

issues.

In the current version of SecurIoTy we focus on and test requirements from these categories:

Category		Tested in current version
Technical	networking	Yes
	middleware	No
	application	Yes
	interoperability	YES
	security	Yes

Table 74: test categories

SecurIoTy offers HTTP(S) interfaces and offers interfaces to connect to cloud storage services which are operated by AvailabilityPlus (DocRAID® CloudRAID).

A detailed description of the System can be found here: **DocRAID server guide E v1.20.pdf**

From the architecture point of view, we have established and tested these components:

Category		Tested in current version
Distributed storage	Storage 1	Yes
	Storage 2	Yes
	Storage 3	Yes
Key Storage	Local storage	Yes
	HSM – hardware security module	No
Controller	One controller at one site	Yes
	Multiple distributed controllers at multiple sites	No
Frontend	HTML	No
Secure Gateway	Gate Keeper	Yes
	Load Balancer	No
	Firewall	No
Frontend access	HTTPS	Yes
	WebDAV	Yes

Table 75: tested components

The following paragraph gives a brief introduction of the implemented security measures and potential vulnerabilities.

Security Level	Use Case	Measures / Best Practice	Potential Vulnerability
Low to very high	All	Files are encrypted by AES256	weak password, security is directly related to the strength of the password
Very high	All	Two different random number generators are used. The random number generator is	It was reported that the random number generator Dual_EC_DRBG contains a

		modularized; upon request customer specific modules can be used.	potential backdoor.
Medium	Single User	Strong user generated password and Keyfiles	weak password, keyfiles stored on local machine
High	Single User	User generated strong password and Keyfiles; Keyfiles stored on external device	weak password, social engineering
Low to very high	Multi User	Admin generated password and Keyfiles	weak password, keyfiles stored on local machine
Low to very high	Multi User	Master Key File must be stored in safe place	Master Key File in an unsecure place
Medium	Multi User	Secret keys to access files are stored by default in the Windows key container	Windows key container can potentially be hacked or contain backdoors
Very high	Multi User	Secret keys to access files are stored on an external protected device, e.g. crypto stick	social engineering
Very high	Multi User	Secret keys to access files in a workspace are exchanged based on the Diffi-Hellman key exchange, i.e. perfect forward secrecy	Currently no backdoor known to hack Diffi-Hellman
Medium	Multi User	User is activated by admin after request. For very high security user verification is required.	Workspace file could be intercepted, e.g. if sent by email. User verification by digital handshake supports user verification.
Very high	Multi User	Digital handshake for user verification.	A man-in-the-middle attack would generate an additional request visible to the admin

Table 76: security measures

3.3.11.1 Integration of IoT framework

The proposed approach will complement the INTER-IoT Architecture and will provide industry standard interfaces to integrate security as necessitated by the respective application. A high degree of interoperability is achieved by adhering to standard protocols like HTTPS(S) and WebDAV (TCP) and by integration of widely used cloud services. In contrast to current approaches to IoT security which mainly focus on single aspects of IoT security, SecurIoTy provides a single framework to cover scalable security from the device level to the application level and which covers all dimensions of security such as confidentiality, integrity and availability (CIA). Put to work in the logistics use case (INTER-logP), SecurIoTy will push the envelope of IoT security well beyond the state of the art.

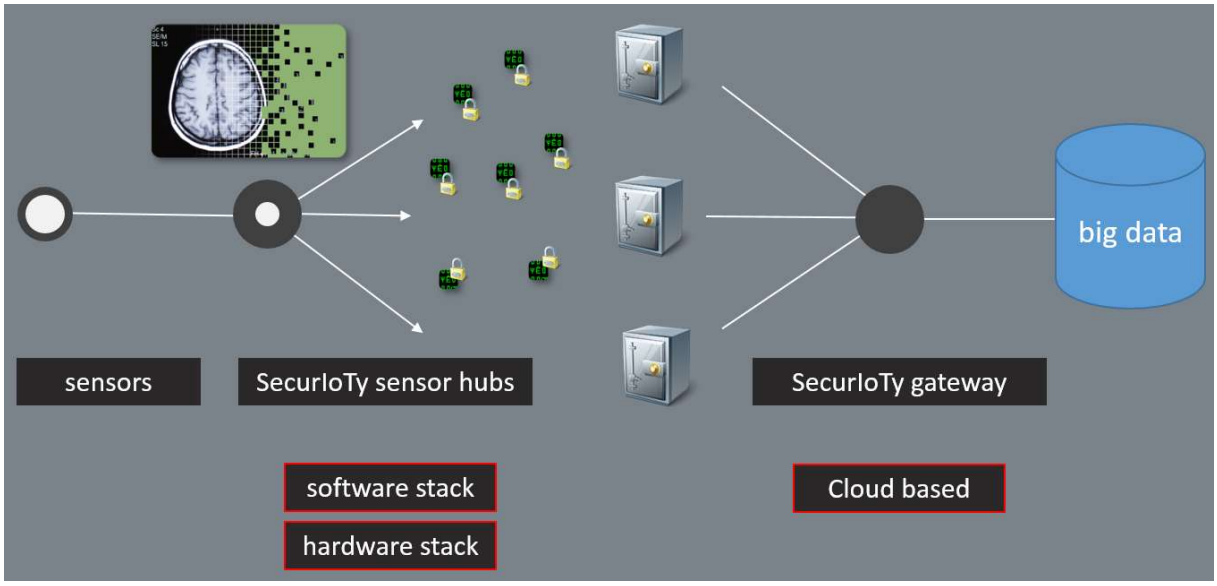


Figure 104: SecurIoTy overview of the solution architecture

SecurIoTy can be deployed as

(1) a sensor hub, collecting data from sensors directly or
(2) alternatively, can be set up as a gateway receiving data from sensor hubs and deliver that data to a data storage.

SecurIoTy could be hosted on a sensor hub, which is typically a piece of hardware. In this scenario SecurIoTy would be part of the hardware stack.

Within the scope of INTER-IoT we will deploy SecurIoTy as a cloud-based gateway.

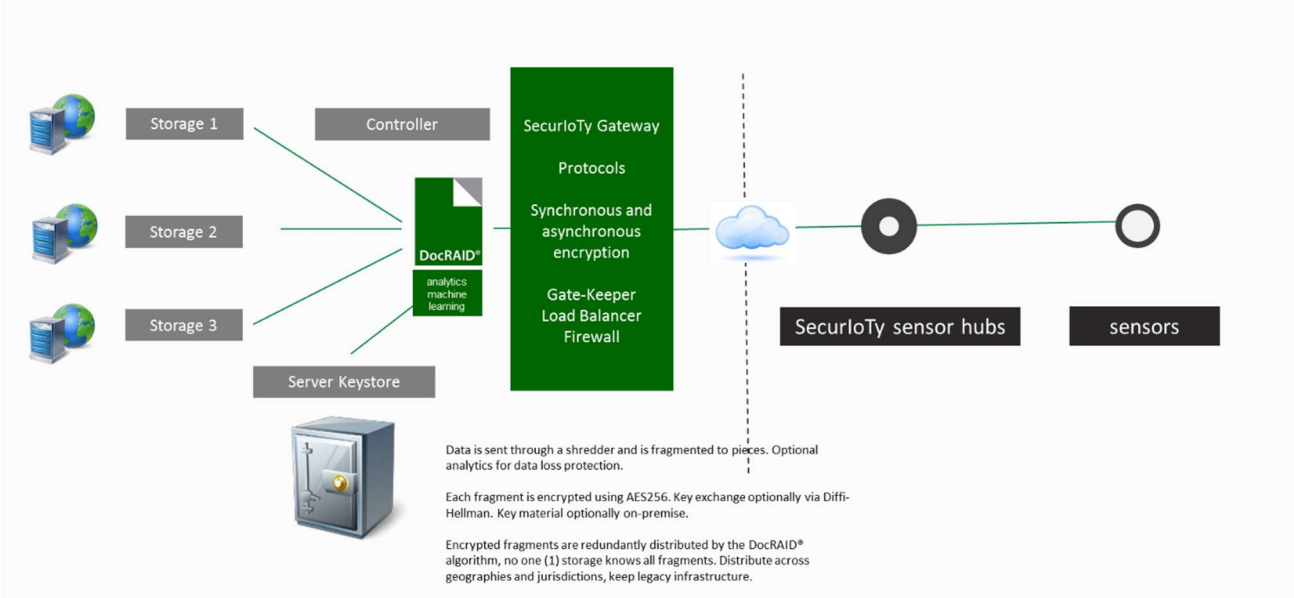


Figure 105: SecurIoTy architecture with the DocRAID crypto proxy

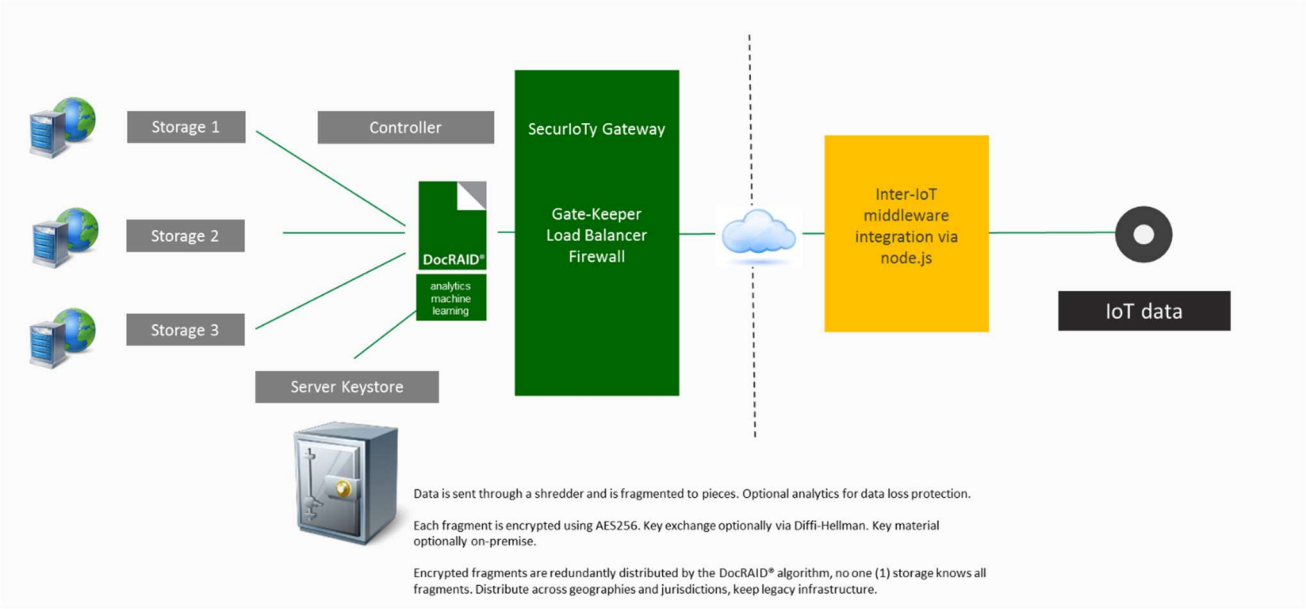


Figure 106: SecurIoTy architecture with INTER-IoT middleware integration

The DocRAID crypto proxy works in three phases:

1. Fragmentation

Data is sent through a shredder and fragmented to pieces.

2. Encryption

Each fragment is encrypted using AES256. Key exchange optionally via Diffi-Hellman.

3. RAID distribution

Encrypted fragments are redundantly distributed by the DocRAID® algorithm, no one (1) storage knows all fragments. Distribution across geographies and jurisdictions, keep legacy infrastructure.

SecurIoTy is designed to accept any communication from middle ware nodes. This could be JSON, XML or any binary format.

The use case will include storing sensitive data from the port authority such as transport orders and gate access data in a secure repository which will be SecureIoTy.

3.3.11.2 Deliverables and version overview

The following table contains a deliverable list which need to be signed of before SAT testing commences.

ID	Description	Check
Documents		
1	Validation and Test reports of the IoT system components	OK
2	Validation and Test reports of the Pilot system components	
Tools		
7	PortSwigger Burp	OK

Table 77: Deliverable checklist

The following table shows the software components and version of which the system release version SecurIoTy 18.07.0201 consists of.

ID	Description	Version	Check
Controller			
10	DocRAID controller	18.07.0401	OK
Secure Gateway			
20	Gate Keeper	17.01.3008	OK
Bridge			
30	Node-red bridge	1.00	OK

Table 78: Component version overview

3.3.11.3 Requirements, scenarios and use cases

The following table provides an overview of the relation between the requirements and the test(s) that validate their implementation.

Requirement-IDs refer to IDs defined in the document “D2.3_INTER-IoT_Requirements-and-business_v1.2.pdf”

ID	Description	Covered by
API		
243	Gateway access API	TC200010, TC200020 TC200030, TC200040
264	API allows create/update/remove users	TC200090, TC200100
Interoperability		
56	Secure synchronization	TC300000, TC300010
Performance		
72	Communication should be done using protocols that are efficient in terms of amount of exchanged information over message size	TC100010

Security		
27 30	System security	TC401010, TC401020 TC401030, TC401040 TC401050, TC401060 TC401070
28 37	System privacy	TC401000
95	Robustness, resilience and availability	TC403000, TC403010
98	Data provenance	
261	A user knows its permissions	TC409000, TC409005
263	Access to personal data needs to be previously authorized	TC409010, TC409020 TC409030, TC409040 TC409050, TC409060 TC409070, TC409080 TC409090, TC409100 TC409110
Non-functional requirements		
47	API for third-party developers	TC200050, TC200060
58	Auditability and Accountability	TC500010, TC409060 TC500030, TC500040
60	AutoLogin	TC200070, TC200080
63	Provision of authentication credentials	TC406000, TC406010 TC406020, TC406030 TC406040
68	Logging	TC500050
69	Confidentiality, Avoid data falsification or disclosure	TC404000, TC403000 TC401000
94	Supports multiplatform	TC300020
Architecture		
36	Scalability. Computing resources	TC100010, TC100020 TC100030, TC405000 TC405010, TC405020

Table 79: Requirements vs. test mapping

The following table provides an overview of the relation between the scenarios and the test(s) that validate their implementation.

ID	Scenario name	Covered by
2	IoT support for transport planning and execution	
3	IoT Weighbridges	TC200010, TC200020 TC200030, TC200040 TC200050, TC200060
4	Monitoring reefer container	TC200010, TC200020 TC200030, TC200040 TC200050, TC200060
5	Monitoring of containers carrying sensitive goods	TC200010, TC200020 TC200030, TC200040 TC200050, TC200060
6	Dynamic lighting in the port	TC200010, TC200020 TC200030, TC200040

		TC200050, TC200060
18	Containership is entering the harbour region	TC200010, TC200020 TC200030, TC200040 TC200050, TC200060
20	Damage or problems to the container during shipment	TC200010, TC200020 TC200030, TC200040 TC200050, TC200060
32	Third party developer using INTER-FW to access data from two different platforms	TC200010, TC200020 TC200030, TC200040 TC200050, TC200060

Table 80: Scenario vs test mapping

3.3.11.4 Test environment

Introduction

To test the functionality of the integrated INTER-IoT in combination with the IoT framework representative test hooks in the system are needed. This chapter will describe this environment and the used hardware, software, tools and platforms.

Test environment

This paragraph describes the test setups, hooks and probes used in the system used during this SAT.

Test tools, hooks and probes

This paragraph describes the test setups, tools, hooks and probes used in the by this document defined tests.

A test setup defines a setup used for a test, and describe the used system parts, interface etc.

Tools define the used COTS tooling that will be used during testing e.g. Logic analyzers, packet sniffer like e.g. Whireshark, developed scripts, etc.

Hooks are used to inject data or control parts of the system used to get the system to handle a needed scenario. This can be hidden, debug or service interfaces which contain the needed functionality.

Probes are used to gather logging from parts of the system under test to provide system feedback and to prove that the system is handling the scenario as it should.

The following paragraphs will define the used setups, tools, hooks and probes which will be referred to by the test descriptions.

TS_01 Test setup

This paragraph describes the test environment and the system setup used during this SAT.

The test system contains these elements:

(1) DocRAID controller
(2) Secure gateway
(3) Keystore
(4) Storages
(5) Burp Scanner

(6) Node-red test node

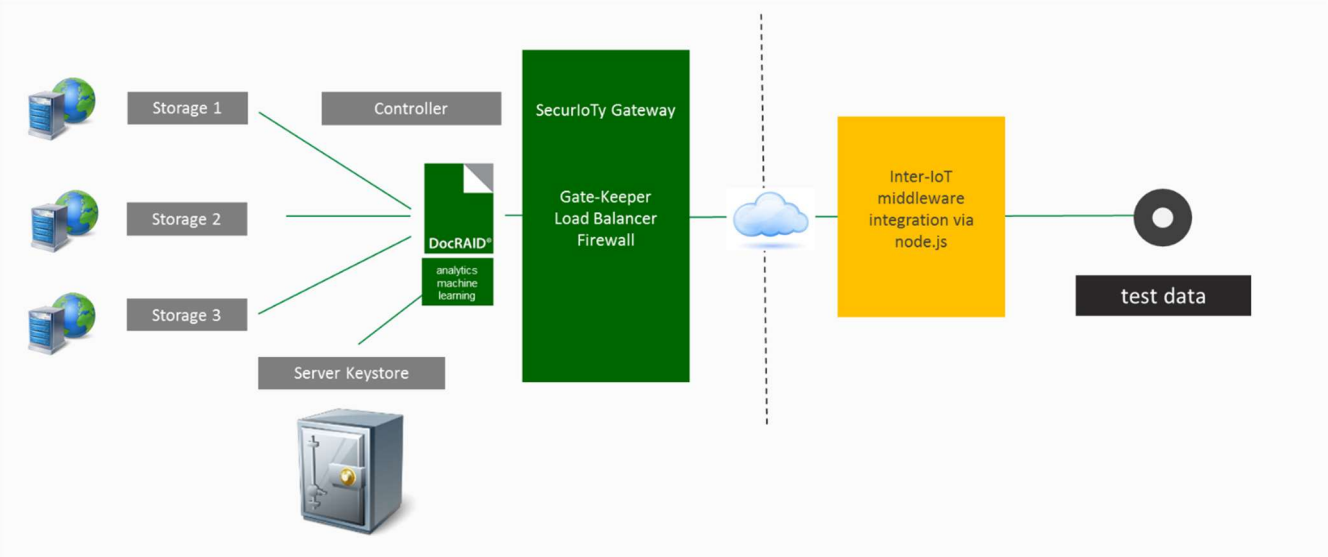


Figure 107: Test architecture

In the test scenarios the controller and the storages reside on one (1) physical server. The SecurIoT Gateway resides on a second physical server in a different network. The test environment is set up using these components:

The server side:

- CPU: 4x Intel Xeon E5-26xx (Sandy Bridge) @ 2.1 GHz
- RAM: 16 GB
- Ethernet: Red Hat VirtIO Ethernet Adapter
- Operating systems:
  - Windows 2012 R2 Server x64

The secure gateway:

- CPU: 4x Intel Xeon E5-26xx (Sandy Bridge) @ 2.1 GHz
- RAM: 16 GB
- Operating systems:
  - Debian 9.2
  - Nginx community edition

Libraries

- Microsoft .net environment 4.51

No additional libraries necessary.

SecurIoTy acceptance test includes sections covering

(1) Acceptance criteria
(2) Severity and priority of bugs
(3) Reporting process of bugs
(4) Test Environment

(5) Test cases

SecurIoTy relevant test cases are classified into these test type:

Test type	Acronym
Manual	MN
Automated	AU
To be done	tbd

Table 81: Requirements vs test mapping

We refer to the requirements given in **D2.3_INTER-IoT_Requirements-and-business_v1.2**

Acceptance criteria

During this project we will deliver prototypical developer (dev) implementations. We define Alpha-, Beta-and release-Versions in the following paragraphs. The quality standards are defined as given in the following tables based on a maximum of allowable errors and severity levels.

Developer Version (Dev)

Developer versions are snap shots of the current developer process. A number of tests should have been passed and documented in test cases. However, there are no formal release criteria defined.

Alpha Version

An alpha version delivers the number of features required for this state. It is feature complete. In the following table the acceptance criteria are defined for the release.

Criticality	Maximum number of Test Cases
Critical	5
Important	10
Low	30
Trivial	60

Table 82: Alpha version quality criteria

Beta Version

A beta version delivers the number of features required for this state. It is feature complete. In the following table the acceptance criteria are defined for the release. Features will not be added anymore. Quality is in the focus now.

Criticality	Maximum number of Test Cases
Critical	0
Important	5
Low	20
Trivial	40

Table 83: Beta version quality criteria

Release Version

A release version delivers is fully featured and complies with the quality standards defined. Quality is defined in the following table.

Criticality	Maximum number of Test Cases
Critical	0
Important	0
Low	15
Trivial	30

Table 84: Release version quality criteria

The software is delivered if all acceptance criteria are met during the tests. Per iteration only those features will be test which are relevant for the current iteration and delivery plan.

Criticality and Priority

Bugs found during testing will be reported in the internal sprint-log (bug tracking). Bugs are classified into criticality and priority. The classification system is given in the following table.

Criticality	Description
Critical	The application, major parts of the application or major features are not available or will crash the system The testcases have not been met.
Important	Important parts of the application are not available or have not passed the tests. There is a workaround to provide the same/similar functionality.
Low	Some functions and features do not work according to the specification. There is a workaround. There is no major disadvantage in using existing workarounds.
Trivial	The application runs smoothly. Changes are made for improved efficiency or for cosmetic reasons.
Feature request	This not a bug rather a new request. Here requirements are reformulated, changed or added.

Table 85: Criticality description

Feature requests have been added to include a way to specify useful extensions. A decision has to be made if this is handled as a change request.

In addition to criticality a bug can be given a priority. Bugs with higher priority will be handled earlier.

Priority	Description
Urgent	Bug handling must be initiated immediately. A patch should be provided asap.
High	Bug handling must be pursued with high priority. The bug can be addressed during the next release.
Medium	Bug can be addressed in one of the next releases.
Low	Bug can be addressed as soon as there are resources available.

Table 86: Priority description

TT_01 Test tool

Burp Suite is a Java based testing framework. It has become an industry standard suite of tools used by information security professionals, software and penetration testers. Burp Suite helps you identify vulnerabilities and verify attack vectors that are affecting applications.

TH_01 Test hook Middleware

The SecurIoTy node provides an API for login, logout, send and retrieve data and to control the data ID. Login data and user management is provided through a web frontend.

The SecurIoTy server can be found at this address: **https://INTER-IoT.docraid.com/**

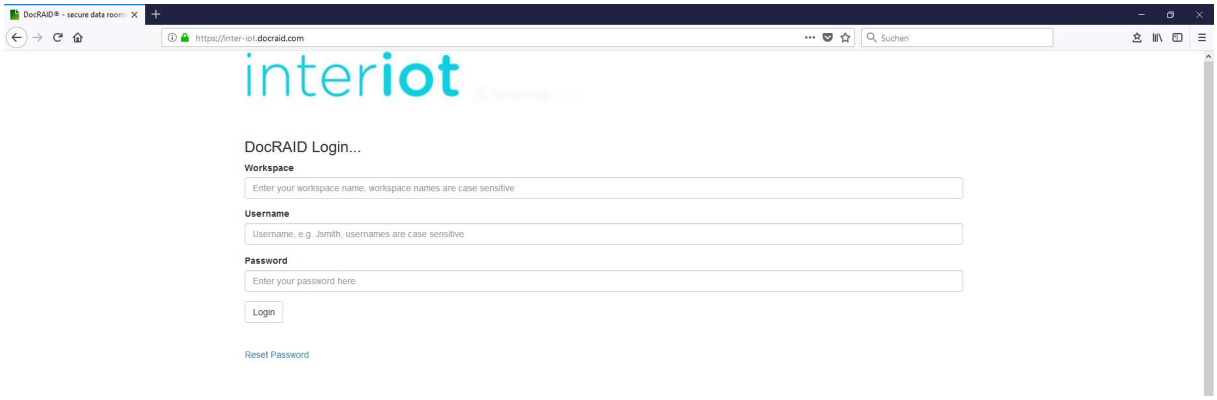


Figure 108: SecurIoTy server

The SecurIoTy server provides features for security, logging and access management.

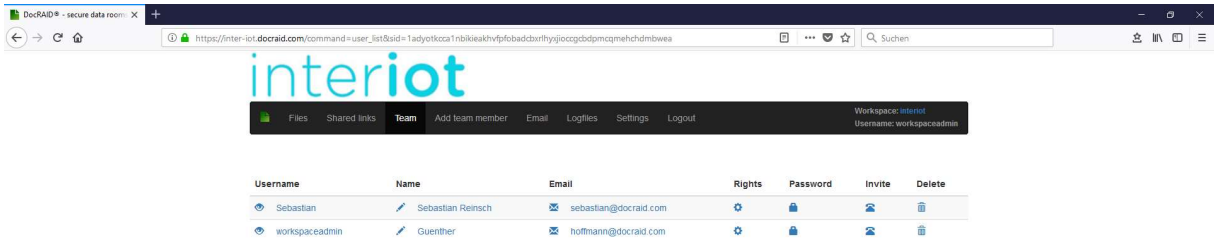


Figure 109: SecurIoTy access management

TP_01 Test probe Binary Data

Test data is produced by the node.js middleware test node and is injected into the SecurIoTy node. Data consists of random binary strings.

3.3.11.5 Test description

Architecture

ID	Name	Description	Reference	Status
TC10xxxx	**Architecture**			
TC100010	Scalability. Computing resources	Use load generator to retrieve data (1) Test with 50 clients (response time < 1000 msec) (2) Test with 250 clients (response time < 1000 msec) (3) Test with 500 clients (response time < 2000 msec) Protocols: use HTTPS and WebDav	REQ3 REQ72	Pass / Fail
TC100015		Use a test node to send and retrieve data from SecurIoTy every (1) 5 seconds (2) 1 second	REQ54	Pass / Fail
TC100020		Add storage to system / hot spare while system is operational.		Pass / Fail
TC100030		Remove storage from system / hot spare while system is operational.		Pass / Fail

API

ID	Name	Description	Reference	Status
TC20xxxx	**API**			
TC200010	Gateway access API	A list of exposed functions can be found in the technical documentations. All exposed functions can be accessed and are covered by the security gateway. Non-allowed, non-exposed and non-existing function calls are blocked	REQ243	Pass / Fail
TC200020		Send unlisted commands to systems, gateway must not send data to controller and will show an error message		Pass / Fail
TC200030		Send listed commands to systems, gateway must send data to controller and will show content		Pass / Fail

TC200040		Send listed commands to system with non-plausible added parameters, gateway must not send data to controller and will show an error message		Pass / Fail
TC200050	API for third-party developers	A list of exposed functions can be found in the technical documentations. HTML frontend is available.	REQ47	Pass / Fail
TC200060		Have a third party integrate the API Node-Red writes and reads data to and from the API		Pass / Fail
TC200070	AutoLogin	A list of exposed functions can be found in the technical documentations. Autologin is one of the exposed functions. Alternative implementations are available. User can assign rights and expiration dates.	REQ60	Pass / Fail
TC200080		Have a third party integrate the API Node-Red writes and reads data to and from the API		Pass / Fail
TC200090	API allows create/update/remove users	A list of exposed functions can be found in the technical documentations. Create/update/remove is part of the exposed functions.	REQ264	Pass / Fail
TC200100		Have a third party integrate the API Node-Red writes and reads data to and from the API		Pass / Fail

Interoperability

ID	Name	Description	Reference	Status
TC30xxxx	**Interoperability**			
TC300000	Secure synchronization	Synchronization is handled by publicly available time servers, here: ptbtime1.ptb.de	REQ56	Pass / Fail
TC300010		Time server on OS level must be set, check in best practice		
TC300020	Supports multiplatform	Needs clarification. SecurIoTy runs natively on Windows Server 2012 R2. Virtualized it runs on any Host including Linux. Clients can be operated from any OS. Verified on Linux Node-Red (Bridge) and Windows Server 2012	REQ94	Pass / Fail

		(Controller)		
		HTTP and WebDAV access has been exposed to the calling node, these protocols can be used by legacy systems older than 2010.	REQ193	Pass / Fail

Privacy/Security

ID	Name	Description	Reference	Status
TC40xxxx	**Privacy / Security (CIA – confidentiality, integrity, availability)**			
TC401000	Sensitive data is stored according to national and EU policies	Third parties cannot access private data or unauthorized data within the SecurIoTy system. Data protection meets the national and European policies. Data security has been tested against German PersDat, §203 StGB, AO. Zero- knowledge data storage: admins and other personal with access to the physical storage have no access to clear text information. Separation of content and operations: admins and other personal with access to the physical storage have no access to clear text information. Fragmentation of content: content is fragmented and distributed across multiple storages. No one storage has all knowledge to recompile a document. Cryptography: fragments are encrypted by AES 256 Resilience: RAID 5-3 concepts protects against failure of a storage and against manipulation of content. Default is that 1 of 3 storages	REQ30 REQ27	Pass / Fail

		may fail. House many clients on one system. Each client has got its own key material.		
TC401010		Read backend data from storage and verify it is encrypted		Pass / Fail
TC401020		Send encrypted fragments though a decryption tool and test if it be can broken		Pass / Fail
TC401030		Remove storage while system is active, system must keep running		Pass / Fail
TC401040		Add storage while system is active, system must add fragments and re-initiate the original state		Pass / Fail
TC401050		Remove keys from keystore, system must not deliver any files anymore		Pass / Fail
TC401060		Alter content of keyfiles in keystore, system must not deliver any files anymore		Pass / Fail
TC401070		reinstall keyfiles in keystore, system must deliver files		Pass / Fail
TC402000	Privacy	See TC401000	REQ37 REQ28	
TC403000	Robustness, resilience and availability	See TC401000 Failure of storage will be compensated by RAID principle; redundant storages will cover and provide fall back. No one (1) storage knows all the fragments to recover a document, if a storage is compromised, the attacker will not gain access to the content even if the attacker is capable of breaking the encryption. The architecture component “security gateway” will shield the controller from non-conform traffic. Security gateway will filter requests and will only let exposed function-calls pass.	REQ95	Pass / Fail

		Optionally to increase availability and performance controllers may be duplicated and spread across a controller farm. The secure gateway will act as a load distributor. If a controller fails or is attacked, spare controller can take over.		
TC403010		Attack the system with an automated attack tool which is fed with current vulnerabilities, result must not show any vulnerability		Pass / Fail
TC404000	Confidentiality	TC401000 TC403000	REQ69	
TC405000	Avoid data falsification or disclosure	Detailed rights on a user level grant access to specific data only. Data manipulation at the backend will be detected by the DocRAID parity checks. Multitenant capable.	REQ36	Pass / Fail
TC405010		Run multiple tenants on same system and verify that data from one tenant cannot be seen by another tenant from the frontend.		Pass / Fail
TC405020		Run multiple tenants on same system and verify that data from one tenant is different than from another tenant. Use same date entry from front end and verify that backend fragments show different content.		Pass / Fail
TC406000	Provision of authentication credentials	Authentication credentials consisting of a User ID and an authentication device, e.g. Password must be given to gain access. Additionally, a second access code may be required, a second factor, this can be a SMS or an Email.	REQ63	Pass / Fail
TC406010		Try to login with false credentials, system must deny access		Pass / Fail
TC406020		Try to login with no credentials, system must deny access		Pass / Fail
TC406030		Try to login with correct credentials, system must grant access		Pass / Fail
TC406040		Leave session open but unattended for > timeframe set at controller, system must deny access, session has ended, user must logon again.		Pass / Fail

TC409000	A user knows its permissions	User may retrieve permissions	REQ261	Pass / Fail
TC409005		Login and retrieve user permissions at the user terminal		Pass / Fail
TC409010	Access to personal data needs to be previously authorized	User may retrieve data by permissions only. Permission can be set to invalidate automatically after a certain time.	REQ263	Pass / Fail
TC409020		Admin grants / denies management right, user can/cannot manage the system		Pass / Fail
TC409030		Admin grants / denies file access rights, user can/cannot read/write files		Pass / Fail
TC409040		Admin grants / denies file share rights, user can/cannot share files		Pass / Fail
TC409050		Admin grants / denies history (version) access rights, user can/cannot read/write files		Pass / Fail
TC409060		Admin grants / denies report access rights, user can/cannot access reports		Pass / Fail
TC409070		Admin grants / denies WebDAV access rights, user can/cannot WebDAV		Pass / Fail
TC409080		Share a file, recipient must be able to retrieve file without credentials		Pass / Fail
TC409090		Share a directory, recipient must be able to retrieve directory without credentials		Pass / Fail
TC409100		Share a file with an expired date, recipient must not be able to retrieve file, error message will be shown		Pass / Fail
TC409110		Share a folder with an expired date, recipient must not be able to retrieve folder, error message will be shown		Pass / Fail
TC410010	**Encryption**			
	Filename encryption	Do not encrypt filenames: The known-plaintext attack (KPA) is an attack model for cryptanalysis where the attacker has samples of both the plaintext (called a crib), and its encrypted version (ciphertext). These can be		

		used to reveal further secret information such as secret keys and code books.		

Compliance (Usability)

ID	Name	Description	Reference	Status
TC50xxxx	**Compliance**			
TC500010	Auditability and Accountability	All user actions are logged into a logfile and can be retrieved with the appropriate rights. Log files can be set to autodelete after a certain period of time, e.g. to be compliant with national law.	REQ58	Pass / Fail
TC500020		TC409060		
TC500030		Set autodelete to 1 (one) day, reports with age > 1 day must be deleted by the system		Pass / Fail
TC500040		Set autodelete to 1 (one) week, reports with age > 1 week must be deleted by the system		Pass / Fail
TC500050	Logging	See TC500010	REQ68	Pass / Fail

3.3.11.6 Test outcome overview

The following table will provide an overview of the test result of all the performed tests in this SAT.

Architecture

ID	Name	Status
TC10xxxx		
TC100010	Scalability. Computing resources	Pass / Fail
TC100020		Pass / Fail
TC100030		Pass / Fail

Table 87: Test outcome overview: Architecture

API

ID	Name	Status
TC20xxxx		
TC200010	Gateway access API	Pass / Fail
TC200020		Pass / Fail
TC200030		Pass / Fail
TC200040		Pass / Fail
TC200050	API for third-party developers	Pass / Fail
TC200060		Pass / Fail
TC200070	AutoLogin	Pass / Fail
TC200080		Pass / Fail
TC200090	API allows create/update/remove users	Pass / Fail
TC200100		Pass / Fail

Table 88: Test outcome overview: API

Interoperability

ID	Name	Status
TC30xxxx		
TC300000	Secure synchronization	Pass / Fail
TC300010		
TC300020	Supports multiplatform	Pass / Fail

Table 89: Test outcome overview: Interoperability

Privacy/Security

ID	Name	Status
TC40xxxx		
TC401000	Sensitive data is stored according to national and EU policies	Pass / Fail
TC401010		Pass / Fail

TC401020		Pass / Fail
TC401030		Pass / Fail
TC401040		Pass / Fail
TC401050		Pass / Fail
TC401060		Pass / Fail
TC401070		Pass / Fail
TC402000	Privacy	
TC403000	Robustness, resilience and availability	Pass / Fail
TC403010		Pass / Fail
TC404000	Confidentiality	
TC405000	Avoid data falsification or disclosure	
TC405010		Pass / Fail
TC405020		Pass / Fail
TC406000	Provision of authentication credentials	
TC406010		Pass / Fail
TC406020		Pass / Fail
TC406030		Pass / Fail
TC406040		Pass / Fail
TC409000	A user knows its permissions	Pass / Fail
TC409005		Pass / Fail
TC409010	Access to personal data needs to be previously authorized	
TC409020		Pass / Fail
TC409030		Pass / Fail
TC409040		Pass / Fail
TC409050		Pass / Fail
TC409060		Pass / Fail
TC409070		Pass / Fail
TC409080		Pass / Fail
TC409090		Pass / Fail
TC409100		Pass / Fail
TC409110		Pass / Fail

Table 90: Test outcome overview: privacy/ security

Compliance (Usability)

ID	Name	Status
TC50xxxx		
TC500010	Auditability and Accountability	
TC500020		
TC500030		Pass / Fail

TC500040		Pass / Fail
TC500050	Logging	Pass / Fail

Table 91: Test outcome overview: compliance

3.3.11.7 Integration ethics and security

Introduction

This section addresses recommendations and proposed solutions to the ethical and security points of the integration.

SecurIoTy

A major purpose of SecurIoTy is to mitigate ethical issues and security risks. For example, a typical requirement in such a scenario would be to separate the operations from the content. In a typical data storage and/or transport scenario this requirement can hardly be upheld. This problem is known as the “privileged account problem”, i.e. a privileged person, e.g. an administrator would typically have access to sensitive data stored and transported by a system. SecurIoTy solves this problem and would thus be a solution for potential ethical issues arising in the context of IoT.

3.3.12 Third Party: E3City

This project integrates E3Tcity devices with the Device Layer of INTER-IoT Inter Layer Platform. E3city has his own platform, this platform is in production stage that is being used in more than 20 towns in Spain. This development will provide INTER-IoT with a whole device/cloud/app vertical solution to be applied to the Smart Port pilot and any project in general.

These are main objectives:

- Connect E3Tcity devices to the Device layer of INTER-IoT, so that cross interaction can be used in further stages.

- Use E3Tcity devices to provide Smart Lighting features to Valencia Port, such as lighting control, power consumption, climatic sensors, movement detection and lightness level.

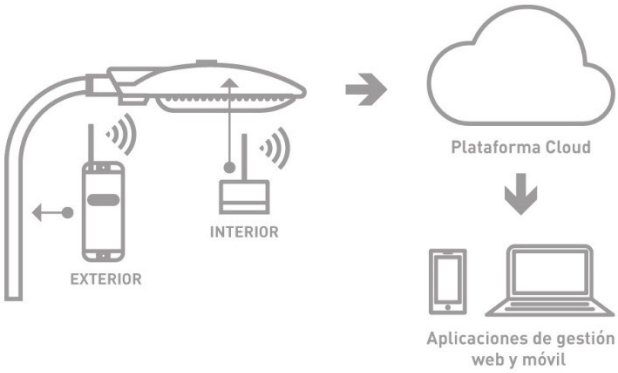


Figure 110: System e3tcity description

The system consists of three elements:

- **E3Tcity drivers**
  All smart city controllers have a MAC address and they communicate with the cloud via Wi-Fi or cellular networks. They are also endowed with intelligence that allows to monitor continuously energy parameters. With this info they can also control their power consumption and detect faults and cable theft alarms.

- **Cloud platform**
  Drivers exchange information in real time with delay of around 1 second with the cloud platform to provide information on their sensors measures and to be able to interact with the system manager. You can set the volume of data exchange Generated data are sent to the cloud and store for a time in the device memory, ensuring their integrity even in case of failure or sabotage.

- **Users applications**
  Users' applications are available for PC and Smartphone. They allow users to know the state of offered services in real time and from anywhere.

This system is in production phase and ready to implement

3.3.12.1 Integration of IoT framework

The following table provides a description of the components of the IoT framework will be integrated in this pilot and witch interfaces are used.

Components	Interfaces	Test
LC_10 (This device will control the sensing part)	Protocol Modbus + INTER-Middleware	If possible activate the AC OUT 110-230 V from the application? The device executes the orders of regulation 0-10 V from the application? The device connects to the router configured in the application?
	Platform e3tcity + INTER-Middleware	The device connects correctly with e3tcity cloud? When you remove the power several times from the device, does it continue to connect with the platform? If you press on repeatedly the device does not lock and allows you turn it on and off again?
LS_10 (This device will control the lights on and off)	Protocol Modbus + INTER-Middleware	If possible activate the AC OUT 110-230 V from the application? The device executes the orders of regulation 0-10 V from the application? The device connects to the router configured in the application? The device connects correctly with e3tcity cloud? The six relay control outputs work by activating them from the application?
	Platform e3tcity + INTER-Middleware	Are the TOTAL POWER ACTIVE (W) measurements monitored correctly from the application? Are the TOTAL POWER REACTIVE (VAr) measurements monitored correctly from the application? Are the Voltage(V) and Intensity(A) measurements monitored correctly from the application e3tapp.com? When you remove the power several times from the device, does it continue to connect with the platform?

Table 1: Description of the components.

3.3.12.2 Deliverables and version overview

The following table contains a deliverable list which need to be signed of before SAT testing commences.

ID	Description	Check
Documents		
1	Validation and Test reports of the IoT system components	
2	Validation and Test reports of the Pilot system components	
Hardware		
3	E3t city Devices	
Tools		
4	Electronic testing tools.	
5	E3t City Platform	

Table 92: Deliverable checklist

The following table shows the software components and version of which the system release version 1.0 consists of.

ID	Description	Version	Check
IoT Physical Gateway			
1	E3t city devices	V3.14	
IoT Virtual Gateway			
2	E3t city platform	V2.3.4	
Universal container			
3	E3t city REST API	V3.2.1	

Table 93: Component version overview

3.3.12.3 Requirements, scenarios and use cases

The following table provides an overview of the relation between the requirements and the test(s) that validate their implementation.

ID	Description	Covered by
Functionality		
11	Addressability and reachability	T_03
20	Real time support	T_03
21	Real time output	T_03
25	Remote programming of devices	T_02
26	Remote device control	T_02
API		
243	Gateway access API	T_02
Interoperability		
226	API for network services	T_02
Operational		
96	Enable (automated or semi-automated) linking of relevant data sources	T_03
204	Support smart network resources allocation in heterogeneous wireless sensor networks	T_03
Security		
98	Data provenance	T_03
27	System security	T_01

95	Robustness, resilience and availability	T_01

Table 94: Requirements vs. test mapping

The following table provides an overview of the relation between the scenarios and the test(s) that validate their implementation.

ID	Scenario name	Covered by
6	Dynamic lighting in the port	T_01 & T_02
32	Third party developer using INTER-FW to access data from two different platforms	T_02

Table 95: Scenario vs test mapping

3.3.12.4 Test environment

Introduction

To test the functionality of the integrated E3T-PDI in combination with the IoT framework representative test hooks in the system are needed. This chapter will describe this environment and the used hardware, software, tools and platforms.

Test environment

To understand our test, it is necessary to understand how solution works.

We provide control and management through our devices. These devices are controlled by users through our platform implemented in the cloud, and they can control them and receive information through web or mobile apps.

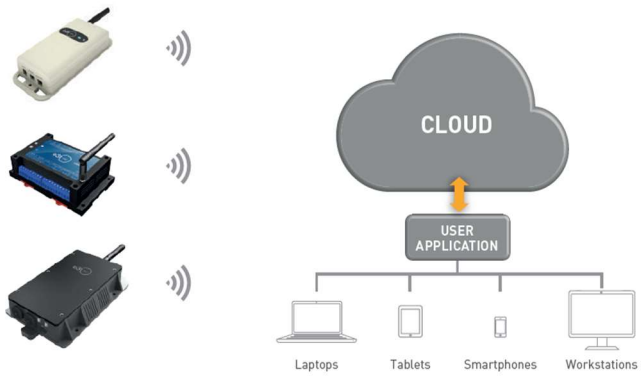


Figure 111: Diagram of solution e3t.

These applications have user level systems so that only users with administration level can manipulate the system configuration.

They also include security measures to prevent the sending of the same orders repeatedly in a short period of time to block the system.

Also, we have an API/REST to offer our clients, where they can implement their own platform or connect our devices to yours.

Next in the following figure 1, we can see a diagram of how the e3tsolution works:

In case of loss connection with the cloud, the devices would continue operating with the previously established parameters.

Our testing phase it consists of the three parts:

- Test power and electrical connectivity.
- Test connectivity to our platform.
- Test the correct reception of data and control our devices through the platform.

Should any of these test fail, the repair should be attempted in the place and, if this is not possible, the equipment will be exchanged for one of the reserve ones.

Test tools, hooks and probes

TS_01 Test setup electrical

We will connect the equipment to the power supply of the place to validate the correct supply of this and that enough signal is available to connect with the CLOUD.

TT_01 Test tool electrical

In relation with the tools necessary for the hardware test, we use the usual electronic testing tool, like Multimeter or oscilloscope.

TP_01 Test probe electrical

The device is designed internally to self-test, once it has been fed and later when it has been connected to our platform, we will receive the data of this test.

TS_02 Test setup connectivity.

Here we test the connectivity between the devices with the cloud, we need an internet connection and the only thing that must be sure that the equipment is connected to the platform, for it the blue led must be fixed. In case this did not happen, the equipment would not be connected to the platform and this could be due to a lack of signal in the place or a failure in the equipment.

TT_02 Test tool connectivity.

The only tool is our platform.

TS_03 Test setup correct interpretation of commands

For this test the only thing that must be sure that the equipment is connected to the platform, for I the blue led must be fixed.

TT_03 Test tool correct interpretation of commands

The only tool is our platform.

TH_03 Test hook correct interpretation of commands

In the last test we probe the response of the device, we sent several commands to the device through the platform, and we check that the device sends measurements correctly.

We follow the next checklist in the verification:

#	Test	Outcome
1	If possible activate the AC OUT 110-230 V from the application (e3tapp.com)?	Pass / Fail
2	The device executes the orders of regulation 0-10 V from the application (e3tapp.com)?	Pass / Fail
3	The six inputs analog/digital for the sensors works?	Pass / Fail
4	The six relay control outputs work by activating them from the application (e3tapp.com)?	Pass / Fail
	Communication Wifi 802.1$^{B/G/N}$ 2.4Ghz	
1	The device connects to the router configured in the application e3tapp.com?	Pass / Fail
2	The device connects correctly with e3tcity cloud?	Pass / Fail
	Single-Phase consumption measures	
1	Are the TOTAL POWER ACTIVE (W) measurements monitored correctly from the application e3tapp.com?	Pass / Fail
2	Are the TOTAL POWER REACTIVE (VAr) measurements monitored correctly from the application e3tapp.com?	Pass / Fail
3	Are the Voltage(V) and Intensity(A) measurements monitored correctly from the application e3tapp.com?	Pass / Fail
	Three-Phase consumption measures	Pass / Fail
1	Are the TOTAL POWER ACTIVE (W) in PHASE1 measurements monitored correctly from the application e3tapp.com?	Pass / Fail
2	Are the TOTAL POWER ACTIVE (W) in PHASE2 measurements monitored correctly from the application e3tapp.com?	Pass / Fail
3	Are the TOTAL POWER ACTIVE (W) PHASE3 measurements monitored correctly from the application e3tapp.com?	Pass / Fail
4	Are the TOTAL POWER REACTIVE (VAr) PHASE1 measurements monitored correctly from the application e3tapp.com?	Pass / Fail
5	Are the TOTAL POWER REACTIVE (VAr) PHASE2 measurements monitored correctly from the application e3tapp.com?	Pass / Fail
6	Are the TOTAL POWER REACTIVE (VAr) PHASE3 measurements monitored correctly from the application e3tapp.com?	Pass / Fail
7	Are the POWER FACTOR PHASE1 measurements monitored correctly from the application e3tapp.com?	Pass / Fail
8	Are the POWER FACTOR PHASE2 measurements monitored correctly from the application e3tapp.com?	Pass / Fail
9	Are the POWER FACTOR PHASE measurements monitored correctly from the application e3tapp.com?	Pass / Fail
10	Are the TOTAL POWER ACTIVE (W) TOTAL measurements monitored correctly from the application e3tapp.com?	Pass / Fail
11	Are the TOTAL POWER REACTIVE (VAr) in TOTAL measurements monitored correctly from the application e3tapp.com?	Pass / Fail
12	Are the POWER FACTOR TOTAL measurements monitored correctly from the application e3tapp.com?	Pass / Fail
13	Are the Voltage(V) and Intensity(A) PHASES measurements monitored correctly from the application e3tapp.com?	Pass / Fail

TH_03 Test probe correct interpretation of commands

The platform shows us in real time if our commands are being interpreted correctly in real time.

TT_04 Test tool robustness.

We need our platform and one auxiliary battery.

TS_04 Test setup robustness

For this test we need that our device is connected to the platform, we will identify it with the fixed blue led to analyze the response of the device at all times.

TH_04 Test hook robustness

We follow the next checklist in the verification:

#	Test	Outcome
1	Is the equipment capable of hold on an erroneous power supply due to the output?	Pass / Fail
2	Is the equipment not blocked when receiving a repeated press of the power on command?	Pass / Fail
3	Does the equipment work without connecting the antenna, although it is recommended that it is always connected?	Pass / Fail
4	Does withdrawing the power repeatedly blocks the device without being able to turn it on again?	Pass / Fail

3.3.12.5 Test description

Scenario Testing power-on, power-off and reception of measurements of a luminaire.

In the following, we detail a scenario where we will check the operation of a system that must control the lighting of luminaire and receive measures from it.

T1.1.1 Test electrical

ID	T1.1.1
Test	This test is to check the correct manufacture of the equipment
Type	Physical
Setup	TS_01
Start	Connected to a battery
Req.	Battery and tools
Input	
Output	
Logs	Our database
Outcome	Pass / Fail

T1.1.2 Test connectivity

ID	T1.1.2
Test	Probe the connection with de cloud
Type	Cloud
Setup	TS_02
Start	Connected to the platform
Req.	
Input	Commands from the platform
Output	Error report
Logs	Our database
Outcome	Pass / Fail

T1.1.3 Test correct interpretation of commands

ID	T1.1.3
Test	Probe the response of the device
Type	Cloud
Setup	TS_03
Start	Connected to the platform
Req.	
Input	Commands from the platform
Output	Correct response to commands
Logs	Our database
Outcome	Pass / Fail

T1.1.4 Test repeated loss of power supply

ID	T1.1.3
Test	Probe the response of the device
Type	Cloud
Setup	TS_04
Start	Connect and disconnect repeatedly power supply to equipment.
Req.	
Input	
Output	The equipment responds well and turn on.
Logs	Our database
Outcome	Pass / Fail

T1.1.5 Test block by repeatedly pulsation.

ID	T1.1.4
Test	Probe the response of the device when you activate several times.
Type	Cloud
Setup	TS_04

Start	Connected to the platform
Req.	
Input	Commands from the platform
Output	Correct response to commands
Logs	Our database
Outcome	Pass / Fail

T1.1.6 Test wrong electrical supply

ID	**T1.1.4**
Test	Probe the response of the device when you supply de equipment by the output.
Type	
Setup	TS_04
Start	Connected to the platform
Req.	
Input	Electrical Supply
Output	Correct response of the equipment.
Logs	Our database
Outcome	Pass / Fail

3.3.12.6 Test outcome overview

Test outcome

The following table will provide an overview of the test result of all the performed tests in this SAT.

Test	Description	Outcome
T.01	Test electrical	Pass / Fail
T.02	Test connectivity.	Pass / Fail
T.03	Test correct interpretation of commands	Pass / Fail
T.04	Test repeated loss of power supply	Pass / Fail
T.05	Test block by repeatedly pulsation	Pass / Fail
T.06	Test wrong electrical supply	Pass / Fail
SAT Outcome		**Pass / Fail**

Table 96: Test outcome overview

3.3.12.7 Integration ethics and security

Introduction

This section addresses recommendations and proposed solutions to the ethical and security points of the integration.

E3T-PDI

In terms of security, we are able to detect if our equipment has been manipulated, since from our platform we can read internal parameters of the devices and detect errors.

Furthermore, all our communication between devices and the platform is secure, since the connection is encrypted, also all our control commands have a token generated by us, to avoid external manipulations.

In the ethical part, this project by its design is completely ethical, with the demonstration that the connection between platforms is possible, many possibilities open up to be able to interconnect many parts of a city.

With our system, to be able to control the consumption, luminosity and a better management of on and off we can have a lower energy consumption and therefore contaminate less.

4 Open Call Third Parties Evaluation

This section will include the outcome of the final evaluation of the open call third parties which will take place in October 2018 and will be made available as an Annex to this document. The first evaluation was conducted in June 2017 for the small contributions and in July 2017 for the large contributions. Also a midterm evaluation was performed in June 2018 with successful results. The midterm evaluation covered the period till December 2017, and this final evaluation will cover the period from June 2018 until the finalization of the collaborations.

5 Conclusions

The document has described Site Acceptance Test (SAT) and final integration activities for the pilots included in the proposal, i.e. INTER-LogP and INTER-Health, as well as the collaborations with the third parties included during INTER-IoT OpenCall.

Following the determined industrial approach, the integration has been structured in three stages, first one was included in D6.1, in which the pilots, use cases and technologies to be used were described; the FAT documents of all collaborations from third parties was included in D6.2 and, in this document, D6.3 provides information about Site Acceptance Test (SAT) documents.

In complementary documents, as the ones presented under WP7 framework, the validation of the integration process and the different KPIs specified is provided together with the testing phase and its results.

The SAT documents provided by the internal pilots and by the third party collaborators, have followed a common template in which they provided the information related with stakeholder's specifications and requirements. Commonly, SAT protocol is tightly related to the FAT protocol and also entails deep inspection of the system and provides documented evidence that a piece of equipment, system, or integrated process that has been delivered to the end user or stakeholder has not been affected in the installation and has been adequately tested at the end user's facility and performed to the end user's expectations after deployment. The SAT document will complete a series of verifications to ensure that what was defined has been supplied by verifying all previous settled specifications. Control system verification will also be executed and documented in the SAT protocol. Performance testing will also be outlined and the results documented to provide assurance that the system being tested meets the end user's expectations and requirements and can move on to the next stage of qualifications required to validate the system.

The information is maintained in a separate document per pilot, so the working documents are fourteen separate files that for commodity have been integrated and adapted in a single document.

Each SAT documents expose the Test Strategy and approach where each third party collaborator presents the program they followed to test the collaboration, the *System Description* where a detailed explanation of the components and functionality of the integrated system is depicted, the *Integration with the INTER-IoT Component* where guide section of the followed steps is presented, *Deliverables and Version Overview* where the list of documents related with the participation of the new Open Call partner is listed, *Requirements, Scenarios and Use Cases* where each partner collaboration' presents the requirements of their system and the requirements, scenarios and use cases derived from the junction with INTER-IoT specific component, *Test Environment* section describes the stack of test that the integrated system has to pass successfully in order to be compliant with the requirements previously defined and the description of the collaboration in the first proposal, *Test Outcome Overview* where the results of the aforementioned test is detailed and discussed and finally, *Integration Ethics and Security* that is a section dedicated to security issues that arise during the integration and the ethical problems or issues that comes related with the incorporation of the new collaboration component onto the INTER-IoT system.

ANNEX A

SAT test execution sign-off

The following table shall contain the test attendance list. Each individual in this list will sign of on his/her presence, the deliverables and the outcome of the tests.

Test execution date			
Test execution time			
System version	1.0		
Name	**Company**	**Function title**	**Signature**

Table 97: SAT test execution sing-off